MOTOROLA

# PowerPC™ MPC821

## Portable Systems Microprocessor User’s Manual

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

❊

*In Memory of*
*Roland Hernandez*

❊

# TABLE OF CONTENTS

Paragraph Number | Title | Page Number

**Section 1**
**Overview**

**Section 2**
**External Signals**

**Section 3**
**Memory Map**

**Section 4**
**Reset**

# TABLE OF CONTENTS (Continued)

## Section 5
## Clocks and Power Control

## Section 6
## Core

# TABLE OF CONTENTS (Continued)

## Section 7
## PowerPC Architecture Compliance

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

## Section 8
## Instruction Execution Timing

## Section 9
## Instruction Cache

# TABLE OF CONTENTS (Continued)

## Section 10
## Data Cache

## Section 11
## Memory Management Unit

# TABLE OF CONTENTS (Continued)

## Section 12
## System Interface Unit

# TABLE OF CONTENTS (Continued)

## Section 13
## External Bus Interface

# TABLE OF CONTENTS (Continued)

**Section 14**
**Endian Modes**

**Section 15**
**Memory Controller**

# TABLE OF CONTENTS (Continued)

Paragraph Number | Title | Page Number

## Section 16
## Communication Processor Module

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

# TABLE OF CONTENTS (Continued)

| Paragraph Number | Title | Page Number |
|---|---|---|

## Section 17
## PCMCIA Interface

## Section 18
## LCD Controller

# TABLE OF CONTENTS (Continued)

**Section 19**
**Development Support**

# TABLE OF CONTENTS (Continued)

## Section 20
## IEEE 1149.1 Test Access Port

## Section 21
## Electrical Characteristics

## Section 22
## CPM Electrical Characteristics

## Section 23
## Mechanical Data and Ordering Information

## Section 24
## Terminology Glossary

## AppendixA
## Serial Communication Performance

## AppendixB
## Quick Reference Guide to MPC821 Registers

## Index

# SECTION 1
# OVERVIEW

The MPC821 Portable Systems Microprocessor 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 high performance and portable communications systems where lower power is essential. Unless otherwise specified, the MPC821 Portable Systems Microprocessor will be referred to simply as the MPC821 in this manual.

The MPC821 is a PowerPC-based derivative of Motorola's MC68360 Quad Integrated Communications Controller (QUICC™), hereafter referred to as the QUICC. The CPU on the MPC821 is a 32-bit implementation with memory management units (MMUs) and instruction and data caches. The communications processor module (CPM) of the MC68360 QUICC has been enhanced in the functionality of the two serial communications controllers (SCCs) channels, the two serial management channels (SMCs), the serial peripheral interface (SPI), and the interprocessor-integrated controller ($I^2C$) channel, particularly in the area of wireless communications protocols and busses necessary for compatibility with personal digital assistant (PDA) and personal intelligent communicator (PIC) implementations. Moderate to high digital signal processing (DSP) functionality has been added to the CPM embedded RISC microcontroller. The memory controller has been enhanced for high performance memories and for new dynamic random access memories (DRAMs) used in portable systems. The system functionality in the MPC821 is also completed by a new parallel I/O capability (PCMCIA x2), a display capability (LCD controller), and a real-time clock which adds to the functionality of the QUICC.

The purpose of this document is to describe the operation of the MPC821's functionality with concentration on the I/O functions. Additional details on the MPC821 can be found in the PowerPC architectural specifications.

## 1.1 FEATURES

The following list summarizes the important features of the MPC821:

- PowerPC single-issue integer core
- Precise exception model
- Extensive system development support
  - On-chip watchpoints and breakpoints
  - Program flow tracking
  - On-chip emulation (OnCE) development interface
- High performance (52K (Dhrystone 2.1) MIPS @50 MHz, 3.3 V, 1.3 Watts total power)

- Low power (<241 mW @25 MHz, 2.2 V internal, 3.3 V I/O - core, caches, MMUs, I/O)
- MPC821 PowerPC system interface, including a periodic interrupt timer, a bus monitor, and clocks
- Fully static design
- Four major power saving modes
  - On, doze, nap, and sleep
  - Gear mode additionally provides a subset of on mode
- 357 OMPAC ball grid array packaging
- 32-Bit address and data busses
  - Bus supports multiple master designs
  - Four-beat transfer bursts, two-clock minimum bus transactions
  - Dynamic bus sizing controlled by on-chip memory controller
  - Supports data parity
  - Tolerates 5 V inputs, provides 3.3 V outputs
- Flexible memory management
  - 32-entry, fully associative instruction translation lookaside buffers
  - 32-entry, fully associative data translation lookaside buffers
  - 4-kbyte, 16-kbyte, 512-kbyte, or 8-Mbyte page size support
  - 1-kbyte protection granularity
  - Support for multiple protection groups and tasks
  - Attribute support for trapping, write-through, cache-inhibit, and memory-mapped I/O
  - Supports software tablewalk
- 4-kbyte physical address, two-way, set-associative data cache
  - Single-cycle access on hit
  - 4-word line size, burst fill, least recently used replacement
  - Cache lockable online granularity
  - Read capability of all tags and attributes provided for debugging purposes
- 4-kbyte physical address, two-way, set-associative instruction cache
  - Single-cycle access on hit
  - Four-word line size, burst fill, least recently used replacement
  - Cache lockable online granularity
  - Cache control supports PowerPC invalidate instruction
  - Cache inhibit supported for the entire cache or per memory management unit page in conjunction with memory management logic
  - Read capability of all tags and attributes provided for debugging purposes
- Eight-bank memory controller
  - Glueless interface to SRAM, DRAM, EPROM, FLASH and other peripherals
  - Byte write enables and selectable parity generation
  - 32-bit address decodes with bit masks

- System interface unit
  - — Clock synthesizer
  - — Power management
  - — Reset controller
  - — PowerPC decrementer and time base
  - — Real-time clock register
  - — Periodic interrupt timer
  - — Hardware bus monitor and software watchdog timer
  - — IEEE 1149.1 JTAG test access port
- Communications processor module
  - — Embedded 32-bit RISC controller architecture for flexible I/O
  - — Interfaces to PowerPC core through on-chip dual-port RAM and virtual DMA channel controller
  - — Continuous mode transmission and reception on all serial channels
  - — Serial DMA channels for reception and transmission on all serial channels
  - — Parallel I/O registers with open-drain and interrupt capability
  - — Can implement memory-to-memory and memory-to-I/O transfers with virtual DMA functionality
  - — Protocols supported by ROM or download microcode and the two hardware serial communications controller channels include, but are not limited to the digital portions of:
    - Ethernet/IEEE 802.3 CS/CDMA
    - HDLC/SDLC and HDLC bus
    - AppleTalk
    - Signaling system #7
    - Universal asynchronous receiver transmitter (UART)
    - Synchronous UART
    - Binary synchronous communications
    - Totally transparent
    - Totally transparent with CRC
    - Profibus (RAM microcode option)
    - Asynchronous HDLC
    - DDCMP
    - V.14 (RAM microcode option)
    - X.21 (RAM microcode option)
    - V.32bis datapump filters
    - IrDA serial infrared
    - Basic rate ISDN (BRI) in conjunction with SMC channels
    - Primary rate ISDN
  - — Two hardware serial communications controller channels supporting the above protocols
  - — Two hardware serial management channels
    - Provide management for BRI devices as general circuit interface controller in time division multiplexed channels
    - Transparent and low-speed UART operation
  - — Hardware serial peripheral interface

    - Multimaster, master, and slave modes
  - I$^2$C (microwire compatible) interface
    - Supports master and slave modes
  - Time-slot assigner
    - Supports one or two TDMA channels
    - Bit or byte resolution
    - Independent transmit and receive routing, frame synchronization, and dynamic clocking modification ability
    - Software-configurable for internal interconnection of CPM serial channels
    - Typically implements T1, CEPT, PCM highway, ISDN basic rate, ISDN primary rate and user-defined TDMA serial interfaces
  - Parallel interface port supports Centronics interfaces and chip-to-chip interconnection
  - Four independent baud rate generators and four input clock pins for supplying clocks to SMC and SCC serial channels
  - Four independent 16-bit timers which can be interconnected as two 32-bit timers
- LCD interface controller
  - 1, 2, or 4 bits per pixel gray mode using advanced FRC algorithm
  - 4, 8 or 9 bits parallel output to the LCD
  - Nonsplit or vertically-split screen support
  - Data for splits: 2+2 or 4+4 parallel bits (2+2 means 2 bits for low screen and 2 bits for high screen in parallel)
  - TFT / RGB output drives advanced buffer LCD driver chips
  - Built-in 256 entries color RAM
  - Maps each 4-bit color code to one of 512 colors. Each color is defined by 3-bit code for red, green, and blue.
  - Maps each 2-bit gray level code to one of eight gray levels
  - Programmable wait time between lines and frames
  - Panel voltage control–programmable LVDAC through duty-cycle, for contrast adjustments. Implemented by using another existing on-chip timer.
  - Programmable polarity for all LCD interface signals
- Two PC card (PCMCIA 2.1) master interface

## 1.2 MPC821 ARCHITECTURE OVERVIEW

The MPC821 is functionally composed of three major blocks:

- A 32-bit PowerPC core with MMUs and caches
- A system interface unit
- A communications processor module

Figure 1-1 provides a block diagram view of the MPC821.

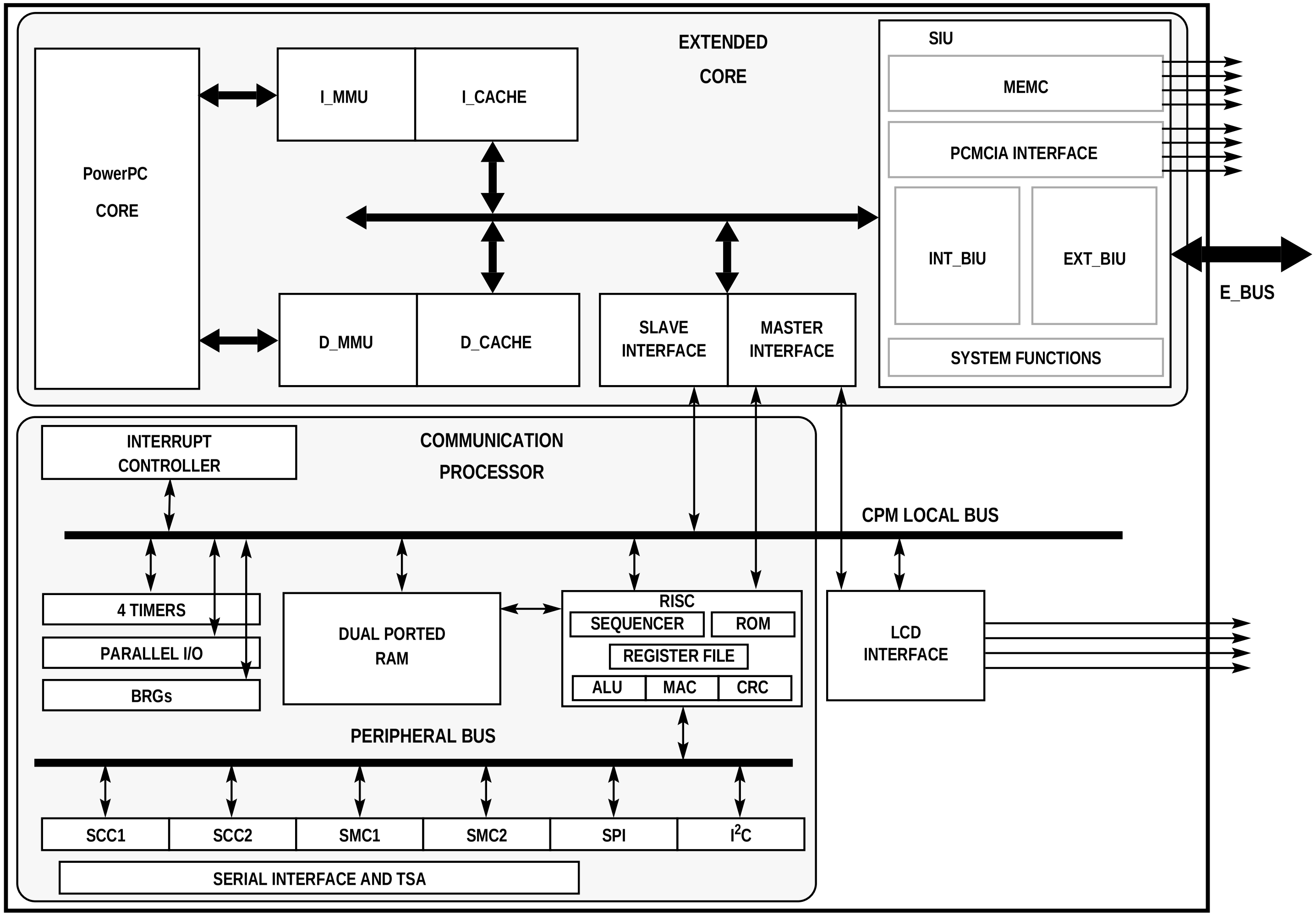


Figure 1-1. MPC821 Block Diagram

## 1.3 UPGRADING DESIGNS FROM THE MC68360 QUICC

Since the MPC821 CPM block uses the same modules as the MC68360 QUICC, microcode routines and chip drivers developed for the QUICC can be readily adapted to the MPC821.

## 1.4 MPC821 GLUELESS SYSTEM DESIGN

The system design approach of the MPC821 provides glueless interfaces to memories, serial transceivers, and bus buffers in a number of applications.

# SECTION 2
# EXTERNAL SIGNALS

## 2.1 SIGNALS DESCRIPTION

This section contains brief descriptions of the MPC821input and output signals in their functional groups as illustrated in Figure 2-1.

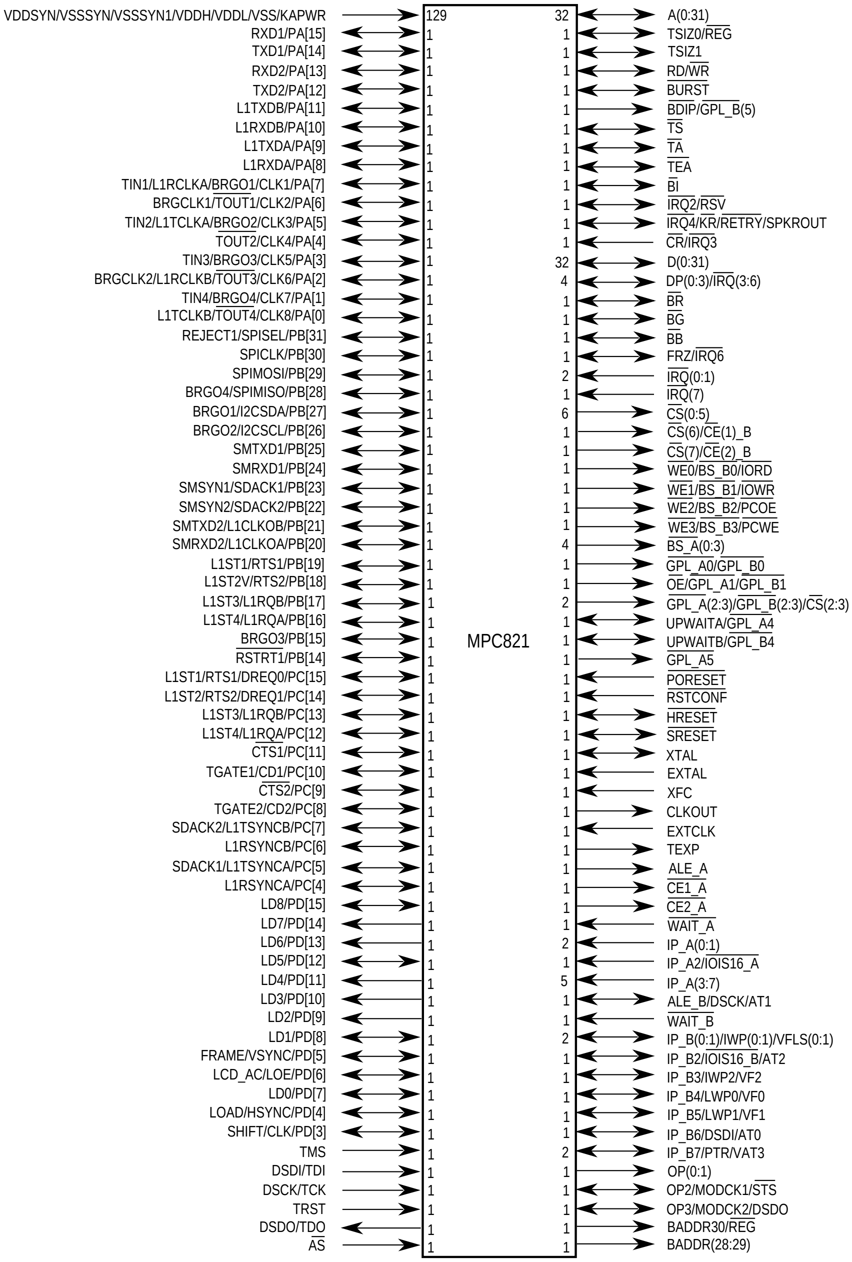


**Figure 2-1. MPC821 External Signals**

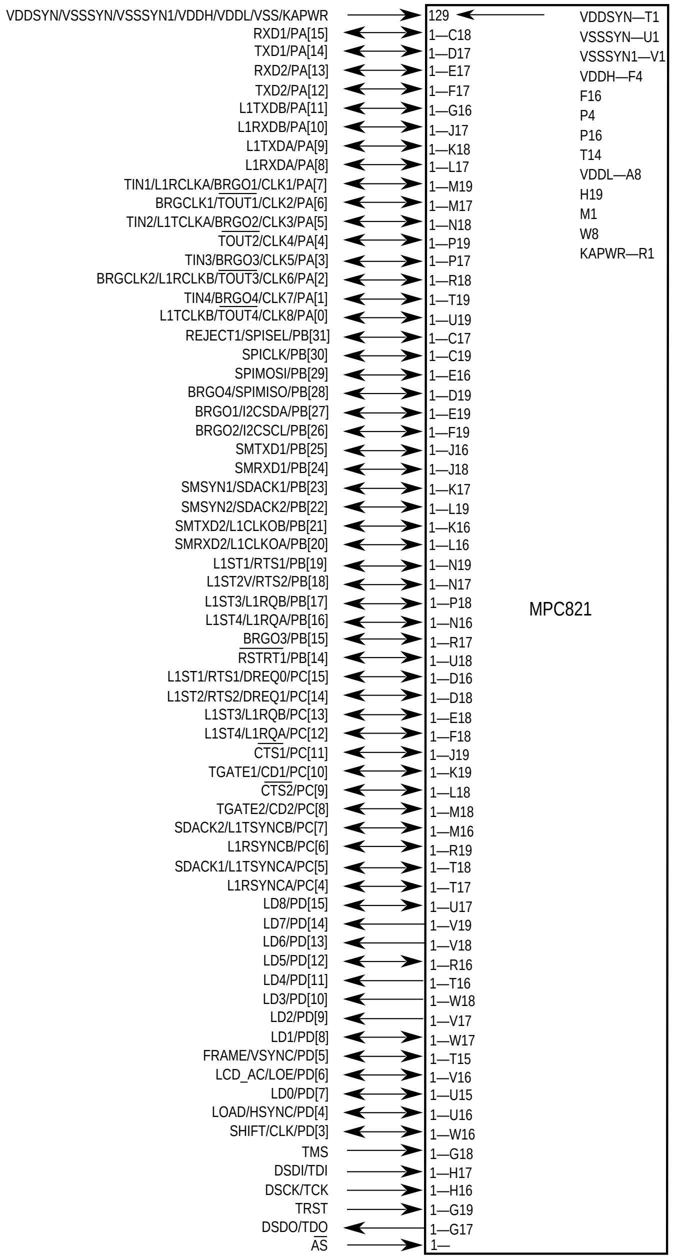


**Figure 2-2. Signals and Pin Numbers (Part 1)**

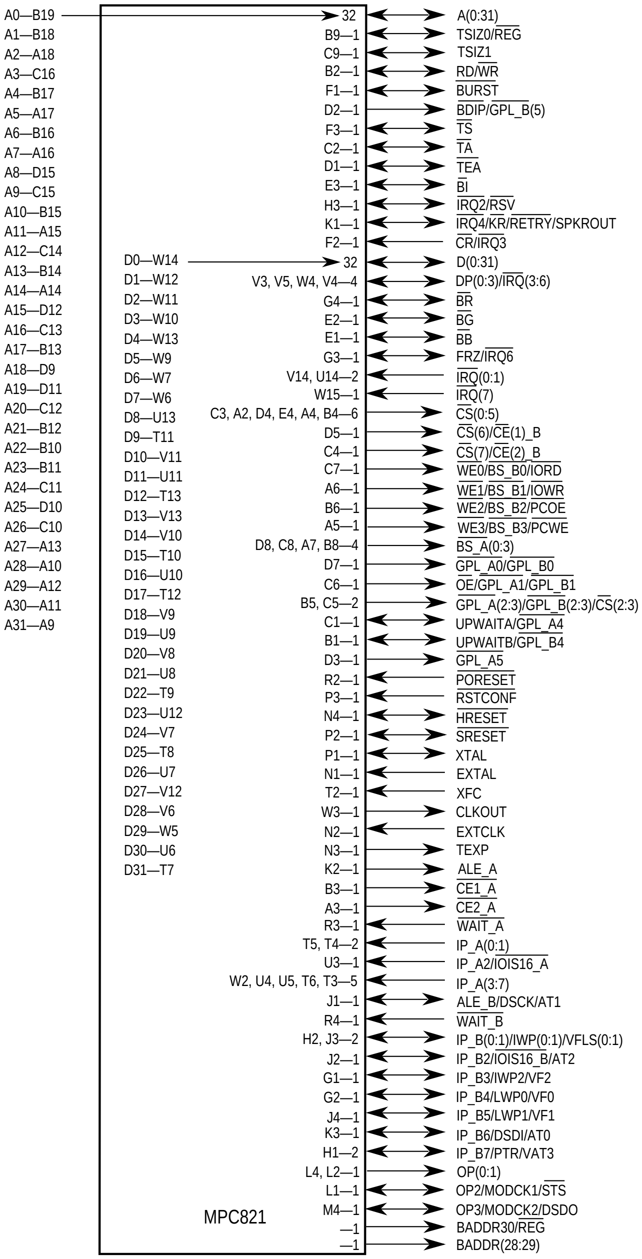


Figure 2-3. Signals and Pin Numbers (Part 2)

### 2.1.1 System Bus Signals

The MPC821 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 following input and output signals are identified by their mnemonic name and each signal's pin number can be found in Figure 2-1.

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| A(0-31) | * | **Address Bus—**This bidirectional three-state bus provides the address for the current bus cycle. A0 is the most-significant signal for this bus. The bus is output when an internal master on the MPC821 initiates a transaction on the external bus.<br>The bus is input when an external master initiates a transaction on the bus and it is sampled internally to allow the memory controller/PCMCIA interface to control the accessed slave device. |
| TSIZ0<br>$\overline{\text{REG}}$ | B9 | **Transfer Size 0**—When accessing a slave in the external bus, this three-state signal is used (together with TSIZ1) by the bus master to indicate the number of operand bytes waiting to be transferred in the current bus cycle.<br>This signal is input when an external master initiates a transaction on the bus and it is sampled internally to allow the memory controller/PCMCIA interface to control the accessed slave device.<br>**Register**—When the access is initiated by an internal master to a slave under control of the PCMCIA interface, this signal is output to indicate which space in the PCMCIA card is currently accessed. |
| TSIZ1 | C9 | **Transfer Size 1—**This three-state signal is used (with TSIZ0) by the bus master to indicate the number of operand bytes waiting to be transferred in the current bus cycle.<br>This signal is driven by the MPC821 when it is the owner of the bus. This signal is input when an external master initiates a transaction on the bus and it is sampled internally to allow the memory controller/PCMCIA interface to control the accessed slave device. |
| RD/$\overline{\text{WR}}$ | B2 | **Read Write**—This three-state signal is driven by the bus master to indicate the direction of the bus's data transfer. A logic one indicates a read from a slave device and a logic zero indicates a write to a slave device.<br>This signal is driven by the MPC821 when it is the owner of the bus. This signal is input when an external master initiates a transaction on the bus and is sampled internally to allow the memory controller/PCMCIA interface to control the accessed slave device. |
| $\overline{\text{BURST}}$ | F1 | **Burst Transaction**—This three-state signal is driven by the bus master to indicate that the current initiated transfer is a burst one.<br>This signal is driven by the MPC821 when it is the owner of the bus. This signal is input when an external master initiates a transaction on the bus; this signal and is sampled internally to allow the memory controller/PCMCIA interface to control the accessed slave device. |
| $\overline{\text{BDIP}}$<br>$\overline{\text{GPL\_B5}}$ | D2 | **Burst Data in Progress**—When accessing a slave device in the external bus, the master on the bus asserts this signal to indicate that the data beat in front of the current one is the one requested by the master. This signal is negated prior to the expected last data beat of the burst transfer.<br>**General-Purpose Line B5**—This line is used by the memory controller when the user programmable machine B (UPMB) takes control of the slave access. . |
| $\overline{\text{TS}}$ | F3 | **Transfer Start—**This three-state signal is asserted by the bus master to indicate the start of a bus cycle that transfers data to or from a slave device.<br>This signal is driven by the master only when it has gained the ownership of the bus. Every master should negate this signal before the bus relinquish. A pull-up resistor should be connected to this line to prevent a slave device from detecting a spurious bus accessing when no master is taking ownership of the bus.<br>This signal is sampled by the MPC821 when it is not the owner of the external bus to allow the memory controller/PCMCIA interface to control the accessed slave device. It indicates that an external synchronous master initiated a transaction. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| $\overline{\text{TA}}$ | C2 | **Transfer Acknowledge**—This bidirectional three-state line indicates that the slave device addressed in the current transaction has accepted the data transferred by the master (write) or has driven the data bus with valid data (read). The line behaves as an output when the PCMCIA interface or the memory controller takes control of the transaction. The only exception occurs when the memory controller is controlling the slave access by means of the GPCM and the corresponding option register is instructed to wait for an external assertion of the transfer acknowledge line. Every slave device should negate the $\overline{\text{TA}}$ signal after the end of the transaction and immediately three-state it to avoid contentions on the line if a new transfer is initiated addressing other slave devices. A pull-up resistor should be connected to this line to avoid a master device detecting the assertion of this line when no slave is addressed in a transfer or when the address detection for the slave addressed is slow. |
| $\overline{\text{TEA}}$ | D1 | **Transfer Error Acknowledge**—This open-drain signal indicates that a bus error occurred in the current transaction. It is driven asserted by the MPC821 when the bus monitor does not detect a bus cycle termination within a reasonable amount of time. The assertion of $\overline{\text{TEA}}$ causes the termination of the current bus cycle thus, ignoring the state of $\overline{\text{TA}}$. |
| $\overline{\text{BI}}$ | E3 | **Burst Inhibit**—This bidirectional three-state line indicates that the slave device addressed in the current burst transaction is unable to support burst transfers. The line behaves as an output when the PCMCIA interface or the memory controller takes control of the transaction. When the MPC821 drives out the signal for a specific transaction, it asserts or negates $\overline{\text{BI}}$ during the transaction according to the value specified by the user in the appropriate control registers. It negates the signal after the end of the transaction and immediately three-states it to avoid contentions if a new transfer is initiated addressing other slave devices. |
| $\overline{\text{RSV}}$<br>$\overline{\text{IRQ2}}$ | H3 | **Reservation**—This three-state line is output by the MPC821 in conjunction with the address bus to indicate that the internal core initiated a transfer as a result of STWCX or a LWARX instruction.<br>**Interrupt Request 2**—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. |
| $\overline{\text{KR/RETRY}}$<br>$\overline{\text{IRQ4}}$<br>SPKROUT | K1 | **Kill Reservation**—This input is used as a part of the storage reservation protocol, when the **MPC821** initiated a transaction as the result of a STWCX instruction.<br>**Retry**—This input is used by the slave device to indicate that it is unable to accept the transaction. The MPC821 must relinquish ownership of the bus and initiate the transaction again after winning in the bus arbitration.<br>**Interrupt Request 4**—This input is one of the eight external lines that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical and of this line (if defined to function as $\overline{\text{IRQ4}}$) and the DP1/$\overline{\text{IRQ4}}$ (if defined to function as $\overline{\text{IRQ4}}$).<br>**SPKROUT**—This output signal provides a digital audio waveform to be driven to the system speaker. |
| $\overline{\text{CR}}$<br>$\overline{\text{IRQ3}}$ | F2 | **Cancel Reservation**—This input is used as a part of the storage reservation protocol.<br>**Interrupt Request 3**—This input is one of the eight external lines that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical and of this line (if defined to function as $\overline{\text{IRQ3}}$) and the DP0/$\overline{\text{IRQ3}}$ if defined to function as $\overline{\text{IRQ3}}$. |
| D(0:31) | * | **Data Bus**—This bidirectional three-state bus provides the general-purpose data path between the MPC821 and all other devices. Although the data path is a maximum of 32 bits wide, it can be dynamically sized to support 8-, 16-, or 32-bit transfers. D0 is the MSB of the data bus. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| DP0<br>IRQ3 | V3 | **Data Parity 0**—This bidirectional three-state line provides parity generation and checking for the data bus lane D(0:7) by transferring to a slave device initiated by the MPC821. 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.<br>**Interrupt Request 3**—This input is one of the eight external lines that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical and of this line (if defined to function as IRQ3) and the CR/IRQ3 if defined to function as IRQ3. |
| DP1<br>IRQ4 | V5 | **Data Parity 1**—This bidirectional three-state line provides parity generation and checking for the data bus lane D(8:15) by transferring to a slave device initiated by the MPC821. 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.<br>**Interrupt Request 4**—This input is one of the eight external lines that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical and of this line (if defined to function as IRQ4) and the KR/IRQ4/SPKROUT if defined to function as IRQ4. |
| DP2<br>IRQ5 | W4 | **Data Parity 2**—This bidirectional three-state line provides parity generation and checking for the data bus lane D(16:23) by transferring to a slave device initiated by the MPC821. 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.<br>**Interrupt Request 5**—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. |
| DP3<br>IRQ6 | V4 | **Data Parity 3**—This bidirectional three-state line provides parity generation and checking for the data bus lane D(24:31) by transferring to a slave device initiated by the MPC821. 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.<br>**Interrupt Request 6**—This input is one of the eight external lines that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical and of this line (if defined to function as IRQ6) and the FRZ/IRQ6 if defined to function as IRQ6. |
| BR | G4 | **Bus Request**—This bidirectional signal is asserted low when a possible master is requesting ownership of the bus. When the MPC821 is configured to work with the internal arbiter, this signal is configured as an input. When the MPC821 is configured to work with an external arbiter, this signal is configured as an output and asserted every time a new transaction is intended to be initiated and no parking on the bus is granted. |
| BG | E2 | **Bus Grant**—This bidirectional signal is asserted low when the arbiter of the external bus grants the specific master ownership of the bus. When the MPC821 is configured to work with the internal arbiter, this signal is configured as an output and asserted every time the external master asserts the BG line and it's priority request is higher than any of the internal sources requiring the initiation of a bus transfer. However, when the MPC821 is configured to work with an external arbiter, this signal is configured as an input. |
| BB | E1 | **Bus Busy**—This bidirectional signal is asserted low by a master to show that it owns the bus. The MPC821 asserts this line after the bus arbiter grants it bus ownership and the BB is negated. |
| FRZ<br>IRQ6 | G3 | **Freeze**—This output signal is asserted to indicate that the internal core is in debug mode.<br>**Interrupt Request 6**—This input is one of the eight external lines that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical and of this line (if defined to function as IRQ6) and the DP3/IRQ6 if defined to function as IRQ6. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| IRQ0 | V14 | **Interrupt Request 0**—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. |
| $\overline{\text{IRQ1}}$ | U14 | **Interrupt Request 1**—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. |
| $\overline{\text{IRQ7}}$ | 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. |
| $\overline{\text{CS}}$(0:5) | C3, A2, D4, E4, A4, B4 | **Chip Select**—These output signals enable peripheral or memory devices at programmed addresses if they are appropriately defined in the memory controller. $\overline{\text{CS0}}$ can be configured to be the global chip-select for the boot device. |
| $\overline{\text{CS6}}$<br>$\overline{\text{CE1\_B}}$ | D5 | **Chip Select 6**—This output signal enables a peripheral or memory device at a programmed address if defined appropriately in the BR6 and OR6 in the memory controller.<br>**Card Enable 1 Slot B**—This output signal enables even byte transfers when accesses to the PCMCIA Slot B are handled under the control of the PCMCIA interface. |
| $\overline{\text{CS7}}$<br>$\overline{\text{CE2\_B}}$ | C4 | **Chip Select 7**—This output signal enables a peripheral or memory device at a programmed address if defined appropriately in the BR7 and OR7 in the memory controller.<br>**Card Enable 2 Slot B**—This output signal enables odd byte transfers when accesses to the PCMCIA Slot B are handled under the control of the PCMCIA interface. |
| $\overline{\text{WE0}}$<br>$\overline{\text{BS\_B0}}$<br>IORD | C7 | **Write Enable 0**—This output line is asserted when a write access to an external slave controlled by the GPCM in the memory controller is initiated by the MPC821. $\overline{\text{WE0}}$ is asserted if the data lane D(0:7) contains valid data to be stored by the slave device.<br>**Byte Select 0 on UPMB**—This output line is asserted under requirement of the UPMB in the memory controller, as programmed by the user. In a read or write transfer, the line is only asserted if the data lane D(0:7) contains valid data.<br>**IO Device Read**—This output line is asserted when the MPC821 initiates a read access to a region under the control of the PCMCIA interface. The signal is only asserted if the access is to a PC Card I/O space. |
| $\overline{\text{WE1}}$<br>$\overline{\text{BS\_B1}}$<br>IOWR | A6 | **Write Enable 1**—This output line is asserted when the MPC821 initiates a write access to an external slave controlled by the GPCM in the memory controller. $\overline{\text{WE1}}$ is asserted if the data lane D(8:15) contains valid data to be stored by the slave device.<br>**Byte Select 1 on UPMB**—This output line is asserted under requirement of the UPMB in the memory controller, as programmed by the user. In a read or write transfer, the line is only asserted if the data lane D(8:15) contains valid data.<br>**I/O Device Write**—This output line is asserted when the MPC821 initiates a write access to a region under control of the PCMCIA interface. The signal is only asserted if the access is to a PC Card I/O space. |
| $\overline{\text{WE2}}$<br>$\overline{\text{BS\_B2}}$<br>$\overline{\text{PCOE}}$ | B6 | **Write Enable 2**—This output line is asserted when the MPC821 initiates a write access to an external slave controlled by the GPCM in the memory controller. $\overline{\text{WE2}}$ is asserted if the data lane D(16:23) contains valid data to be stored by the slave device.<br>**Byte Select 2 on UPMB**—This output line is asserted under requirement of the UPMB in the memory controller, as programmed by the user. In a read or write transfer, the line is only asserted if the data lane D(16:23) contains valid data.<br>**PCMCIA Output Enable**—This output line is asserted when the MPC821 initiates a read access to a memory region under the control of the PCMCIA interface. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| WE3<br>BS_B3<br>PCWE | A5 | **Write Enable 3**—This output line is asserted when the MPC821 initiates a write access to an external slave controlled by the GPCM in the memory controller. WE3 is asserted if the data lane D(24:31) contains valid data to be stored by the slave device.<br>**Byte Select 3 on UPMB**—This output line is asserted under requirement of the UPMB in the memory controller, as programmed by the user. In a read or write transfer, the line is only asserted if the data lane D(24:31) contains valid data.<br>**PCMCIA Write Enable**—This output line is asserted when the MPC821 initiates a write access to a memory region under control of the PCMCIA interface. |
| BS_A(0:3) | D8, C8, A7, B8 | **Byte Select 0 to 3 on UPMA**—These output lines are asserted under requirement of the UPMB in the memory controller, as programmed by the user. In a read or write transfer, the lines are asserted only if their corresponding data lanes contain valid data:<br>BS_A0 for D(0:7), BS_A1 for D(8:15),<br>BS_A2 for D(16:23), BS_A3 for D(24:31) |
| GPL_A0<br>GPL_B0 | D7 | **General-Purpose Line 0 on UPMA**—This output line reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA).<br>**General-Purpose Line 0 on UPMB**—This output line reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB). |
| OE<br>GPL_A1<br>GPL_B1 | C6 | **Output Enable**—This output line is asserted when the MPC821 initiates a read access to an external slave controlled by the GPCM in the memory controller.<br>**General-Purpose Line 1on UPMA**—This output line reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA).<br>**General-Purpose Line 1 on UPMB**—This output line reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB). |
| GPL_A(2:3)<br>GPL_B(2:3)<br>CS(2:3) | B6, C5 | **General-Purpose Line 2 and 3 on UPMA**—These output lines reflect the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA).<br>**General-Purpose Line 2 and 3 on UPMB**—These output lines reflect the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB).<br>**Chip Select 2 and 3**—These output signals enable peripheral or memory devices at programmed addresses if they are appropriately defined in the memory controller. The double drive capability for CS2 and CS3 is independently defined for each signal in the SIUMCR. |
| UPWAITA<br>GPL_A4 | C1 | **User Programmable Machine Wait A**—This input line is sampled as a requirement by the user when an access to an external slave is controlled by the UPMA in the memory controller.<br>**General-Purpose Line 4 on UPMA**—This output line reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). |
| UPWAITB<br>GPL_B4 | B1 | **User Programmable Machine Wait B**—This input line is sampled as a requirement by the user when an access to an external slave is controlled by the UPMB in the memory controller.<br>**General-Purpose Line 4 on UPMB**—This output line reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB). |
| GPL_A5 | D3 | **General-Purpose Line 5 on UPMA**—This output line reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). This signal can also be controlled by the UPMB. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| PORESET | R2 | **Power on Reset**—When asserted, this input line causes the MPC821 to enter the power-on reset state. |
| RSTCONF | P3 | **Reset Configuration**—This input line is sampled by the MPC821 during the assertion of the HRESET signal. If the line is asserted, the configuration mode is sampled in the form of the hard reset configuration word driven on the data bus. When this line is negated, the default configuration mode is adopted by the MPC821. Notice that the initial base address of internal registers is determined in this sequence. |
| HRESET | N4 | **Hard Reset**—This open drain line, when asserted causes the MPC821to enter the hard reset state. |
| SRESET | P2 | **Soft Reset**—This open drain line, when asserted causes the MPC821to enter the soft reset state. |
| XTAL | P1 | This output line is one of the connections to an external crystal for the internal oscillator circuitry. |
| EXTAL | N1 | This line is one of the connections to an external crystal for the internal oscillator circuitry. |
| XFC | T2 | **External Filter Capacitance**—This input line is the connection pin for an external capacitor filter for the PLL circuitry. |
| CLKOUT | W3 | **Clock Out**—This output line is the clock system frequency. |
| EXTCLK | N2 | **External Clock**—This input line is the external input clock from an external source. |
| TEXP | N3 | **Timer Expired**—This output line reflects the status of the TEXPS bit in the PLPRCR in the CLOCK interface. |
| ALE_A | K2 | **Address Latch Enable A**—This output line is asserted when MPC821 initiates an access to a region under the control of the PCMCIA interface to socket A. |
| CE1_A | B3 | **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 | A3 | **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 | R3 | **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 | R4 | **Wait Slot B**—This input signal, if asserted low, causes a delay in the completion of a transaction on the PCMCIA controlled Slot B. |
| IP_A(0:1) | T5, T4 | **Input Port A 0-1**—These input signals are monitored by the MPC821 and are reflected in the PIPR and PSCR of the PCMCIA interface. |
| IP_A2<br>IOIS16_A | U3 | **Input Port A 2**—This input signal is monitored by the MPC821 and it's value and changes are reported in the PIPR and PSCR of the PCMCIA interface.<br>**I/O Device A is 16 Bits Ports Size**—This input signal is monitored by the MPC821 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(3:7) | W2, U4, U5, T6, T3 | **Input Port A 3-7**—These input signals are monitored by the MPC821 and their values and changes are reported in the PIPR and PSCR of the PCMCIA interface. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| ALE_B<br>DSCK/AT1 | J1 | **Address Latch Enable B**—This output line is asserted when the MPC821 initiates an access to a region under the control of the PCMCIA socket B interface.<br>**Development Serial Clock**—This input line is the clock for the debug port interface.<br>**Address Type 1**—This bidirectional three-state line is driven by the MPC821 when it initiates a transaction on the external bus. When the transaction is initiated by the internal core, it indicates if the transfer is for problem or privilege state. |
| IP_B(0:1)<br>IWP(0:1)<br>VFLS(0:1) | H2, J3 | **Input Port B 0-1**—These input signals are sensed by the MPC821 and their values and changes reported in the PIPR and PSCR of the PCMCIA interface.<br>**Instruction Watchpoint 0-1**—These output lines report the detection of an instruction watchpoint in the program flow executed by the internal core.<br>**Visible History Buffer Flushes Status**—These output lines are output by the MPC821 when a program instructions flow tracking is required by the user. They report the number of instructions flushed from the history buffer in the internal core. |
| IP_B2<br>IOIS16_B<br>AT2 | J2 | **Input Port B 2**—This input signal is sensed by the MPC821 and it's value and changes are reported in the PIPR and PSCR of the PCMCIA interface.<br>**I/O Device B is 16 Bits Port Size**—This input signal is monitored by the MPC821 when a PCMCIA interface transaction is initiated to an I/O region in socket B within the PCMCIA space.<br>**Address Type 2**—This bidirectional three-state line is driven by the MPC821 when it initiates a transaction on the external bus. When the transaction is initiated by the internal core, it indicates if the transfer is instruction or data. |
| IP_B3<br>IWP2<br>VF2 | G1 | **Input Port B 3**—This input signal is monitored by the MPC821 and it's value and changes are reported in the PIPR and PSCR of the PCMCIA interface.<br>**Instruction Watchpoint 2**—This output line reports the detection of an instruction watchpoint in the program flow executed by the internal core.<br>**Visible Instruction Queue Flush Status**—This output line together with VF0 and VF1 is output by the MPC821 when program instruction flow tracking is required by the user. VFx reports the number of instructions flushed from the instruction queue in the internal core. |
| IP_B4<br>LWP0<br>VF0 | G2 | **Input Port B 4**—This input signal is monitored by the MPC821 and it's value and changes are reported in the PIPR and PSCR of the PCMCIA interface.<br>**Load/Store Watchpoint 0**—This output line reports the detection of a data watchpoint in the program flow executed by the internal core.<br>**Visible Instruction Queue Flushes Status**—This output line combined with VF1 and VF2 is output by the MPC821 when a program instructions flow tracking is required by the user. VF reports the number of instructions flushed from the instruction queue in the internal core. |
| IP_B5<br>LWP1<br>VF1 | J4 | **Input Port B 5**—This input signal is monitored by the MPC821 and it's value and changes are reported in the PIPR and PSCR of the PCMCIA interface.<br>**Load/Store Watchpoint 1**—This output line reports the detection of a data watchpoint in the program flow executed by the internal core.<br>**Visible Instruction Queue Flushes Status**—This output line combined with VF0 and VF2 is output by the MPC821 when a program instructions flow tracking is required by the user. VF reports the number of instructions flushed from the instruction queue in the internal core. |
| IP_B6<br>DSDI<br>AT0 | K3 | **Input Port B 6**—This input signal is sensed by the MPC821 and it's value and changes are reported in the PIPR and PSCR of the PCMCIA interface. See **Section 12 PCMCIA Interface** for details.<br>**Development Serial Data Input**—This input line is the data in for the debug port interface. See **Section 5 Development Support** for details.<br>**Address Type 0**—This bidirectional three-state line is driven by the MPC821 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. See **Section 10 MPC821 Bus Interface** for details. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| IP_B7<br>PTR<br>AT3 | H1 | **Input Port B 7**—This input signal is monitored by the MPC821 and it's value and changes are reported in the PIPR and PSCR of the PCMCIA interface.<br>**Program Trace**—This output line is asserted by the MPC821 to indicate an instruction fetch is taking place in order to allow program flow tracking.<br>**Address Type 3**—This bidirectional three-state line is driven by the MPC821 when it initiates a transaction on the external bus. When the transaction is initiated by the internal core, it indicates if the transfer is reservation for data transfers or a program trace indication for instructions fetch. |
| OP(0:1) | L4, L2 | **Output Port 0-1**—These output signals are generated by the MPC821 as a result of a write to the PGCRA register in the PCMCIA interface. |
| OP2<br>MODCK1<br>$\overline{\text{STS}}$ | L1 | **Output Port 2**—This output signal is generated by the MPC821 as a result of a write to the PGCRB register in the PCMCIA interface.<br>**Mode Clock 1**—This input signal is sampled at the negation $\overline{\text{PORESET}}$ to configure the PLL/clock mode of operation.<br>**Special Transfer Start**—This output signal is driven by the MPC821 to indicate the start of a transaction on the external bus or signal the beginning of an internal transaction in show cycle mode. |
| OP3<br>MODCK2<br>DSDO | M4 | **Output Port 3**—This output signal is generated by the MPC821 as a result of a write to the PGCRB register in the PCMCIA interface.<br>**Mode Clock 2**—This input signal is sampled at the $\overline{\text{PORESET}}$ negation to configure the PLL/clock mode of operation.<br>**Development Serial Data Output**—This output line is the data out of the debug port interface. |
| BADDR30<br>$\overline{\text{REG}}$ | **M3** | **Burst Address 30**—This output line duplicates the value of A30 when:<br>• An internal master in the MPC821 initiates a transaction on the external bus.<br>• An asynchronous external master initiates a transaction.<br>• A synchronous external master initiates a single beat transaction.<br>This line is used by the memory controller to allow increments in the address lines that connect to memory devices when a synchronous external master or an internal master initiates a burst transfer.<br>**Register**—When the access is initiated by an internal master to a slave under control of the PCMCIA interface, this signal duplicates the value of the pin TSIZE0/$\overline{\text{REG}}$. When the access is initiated by an external master, this line is output by the PCMCIA interface (if it has to handle the transfer) to indicate which space in the PCMCIA card is currently accessed. |
| BADDR(28:29) | M2<br>K4 | **Burst Address**—These output lines duplicate the value of A(28:29) when:<br>• An internal master in the MPC821 initiates a transaction on the external bus.<br>• An asynchronous external master initiates a transaction.<br>• A synchronous external master initiates a single beat transaction.<br>This line is used by the memory controller to allow increments in the address lines that connect to memory devices when a synchronous external or internal master initiates a burst transfer. |
| $\overline{\text{AS}}$ | L3 | **Address Strobe**—This input pin is driven by an external asynchronous master to indicate a valid address on the A(0:31) lines. The memory controller in the MPC821 synchronizes this signal and controls the memory device addressed under it's control. |
| PA[15]<br>RXD1 | C18 | **General-Purpose I/O Port A Bit 15**—This is bit 15 of the general-purpose I/O port A.<br>**RXD1**—This is the receive data input signal for SCC1. |
| PA[14]<br>TXD1 | D17 | **General-Purpose I/O Port A Bit 14**—This is bit 14 of the general-purpose I/O port A.<br>**TXD1**—This is the transmit data output signal for SCC1. TXD1 has an open-drain capability. |
| PA[13]<br>RXD2 | E17 | **General-Purpose I/O Port A Bit 13**—This is bit 13 of the general-purpose I/O port A.<br>**RXD2**—This is the receive data input signal for SCC2. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| PA[12]<br>TXD2 | F17 | **General-Purpose I/O Port A Bit 12**—This is bit 12 of the general-purpose I/O port A.<br>**TXD2**—This is the transmit data output signal for SCC2. TXD2 has an open-drain capability. |
| PA[11]<br>L1TXDB | G16 | **General-Purpose I/O Port A Bit 11**—This is bit 11 of the general-purpose I/O port A.<br>**L1TXDB**—This is the transmit data output signal for the serial interface TDM port B. L1TXDB has an open-drain capability. |
| PA[10]<br>L1RXDB | J17 | **General-Purpose I/O Port A Bit 10**—This is bit 10 of the general-purpose I/O port A.<br>**L1RXDB**—This is the receive data input signal for the serial interface TDM port B. |
| PA[9]<br>L1TXDA | K18 | **General-Purpose I/O Port A Bit 11**—This is bit 9 of the general-purpose I/O port A.<br>**L1TXDA**—This is the transmit data output signal for the serial interface TDM port A. L1TXDA has an open-drain capability. |
| PA[8]<br>L1RXDA | L17 | **General-Purpose I/O Port A Bit 8**—This is bit 8 of the general-purpose I/O port A.<br>**L1RXDA**—This is the receive data input signal for the serial interface TDM port A. |
| PA[7]<br>CLK1<br>TIN1<br>L1RCLKA<br>BRGO1 | M‘19 | **General-Purpose I/O Port A Bit 7**—This is bit 7 of the general-purpose I/O port A.<br>**CLK1**—This input signal is one of the eight clock pins that can be used to clock the SCCs and the SMCs.<br>**TIN1**—This is the timer 1 external clock pin.<br>**L1RCLKA**—This is the receive clock for the serial interface TDM port A.<br>**BRGO1**—This is the output clock of BRG1. |
| PA[6]<br>CLK2<br>$\overline{\text{TOUT1}}$<br>BRGCLK1 | M17 | **General-Purpose I/O Port A Bit 6**—This is bit 6 of the general-purpose I/O port A.<br>**CLK2**—This input signal is one of the eight clock pins that can be used to clock the SCCs and the SMCs.<br>**$\overline{\text{TOUT1}}$**—This is the timer 1 output pin.<br>**BRGCLK1**—This is one of the two external clock inputs of the BRGs. |
| PA[5]<br>CLK3<br>TIN2<br>L1TCLKA<br>BRGOUT2 | M18 | **General-Purpose I/O Port A Bit 5**—This is bit 5 of the general-purpose I/O port A.<br>**CLK3**—This input signal is one of the eight clock pins that can be used to clock the SCCs and the SMCs.<br>**TIN2**—This is the timer 2 external clock input pin.<br>**L1TCLKA**—This is the transmit clock for the serial interface TDM port A.<br>**BRGOUT2**—This is the output clock of BRG2. |
| PA[4]<br>CLK4<br>$\overline{\text{TOUT2}}$ | P19 | **General-Purpose I/O Port A Bit 4—**This is bit 4 of the general-purpose I/O port A.<br>**CLK4**—This input signal is one of the eight clock pins that can be used to clock the SCCs and the SMCs.<br>**$\overline{\text{TOUT2}}$**—This is the timer 2 output pin. |
| PA[3]<br>CLK5<br>TIN3<br>BRGOUT3 | P17 | **General-Purpose I/O Port A Bit 3**—This is bit 3 of the general-purpose I/O port A.<br>**CLK5**—This input signal is one of the eight clock pins that can be used to clock the SCCs and the SMCs.<br>**TIN3**—This is the timer 3 external clock input pin.<br>**BRGOUT3**—This is the output clock of BRG3. |
| PA[2]<br>CLK6<br>$\overline{\text{TOUT3}}$<br>L1RCLKB<br>BRGCLK2 | R18 | **General-Purpose I/O Port A Bit 2**—This is bit 2 of the general-purpose I/O port A.<br>**CLK6**—This input signal is one of the eight clock pins that can be used to clock the SCCs and SMCs.<br>**$\overline{\text{TOUT3}}$**—This is the timer 3 output pin.<br>**L1RCLKB**—This is the receive clock for the serial interface TDM port B.<br>**BRGCLK2**—This is the one of the two external clock inputs of the BRGs. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| PA[1]<br>CLK7<br>TIN4<br>BRGO4 | T19 | **General-Purpose I/O Port A Bit 1**—This is bit 1 of the general-purpose I/O port A.<br>**CLK7**—This input signal is one of the eight clock pins that can be used to clock the SCCs and the SMCs.<br>**TIN4**—This is the timer 4 external clock input pin.<br>**BRGO4—**This is BRG4 output clock. |
| PA[0]<br>CLK8<br>$\overline{\text{TOUT4}}$<br>L1TCLKB | U19 | **General-Purpose I/O Port A Bit 0**—This is bit 0 of the general-purpose I/O port A.<br>**CLK8**—This input signal is one of the eight clock pins that can be used to clock the SCCs and the SMCs.<br>**$\overline{\text{TOUT4}}$—**This is the timer 4 output pin.<br>**L1TCLKB**—**T**his is the transmit clock for the serial interface TDM port B. |
| PB[31]<br>$\overline{\text{SPISEL}}$<br>$\overline{\text{REJECT1}}$ | C17 | **General-Purpose I/O Port B Bit 31**—This is bit 31 of the general-purpose I/O port B.<br>**$\overline{\text{SPISEL}}$**—This is the SPI slave select input pin.<br>**$\overline{\text{REJECT1}}$**—This is SCC1 CAM interface reject pin. |
| PB[30]<br>SPICLK | C19 | **General-Purpose I/O Port B Bit 30**—This is bit 30 of the general-purpose I/O port B.<br>**SPICLK**—This is the SPI output clock when it is configured as a master or SPI input clock when it is configured as a slave. |
| PB[29]<br>SPIMOSI | E16 | **General-Purpose I/O Port B Bit 29**—This is bit 29 of the general-purpose I/O port B.<br>**SPIMOSI**—This is the SPI output data when it is configured as a master or SPI input data when it is configured as a slave. |
| PB[28]<br>SPIMISO<br>BRGO4 | D19 | **General-Purpose I/O Port B Bit 28**—This is bit 29 of the general-purpose I/O port B.<br>**SPIMISO**—This is the SPI input data when it is configured as a master or SPI output data when it is configured as a slave.<br>**BRGO4**—This is BRG4 output clock. |
| PB[27]<br>I2CSDA<br>BRGO1 | E19 | **General-Purpose I/O Port B Bit 27**—This is bit 27 of the general-purpose I/O port B.<br>**I2CSDA**—This is the $I^2C$ serial data pin. I2CSDA is bidirectional and should be configured as an open-drain output.<br>**BRGO1**—This is BRG1 output clock. |
| PB[26]<br>I2CSCL<br>BRGO2 | F19 | **General-Purpose I/O Port B Bit 26**—This is bit 26 of the general-purpose I/O port B.<br>**I2CSCL**—This is the $I^2C$ serial clock pin. I2CSCL is bidirectional and should be configured as an open-drain output.<br>**BRGO2**—This is BRG2 output clock. |
| PB[25]<br>SMTXD1 | J16 | **General-Purpose I/O Port B Bit 25**—This is bit 25 of the general-purpose I/O port B.<br>**SMTXD1**—This is the SMC1 transmit data output pin. |
| PB[24]<br>SMRXD1 | J18 | **General-Purpose I/O Port B Bit 24**—This is bit 24 of the general-purpose I/O port B.<br>**SMRXD1**—This is the SMC1 receive data input pin. |
| PB[23]<br>$\overline{\text{SMSYN1}}$<br>SDACK1 | K17 | **General-Purpose I/O Port B Bit 23**—This is bit 23 of the general-purpose I/O port B.<br>**$\overline{\text{SMSYN1}}$**—This is the SMC1 external sync input pin.<br>SDACK1—This is the SDMA acknowledge 1 output pin that is used as a peripheral interface signal for IDMA emulation, or as a CAM interface signal for ethernet. |
| PB[22]<br>$\overline{\text{SMSYN2}}$<br>SDACK2 | L19 | **General-Purpose I/O Port B Bit 22—**This is bit 22 of the general-purpose I/O port B.<br>**$\overline{\text{SMSYN2}}$**—This is the SMC2 external sync input pin.<br>SDACK2—This is the SDMA acknowledge 2 output pin that is used as a peripheral interface signal for IDMA emulation, or as a CAM interface signal for ethernet. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| PB[21]<br>SMTXD2<br>L1CLKOB | K16 | **General-Purpose I/O Port B Bit 21**—This is bit 21 of the general-purpose I/O port B.<br>**SMTXD2**—This is the SMC2 transmit data output pin.<br>**L1CLKOB**—This is clock output from the serial interface TDM port B. |
| PB[20]<br>SMRXD2<br>L1CLKOA | L16 | **General-Purpose I/O Port B Bit 20**—This is bit 20 of the general-purpose I/O port B.<br>**SMRXD2**—This is the SMC2 receive data input pin.<br>**L1CLKOA**—This is clock output from the serial interface TDM port A. |
| PB[19]<br>RTS1<br>L1ST1 | N19 | **General-Purpose I/O Port B Bit 19**—This is bit 19 of the general-purpose I/O port B.<br>**RTS1**—This is the request to send modem line for SCC1.<br>**L1ST1**—This one of four output strobes that can be generated by the serial interface. |
| PB[18]<br>RTS2<br>L1ST2 | N17 | **General-Purpose I/O Port B Bit 18**—This is bit 18 of the general-purpose I/O port B.<br>**RTS2**—This is the request to send modem line for SCC2.<br>**L1ST2**—This one of four output strobes that can be generated by the serial interface. |
| PB[17]<br>L1RQB<br>L1ST3 | P18 | **General-Purpose I/O Port B Bit 17**—This is bit 17 of the general-purpose I/O port B.<br>**L1RQB**—This is the D-channel request signal for the serial interface TDM port B.<br>**L1ST3**—This one of four output strobes that can be generated by the serial interface. |
| PB[16]<br>L1RQA<br>L1ST4 | N16 | **General-Purpose I/O Port B Bit 16**—This is bit 16 of the general-purpose I/O port B.<br>**L1RQA**—This is the D-channel request signal for the serial interface TDM port A.<br>**L1ST4**—This one of four output strobes that can be generated by the serial interface. |
| PB[15]<br>BRGO3 | R17 | **General-Purpose I/O Port B Bit 15**—This is bit 15 of the general-purpose I/O port B.<br>**BRGO3**—This is BRG3 output clock. |
| PB[14]<br>RSTRT1 | U18 | **General-Purpose I/O Port B Bit 14**—This is bit 14 of the general-purpose I/O port B.<br>**RSTRT1**—This is the SCC1 serial CAM interface output signals that marks the start of a frame. |
| PC[15]<br>DREQ1<br>RTS1<br>L1ST1 | D16 | **General-Purpose I/O Port C Bit 15**—This is bit 15 of the general-purpose I/O port C.<br>DREQ1—This is the IDMA channel 1 request input signal.<br>**RTS1**—This is the request to send modem line for SCC1.<br>**L1ST1**—This one of four output strobes that can be generated by the serial interface. |
| PC[14]<br>DREQ2<br>RTS2<br>L1ST2 | D18 | **General-Purpose I/O Port C Bit 14**—This is bit 14 of the general-purpose I/O port C.<br>DREQ2—This is the IDMA channel 2 request input signal.<br>**RTS2**—This is the request to send modem line for SCC2.<br>**L1ST2**—This one of four output strobes that can be generated by the serial interface. |
| PC[13]<br>L1RQB<br>L1ST3 | E18 | **General-Purpose I/O Port C Bit 13**—This is bit 13 of the general-purpose I/O port C.<br>**L1RQB**—This is the D-channel request signal for the serial interface TDM port B.<br>**L1ST3**—This one of four output strobes that can be generated by the serial interface. |
| PC[12]<br>L1RQA<br>L1ST4 | F18 | **General-Purpose I/O Port C Bit 12**—This is bit 12 of the general-purpose I/O port C.<br>**L1RQA**—This is the D-channel request signal for the serial interface TDM port A.<br>**L1ST4**—This one of four output strobes that can be generated by the serial interface. |
| PC[11]<br>CTS1 | J19 | **General-Purpose I/O Port C Bit 11**—This is bit 11 of the general-purpose I/O port C.<br>**CTS1**—This is the clear to send modem line for SCC1. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| PC[10]<br>CD1<br>TGATE1 | K19 | **General-Purpose I/O Port C Bit 10**—This is bit 10 of the general-purpose I/O port C.<br>CD1—This is the carrier detect modem line for SCC1.<br>**TGATE1**—This is the timer 1/timer 2 gate signal. |
| PC[9] | L18 | **General-Purpose I/O Port C Bit 9**—This is bit 9 of the general-purpose I/O port C. |
| PC[8]<br>TGATE2 | M18 | **General-Purpose I/O Port C Bit 8**—This is bit 8 of the general-purpose I/O port C.<br>**TGATE2**—This is the timer 3/timer 4 gate signal. |
| PC[7]<br>L1TSYNCB<br>SDACK2 | M16 | **General-Purpose I/O Port C Bit 7**—This is bit 7 of the general-purpose I/O port C.<br>**L1TSYNCB**—This is the transmit sync input for the serial interface TDM port B.<br>SDACK2—This is the SDMA acknowledge 2 output pin that is used as a peripheral interface signal for IDMA emulation or as a CAM interface signal for ethernet. |
| PC[6]<br>L1RSYNCB | R19 | **General-Purpose I/O Port C Bit 6**—This is bit 6 of the general-purpose I/O port C.<br>**L1RSYNCB**—This is the receive sync input for the serial interface TDM port B. |
| PC[5]<br>L1TSYNCA<br>SDACK1 | T18 | **General-Purpose I/O Port C Bit 5**—This is bit 5 of the general-purpose I/O port C.<br>**L1TSYNCA**—This is the transmit sync input for the serial interface TDM port A.<br>SDACK1—This is the SDMA acknowledge 1output pin that is used as a peripheral interface signal for IDMA emulation or as a CAM interface signal for ethernet. |
| PC[4]<br>L1RSYNCA | T17 | **General-Purpose I/O Port C Bit 4**—This is bit 4 of the general-purpose I/O port C.<br>**L1RSYNCA**—This is the receive sync input for the serial interface TDM port A. |
| PD[15]<br>LD8 | U17 | **General-Purpose I/O Port D Bit 15**—This is bit 15 of the general-purpose I/O port D.<br>**LD8**—This is one of the data bus bits used to drive the LCD panel. |
| PD[14]<br>LD7 | V19 | **General-Purpose I/O Port D Bit 14**—This is bit 14 of the general-purpose I/O port D.<br>**LD7**—This is one of the data bus bits used to drive the LCD panel. |
| PD[13]<br>LD6 | V18 | **General-Purpose I/O Port D Bit 13**—This is bit 13 of the general-purpose I/O port D.<br>**LD6**—This is one of the data bus bits used to drive the LCD panel. |
| PD[12]<br>LD5 | R16 | **General-Purpose I/O Port D Bit 12**—This is bit 12 of the general-purpose I/O port D.<br>**LD5**—This is one of the data bus bits used to drive the LCD panel. |
| PD[11]<br>LD4 | T16 | **General-Purpose I/O Port D Bit 11**—This is bit 11 of the general-purpose I/O port D.<br>**LD4**—This is one of the data bus bits used to drive the LCD panel. |
| PD[10]<br>LD3 | W18 | **General-Purpose I/O Port D Bit 10**—This is bit 10 of the general-purpose I/O port D.<br>**LD3**—This is one of the data bus bits used to drive the LCD panel. |
| PD[9]<br>LD2 | V17 | **General-Purpose I/O Port D Bit 9—**This is bit 9 of the general-purpose I/O port D.<br>**LD2**—This is one of the data bus bits used to drive the LCD panel. |
| PD[8]<br>LD1 | W17 | **General-Purpose I/O Port D Bit 8**—This is bit 8 of the general-purpose I/O port D.<br>**LD1**—This is one of the data bus bits used to drive the LCD panel. |
| PD[7]<br>LD0 | T15 | **General-Purpose I/O Port D Bit 7**—This is bit 7 of the general-purpose I/O port D.<br>**LD0**—This is one of the data bus bits used to drive the LCD panel. |

| PIN NAME | PIN NUMBER | DESCRIPTION |
|---|---|---|
| PD[6]<br>LCD_AC<br>LOE | V16 | **General-Purpose I/O Port D Bit 6**—This is bit 6 of the general-purpose I/O port D.<br>**LCD_AC**—This output signal from the LCD controller toggles once every programmable number of frames. It is used with passive panels.<br>**LOE**—This is the output enable signal which is used with TFT panels. |
| PD[5]<br>FRAME/VSYNC | U15 | **General-Purpose I/O Port D Bit 5**—This is bit 5 of the general-purpose I/O port D.<br>**FRAME/VSYNC**—This output from the LCD controller marks the beginning of a new frame. |
| PD[4]<br>LOAD/HSYNC | U16 | **General-Purpose I/O Port D Bit 4**—This is bit 4 of the general-purpose I/O port D.<br>**LOAD/HSYNC**—This output from the LCD controller marks the beginning of a new display line. |
| PD[3]<br>SHIFT/CLK | W16 | **General-purpose I/O Port D Bit 3**—This is bit 3 of the general-purpose I/O port D.<br>**SHIFT/CLK**—This output from the LCD controller generates the shift clock timing to the LCD panel. |
| POWER SUPPLY | * | **VDDL**—This is the power supply of the internal logic.<br>**VDDH**—This is the power supply of the I/O buffers and certain parts of the clock control.<br>**VDDSYN**—This is the power supply of the PLL circuitry.<br>**KAPWR**—This is the power supply of the internal OSCM, RTC, PIT, DEC, and TB. |
| TCK<br>DSCK | H16 | Provides clock to scan chain logic or for the development port logic. Should be tied to Vcc if JTAG or development port are not used. |
| TMS | G18 | This pin controls the scan chain test mode operations. Should be tied to ground if unused. |
| TDI<br>DSDI | H17 | This pin is the input serial data for either the scan chain logic or the development port and determines the operating mode of the development port at reset. This pin should be tied to ground. |
| TDO<br>DSDO | G17 | This pin is the output serial data for either the scan chain logic or for the development port. |
| $\overline{\text{TRST}}$ | G19 | Reset for the scan chain logic. Should be tied to $\overline{\text{HRESET}}$. |

NOTE: * See Figure 2-3.

# SECTION 3
# MEMORY MAP

The MPC821 internal memory resources are mapped within a contiguous block of storage. The size of the internal space in the MPC821 is 16 kbytes. The location of this block within the global 4-gigabyte real storage space can be mapped on 64 kbytes resolution through an implementation specific special register called the internal memory map register (IMMR). Refer to **Section 12.4.1.2 Internal Memory Map Register** for more information. The following table defines the internal memory map of the MPC821.

**Table 3-1. MPC821 Internal Memory Map**

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| **GENERAL SIU** | | | |
| 000 | SIUMCR | SIU Module Configuration Register | 32 |
| 004 | SYPCR | System Protection Control Register | 32 |
| 008 | Reserved | | |
| 00E | SWSR | Software Service Register | 16 |
| 010 | SIPEND | SIU Interrupt Pending Register | 32 |
| 014 | SIMASK | SIU Interrupt Mask Register | 32 |
| 018 | SIEL | SIU Interrupt Edge Level Mask Register | 32 |
| 01C | SIVEC | SIU Interrupt Vector Register | 32 |
| 020 | TESR | Transfer Error Status Register | 32 |
| 024 to 02F | Reserved | | |
| 030 | SDCR | SDMA Configuration Register | 32 |
| 034 to 07F | Reserved | | |
| **PCMCIA** | | | |
| 080 | PBR0 | PCMCIA Interface Base Register 0 | 32 |
| 084 | POR0 | PCMCIA Interface Option Register 0 | 32 |
| 088 | PBR1 | PCMCIA Interface Base Register 1 | 32 |
| 08C | POR1 | PCMCIA Interface Option Register 1 | 32 |
| 090 | PBR2 | PCMCIA Interface Base Register 2 | 32 |

Table 3-1. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 094 | POR2 | PCMCIA Interface Option Register 2 | 32 |
| 098 | PBR3 | PCMCIA Interface Base Register 3 | 32 |
| 09C | POR3 | PCMCIA Interface Option Register 3 | 32 |
| 0A0 | PBR4 | PCMCIA Interface Base Register 4 | 32 |
| 0A4 | POR4 | PCMCIA Interface Option Register 4 | 32 |
| 0A8 | PBR5 | PCMCIA Interface Base Register 5 | 32 |
| 0AC | POR5 | PCMCIA Interface Option Register 5 | 32 |
| 0B0 | PBR6 | PCMCIA Interface Base Register 6 | 32 |
| 0B4 | POR6 | PCMCIA Interface Option Register 6 | 32 |
| 0B8 | PBR7 | PCMCIA Interface Base Register 7 | 32 |
| 0BC | POR7 | PCMCIA Interface Option Register 7 | 32 |
| 0C0 to 0DF | Reserved | | |
| 0E0 | PGCRA | PCMCIA Interface General Control Register A | 32 |
| 0E4 | PGCRB | PCMCIA Interface General Control Register B | 32 |
| 0E8 | PSCR | PCMCIA Interface Status Changed Register | 32 |
| 0EC to 0EF | Reserved | | |
| 0F0 | PIPR | PCMCIA Interface Input Pins Register | 32 |
| 0F4 to 0F7 | Reserved | | |
| 0F8 | PER | PCMCIA Interface Enable Register | 32 |
| 0FC to 0FF | Reserved | | |
| **MEMC** | | | |
| 100 | BR0 | Base Register Bank 0 | 32 |
| 104 | OR0 | Option Register Bank 0 | 32 |
| 108 | BR1 | Base Register Bank 1 | 32 |
| 10c | OR1 | Option Register Bank 1 | 32 |
| 110 | BR2 | Base Register Bank 2 | 32 |
| 114 | OR2 | Option Register Bank 2 | 32 |
| 118 | BR3 | Base Register Bank 3 | 32 |
| 11C | OR3 | Option Register Bank 3 | 32 |
| 120 | BR4 | Base Register Bank 4 | 32 |
| 124 | OR4 | Option Register Bank 4 | 32 |
| 128 | BR5 | Base Register Bank 5 | 32 |

## Table 3-1. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 12C | OR5 | Option Register Bank 5 | 32 |
| 130 | BR6 | Base Register Bank 6 | 32 |
| 134 | OR6 | Option Register Bank 6 | 32 |
| 138 | BR7 | Base Register Bank 7 | 32 |
| 13C | OR7 | Option Register Bank 7 | 32 |
| 140 to 163 | Reserved | | |
| 164 | MAR | Memory Address Register | 32 |
| 168 | MCR | Memory Command Register | 32 |
| 16C to 16F | Reserved | | |
| 170 | MAMR | Machine A Mode Register | 32 |
| 174 | MBMR | Machine B Mode Register | 32 |
| 178 | MSTAT | Memory Status Register | 16 |
| 17A | MPTPR | Memory Periodic Timer Prescaler | 16 |
| 17C | MDR | Memory Data Register | 32 |
| 180 to 1FF | Reserved | | |
| **SYSTEM INTEGRATION TIMERS** | | | |
| 200 | TBSCR | Timebase Status and Control Register | 16 |
| 204 | TBREFF0 | Timebase Reference Register 0 | 32 |
| 208 | TBREFF1 | Timebase Reference Register 1 | 32 |
| 20C to 21F | Reserved | | |
| 220 | RTCSC | Real-Time Clock Status and Control Register | 16 |
| 224 | RTC | Real-Time Clock Register | 32 |
| 228 | RTSEC | Real-Time Alarm Seconds | 32 |
| 22C | RTCAL | Real-Time Alarm Register | 32 |
| 230 to 23F | Reserved | | |
| 240 | PISCR | Periodic Interrupt Status and Control Register | 16 |
| 244 | PITC | Periodic Interrupt Count Register | 32 |
| 248 | PITR | Periodic Interrupt Timer Register | 32 |
| 24C to 27F | Reserved | | |
| **CLOCKS AND RESET** | | | |
| 280 | SCCR | System Clock Control Register | 32 |
| 284 | PLPRCR | PLL, Low Power and Reset Control Register | 32 |

**Table 3-1. MPC821 Internal Memory Map (Continued)**

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 288 | RSR | Reset Status Register | 32 |
| 28C to 2FF | Reserved | | |
| SYSTEM INTEGRATION TIMERS KEYS | | | |
| 300 | TBSCRK | Timebase Status and Control Register Key | 32 |
| 304 | TBREFF0K | Timebase Reference Register 0 Key | 32 |
| 308 | TBREFF1K | Timebase Reference Register 1 Key | 32 |
| 30C | TBK | Timebase and Decrementer Register Key | 32 |
| 310 to 31F | Reserved | | |
| 320 | RTCSCK | Real-Time Clock Status and Control Register Key | 32 |
| 324 | RTCK | Real-Time Clock Register Key | 32 |
| 328 | RTSECK | Real-Time Alarm Seconds Key | 32 |
| 32C | RTCALK | Real-Time Alarm Register Key | 32 |
| 330 to 33F | Reserved | | |
| 340 | PISCRK | Periodic Interrupt Status and Control Register Key | 32 |
| 344 | PITCK | Periodic Interrupt Count Register Key | 32 |
| 348 to 37F | Reserved | | |
| CLOCKS AND RESET KEYS | | | |
| 380 | SCCRK | System Clock Control Key | 32 |
| 384 | PLPRCRK | PLL, Low Power and Reset Control Register Key | 32 |
| 388 | RSRK | Reset Status Register Key | 32 |
| 38C to 3FF | Reserved | | |
| 400 to 7FF | Reserved | | |
| 800 to 83F | Reserved | | |
| LCD | | | |
| 840 | LCCR | LCD Configuration Register | 32 |
| 844 | LCHCR | LCD Horizontal Control Register | 32 |
| 848 | LCVCR | LCD Vertical Control Register | 32 |
| 84C | Reserved | | |
| 850 | LCFAA | LCD Frame Buffer A Start Address | 32 |
| 854 | LCFBA | LCD Frame Buffer B Start Address | 32 |
| 858 | LCSR | LCD Status Register | 8 |

### Table 3-1. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 859 to 85F | Reserved | | |
| **I²C** | | | |
| 860 | I2MOD | I²C Mode Register | 8 |
| 864 | I2ADD | I²C Address Register | 8 |
| 868 | I2BRG | I²C BRG Register | 8 |
| 86C | I2COM | I²C Command Register | 8 |
| 870 | I2CER | I²C Event Register | 8 |
| 874 | I2CMR | I²C Mask Register | 8 |
| **DMA** | | | |
| 900 to 903 | Reserved | | |
| 904 | SDAR | SDMA Address Register | 32 |
| 908 | SDSR | SDMA Status Register | 8 |
| 909 to 90B | Reserved | | |
| 90C | SDMR | SDMA Mask Register | 8 |
| 90D to 90F | Reserved | | |
| 910 | IDSR1 | IDMA1 Status Register | 8 |
| 911 to 913 | Reserved | | |
| 914 | IDMR1 | IDMA1 Mask Register | 8 |
| 915 to 917 | Reserved | | |
| 918 | IDSR2 | IDMA2 Status Register | 8 |
| 919 to 91B | Reserved | | |
| 91C | IDMR2 | IDMA2 Mask Register | 8 |
| 91D to 92F | Reserved | | |
| **CPM INTERRUPT CONTROL** | | | |
| 930 | CIVR | CP Interrupt Vector Register | 16 |
| 932 to 93F | Reserved | | |
| 940 | CICR | CP Interrupt Configuration Register | 32 |
| 944 | CIPR | CP Interrupt Pending Register | 32 |
| 948 | CIMR | CP Interrupt Mask Register | 32 |
| 94C | CISR | CP In-Service Register | 32 |

Table 3-1. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| INPUT/OUTPUT PORT | | | |
| 950 | PADIR | Port A Data Direction Register | 16 |
| 952 | PAPAR | Port A Pin Assignment Register | 16 |
| 954 | PAODR | Port A Open Drain Register | 16 |
| 956 | PADAT | Port A Data Register | 16 |
| 958 to 95F | Reserved | | |
| 960 | PCDIR | Port C Data Direction Register | 16 |
| 962 | PCPAR | Port C Pin Assignment Register | 16 |
| 964 | PCSO | Port C Special Options | 16 |
| 966 | PCDAT | Port C Data Register | 16 |
| 968 | PCINT | Port C Interrupt Control Register | 16 |
| 96A to 96F | Reserved | | |
| 970 | PDDIR | Port D Data Direction Register | 16 |
| 972 | PDPAR | Port D Pin Assignment Register | 16 |
| 974 | Reserved | Reserved | 16 |
| 976 | PDDAT | Port D Data Register | 16 |
| 978 to 97F | Reserved | | |
| CPM TIMERS | | | |
| 980 | TGCR | Timer Global Configuration Register | 16 |
| 982 to 98F | Reserved | | |
| 990 | TMR1 | Timer1 Mode Register | 16 |
| 992 | TMR2 | Timer2 Mode Register | 16 |
| 994 | TRR1 | Timer1 Reference Register | 16 |
| 996 | TRR2 | Timer2 Reference Register | 16 |
| 998 | TCR1 | Timer1 Capture Register | 16 |
| 99A | TCR2 | Timer2 Capture Register | 16 |
| 99C | TCN1 | Timer1 Counter | 16 |
| 99E | TCN2 | Timer2 Counter | 16 |
| 9A0 | TMR3 | Timer3 Mode Register | 16 |
| 9A2 | TMR4 | Timer4 Mode Register | 16 |
| 9A4 | TRR3 | Timer3 Reference Register | 16 |
| 9A6 | TRR4 | Timer4 Reference Register | 16 |

**Table 3-1. MPC821 Internal Memory Map (Continued)**

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 9A8 | TCR3 | Timer3 Capture Register | 16 |
| 9AA | TCR4 | Timer4 Capture Register | 16 |
| 9AC | TCN3 | Timer3 Counter | 16 |
| 9AE | TCN4 | Timer4 Counter | 16 |
| 9B0 | TER1 | Timer1 Event Register | 16 |
| 9B2 | TER2 | Timer2 Event Register | 16 |
| 9B4 | TER3 | Timer3 Event Register | 16 |
| 9B6 | TER4 | Timer4 Event Register | 16 |
| 9B8 to 9BF | Reserved | | |
| **COMMUNICATION PROCESSOR** | | | |
| 9CO | CPCR | Communication Processor Command Register | 16 |
| 9C4 | RCCR | RISC Configuration Register | 16 |
| 9C6 | RES | Reserved | 8 |
| 9C7 | RMDS | RISC Development Support Status Register | 8 |
| 9C8 | RMDR | RISC Microcode Development Support Control Register | 32 |
| 9CC | RCTR1 | RISC Controller Trap Register 1 | 16 |
| 9CE | RCTR2 | RISC Controller Trap Register 2 | 16 |
| 9D0 | RCTR3 | RISC Controller Trap Register 3 | 16 |
| 9D2 | RCTR4 | RISC Controller Trap Register 4 | 16 |
| 9D6 | RTER | RISC Timers Event Register | 16 |
| 9DA | RTMR | RISC Timers Mask Register | 16 |
| 9DC to 9EF | Reserved | | |
| **BRGs** | | | |
| 9F0 | BRGC1 | BRG1 Configuration Register | 32 |
| 9F4 | BRGC2 | BRG2 Configuration Register | 32 |
| 9F8 | BRGC3 | BRG3 Configuration Register | 32 |
| 9FC | BRGC4 | BRG4 Configuration Register | 32 |
| **SCC1** | | | |
| A00 | GSMR_L1 | SCC1 General Mode Register | 32 |
| A04 | GSMR_H1 | SCC1 General Mode Register | 32 |
| A08 | PSMR1 | SCC1 Protocol Specific Mode Register | 16 |

### Table 3-1. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| A0C | TODR1 | SCC1 Transmit-On-Demand Register | 16 |
| A0E | DSR1 | SCC1 Data Synchronization Register | 16 |
| A10 | SCCE1 | SCC1 Event Register | 16 |
| A14 | SCCM1 | SCC1 Mask Register | 16 |
| A17 | SCCS1 | SCC1 Status Register | 8 |
| A18 to A1F | Reserved | | |
| **SCC2** | | | |
| A20 | GSMR_L2 | SCC2 General Mode Register | 32 |
| A24 | GSMR_H2 | SCC2 General Mode Register | 32 |
| A28 | PSMR2 | SCC2 Protocol Specific Mode Register | 16 |
| A2C | TODR2 | SCC2 Transmit-On-Demand Register | 16 |
| A2E | DSR2 | SCC2 Data Synchronization Register | 16 |
| A30 | SCCE2 | SCC2 Event Register | 16 |
| A34 | SCCM2 | SCC2 Mask Register | 16 |
| A37 | SCCS2 | SCC2 Status Register | 8 |
| A38 to A3F | Reserved | | |
| **SMC1** | | | |
| A82 | SMCMR1 | SMC1 Mode Register | 16 |
| A86 | SMCE1 | SMC1 Event Register | 8 |
| A8A | SMCM1 | SMC1 Mask Register | 8 |
| A8C | Reserved | | |
| **SMC2** | | | |
| A92 | SMCMR2 | SMC2 Mode Register | 16 |
| A96 | SMCE2 | SMC2 or PIP Event Register | 8 |
| A9A | SMCM2 | SMC2 Mask Register | 8 |
| A9C | Reserved | | |
| **SPI** | | | |
| AA0 | SPMODE | SPI Mode Register | 16 |
| AA6 | SPIE | SPI Event Register | 8 |
| AAA | SPIM | SPI Mask Register | 8 |
| AAD | SPCOM | SPI Command Register | 8 |

### Table 3-1. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| **PIP** | | | |
| AB2 | PIPC | PIP Configuration Register | 16 |
| AB6 | PTPR | PIP Timing Parameters Register | 16 |
| AB8 | PBDIR | Port B Data Direction Register | 32 |
| ABC | PBPAR | Port B Pin Assignment Register | 32 |
| AC2 | PBODR | Port B Open Drain Register | 16 |
| AC4 | PBDAT | Port B Data Register | 32 |
| AC8 to ADF | Reserved | | |
| **SI** | | | |
| AE0 | SIMODE | SI Mode Register | 32 |
| AE4 | SIGMR | SI Global Mode Register | 8 |
| AE6 | SISTR | SI Status Register | 8 |
| AE7 | SICMR | SI Command Register | 8 |
| AE8 | RES | Reserved | 32 |
| AEC | SICR | SI Clock Route | 32 |
| AF0 | SIRP | SI RAM Pointers | 32 |
| AF4 to BFF | Reserved | | |
| C00 to DFF | SIRAM | SI Routing RAM | 512 bytes |
| E00 to FFF | LCOLR | LCD Color RAM | 512 bytes |
| 1000 to 1FFF | Reserved | | |
| 2000 to 4000 | DPRAM | | |

# SECTION 4
# RESET

## 4.1 RESET OPERATION

The MPC821 has several inputs to the reset logic:

- Power-on reset (POR)
- External hard reset ($\overline{\text{HRESET}}$)
- System reset pin ($\overline{\text{SRESET}}$)
- Loss of lock
- Software watchdog reset
- Checkstop reset
- Debug port hard reset
- Debug port soft reset
- JTAG 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 (RSR) reflects the last source to cause a reset.

### 4.1.1 Reset Causes

**4.1.1.1 POWER-ON RESET.** The power-on reset is an active low input pin. In a system with power-down low power mode, this pin should only be activated as a result of a voltage fail in the KAPWR rail. When this pin is asserted, the MODCK[1:2] bits are sampled and the PLL multiplication factor and pitrtclk and tmbclk sources are changed to their default values. When this pin is negated, internal MODCK[1:2] values are unchanged. The $\overline{\text{PORESET}}$ pin should be asserted for a minimum of 3 microseconds. After detecting this assertion, the MPC821 enters the power-on reset state and remains in this state until the last of the following two events occur:

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

During the power-on reset state, the pins $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ are asserted by the MPC821.

$\overline{\text{HRESET}}$—Hard Reset

This is a bidirectional I/O pin, active low. The MPC821 can only detect an external assertion of $\overline{\text{HRESET}}$ if it occurs while the MPC821 is not asserting reset. During $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$ is asserted. The $\overline{\text{HRESET}}$ is an open collector type pin.

$\overline{\text{SRESET}}$—Soft Reset

This is a bidirectional I/O pin, active low. The MPC821 can only detect an external assertion of $\overline{\text{SRESET}}$ if it occurs while the MPC821 is not asserting reset. The $\overline{\text{SRESET}}$ is an open collector type pin.

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

**4.1.1.3 SOFTWARE WATCHDOG RESET.** After the CPU-chip watchdog counts to zero, a software watchdog reset is asserted. The enabled software watchdog event then generates an internal hard reset sequence.

**4.1.1.4 CHECKSTOP RESET.** If the CPU-core enters a checkstop state and the checkstop reset is enabled (the CSR bit in the PLPRCR is set), the checkstop reset is asserted. The enabled checkstop event then generates an Internal hard reset sequence.

**4.1.1.5 DEBUG PORT HARD RESET.** When the development port receives a hard reset request from the development tool, an internal hard reset sequence is generated. In this case the development tool must reconfigure the debug port. Refer to **Section 19.3.2.5.1 Serial Data Into Development Port** for more information.

**4.1.1.6 DEBUG PORT SOFT RESET.** When the development port receives a soft reset request from the development tool, an internal soft reset sequence is generated. In this case the development tool must reconfigure the debug port. Refer to **Section 19.3.2.5.1 Serial Data Into Development Port** for more information.

**4.1.1.7 JTAG RESET.** When the JTAG logic asserts the JTAG soft reset signal, an internal soft reset sequence will be generated. This reset is only needed for production test.

### 4.1.2 Reset Actions

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

**Table 4-1. Reset Action Taken For Each Reset Cause**

| RESET EFFECT | RESET LOGIC AND PLL STATES RESET | SYSTEM CONFIG RESET | CLOCK MODULE RESET | $\overline{\text{HRESET}}$ PIN DRIVEN | DEBUG PORT CONFIG | OTHER INTERNAL LOGIC RESET | $\overline{\text{SRESET}}$ PIN DRIVEN |
|---|---|---|---|---|---|---|---|
| RESET SOURCE | | | | | | | |
| Power-On Reset | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| External Hard Reset<br>Loss of Lock<br>Software Watchdog<br>Check Stop<br>Debug Port Hard Reset<br>JTAG Reset | No | Yes | Yes | Yes | Yes | Yes | Yes |
| External Soft Reset<br>Debug Port Soft Reset | No | No | No | No | Yes | Yes | Yes |

### 4.1.3 Power-On Reset Flow

When $\overline{\text{PORESET}}$ is asserted (driven low), the MPC821 enters the power-on reset (POR) state and during this state, $\overline{\text{SRESET}}$ and $\overline{\text{HRESET}}$ are asserted (driven low) by the chip. When the MPC821 remains in POR, the extension counter of 512 is reset and the MODCK[1:2] pins are sampled at the negation of the POR pin. After the negation of $\overline{\text{PORESET}}$ or the PLL LOCK (the last of these two), the chip enters the state of internal initiated $\overline{\text{HRESET}}$ and continues driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ for 512 cycles.

When the timer expires, which is usually after 512 cycles, the configuration is sampled from data pins (if required) and the chip stops driving the pins. An external pull-up resistor should drive the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins high (negate). After the negation of the pins is detected, a 16-cycle period is taken before testing the presence of an external (hard/soft) reset. Refer to **Section 4.3.1 Hard Reset Configuration** for more information.

### 4.1.4 External $\overline{\text{HRESET}}$ Flow

When an external $\overline{\text{HRESET}}$ assertion (driven low) is detected, the chip begins driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ for 512 cycles. When the timer expires (after 512 cycles) the configuration is sampled from data pins and the chip stops driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins. An external pull-up resistor should drive the pins high (negate) and after negation is detected, a 16-cycle period is taken before testing the presence of an external (hard/soft) reset. Refer to **Section 4.3.1 Hard Reset Configuration** for more information.

### 4.1.5 Internal $\overline{\text{HRESET}}$ Flow

When the chip detects a cause to assert $\overline{\text{HRESET}}$, it begins driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins for 512 cycles. When the timer expires (after 512 cycles) the configuration is sampled from data pins and the chip stops driving the pins. An external pull-up resistor should drive the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins high (negate) and after negation of the pins is detected, a 16-cycle period is taken before testing the presence of an external (hard/soft) reset. Refer to **Section 4.3.1 Hard Reset Configuration** for more information.

### 4.1.6 External $\overline{\text{SRESET}}$ Flow

When an external $\overline{\text{SRESET}}$ assertion (driven low) is detected, the chip begins driving the $\overline{\text{SRESET}}$. When the timer expires (after 512 cycles) the debug port configuration is sampled from the DSDI and DSCK pins and the chip stops driving the pin. An external pull-up resistor should drive it high (negate) and after negation is detected, a 16-cycle period is taken before testing the presence of an external soft reset.

### 4.1.7 Internal $\overline{\text{SRESET}}$ Flow

When the chip detects a cause to assert $\overline{\text{SRESET}}$, it begins driving the $\overline{\text{SRESET}}$. When the timer expires (after 512 cycles) the debug port configuration is sampled from the DSDI and DSCK pins and the chip stops driving the $\overline{\text{SRESET}}$ pin. An external pull-up resistor should drive the pin high (negate) and after negation is detected, a 16-cycle period is taken before testing the presence of an external soft reset.

## 4.2 RESET STATUS REGISTER

The reset status register (RSR) is a 16-bit register powered by the keep alive power supply. The RSR is memory-mapped into the MPC821, SIU register map and it receives it's default reset values at power-on reset. See Table 4-2 for an overview of this register.

**Table 4-2. Reset Status Register**

| BITS | MNEMONIC | POWER-ON RESET DEFAULT VALUE | DESCRIPTION | FUNCTION |
|---|---|---|---|---|
| 0 | EHRS | 0 | External Hard Reset Status | 0 = No external hard reset event occurs<br>1 = An external hard reset event occurs |
| 1 | ESRS | 0 | External Soft Reset Status | 0 = No external soft reset event occurs<br>1 = An external soft reset event occurs |
| 2 | LLRS | 0 | Loss of Lock Status | 0 = No enabled loss of lock reset event occurs<br>1 = An enabled loss of lock reset event occurs |
| 3 | SWRS | 0 | Software Watchdog Reset Status | 0 = No software watchdog reset event occurs<br>1 = A software watchdog reset event occurs |
| 4 | CSRS | 0 | Check Stop Reset Status | 0 = No enabled checkstop reset event occurs<br>1 = An enabled checkstop reset event occurs |

Table 4-2. Reset Status Register (Continued)

| BITS | MNEMONIC | POWER-ON RESET DEFAULT VALUE | DESCRIPTION | FUNCTION |
|---|---|---|---|---|
| 5 | DBHRS | 0 | Debug Port Hard Reset Status | 0 = No debug port hard reset request occurs<br>1 = A debug port hard reset request occurs |
| 6 | DBSRS | 0 | Debug Port Soft Reset Status | 0 = No debug port soft reset request occurs<br>1 = A debug port soft reset request occurs |
| 7 | JTRS | 0 | JTAG Reset Status | 0 = No JTAG reset event occurs<br>1 = A JTAG reset event occurs |

EHRS—External Hard Reset Status Bit

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

ESRS—External Soft Reset Status Bit

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

LLRS—Loss of Lock Reset Status Bit

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

SWRS—Software Watchdog Reset Status Bit

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

CSRS—Check Stop Reset Status Bit

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

DBHRS—Debug Port Hard Reset Status Bit

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

DBSRS—Debug Port Soft Reset Status Bit

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

JTRS—JTAG Reset Status Bit

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

## 4.3 RESET CONFIGURATION

### 4.3.1 Hard Reset Configuration

When a hard reset event occurs ($\overline{\text{HRESET}}$ asserted), the MPC821 reconfigures it's hardware system as well as the development port configuration. The logical value of the bits that determine it's initial mode of operation, are sampled either from the data bus or from an internal default constant (D(0:31)=x'00000000). If at the sampling time $\overline{\text{RSTCONF}}$ is asserted, the configuration is sampled from the data bus, otherwise, it is sampled from the internal default. While $\overline{\text{HRESET}}$ and $\overline{\text{RSTCONF}}$ are asserted, the MPC821 pulls the data bus LOW through a weak resistor. The user can overwrite this default by driving HIGH to the appropriate bit (see Figure 4-1 below). The hardware reset configuration scheme when the $\overline{\text{PORESET}}$ line is asserted, is shown in Figure 4-2 through Figure 4-4. During the assertion of the $\overline{\text{PORESET}}$ input signal, the chip assumes the default reset configuration. This assumed configuration changes (if the input signal $\overline{\text{RSTCONF}}$ is asserted) when the $\overline{\text{PORESET}}$ is negated or the CLKOUT starts oscillating. In this last case, the hardware configuration is sampled every 9 clock cycles on the rising edge of the CLKOUT. The setup time required for the data bus is 15 cycles and the maximum rise time of $\overline{\text{HRESET}}$ should be less than 6 clock cycles. Refer to **Section 4.3.2 Soft Reset Configuration** for more information.

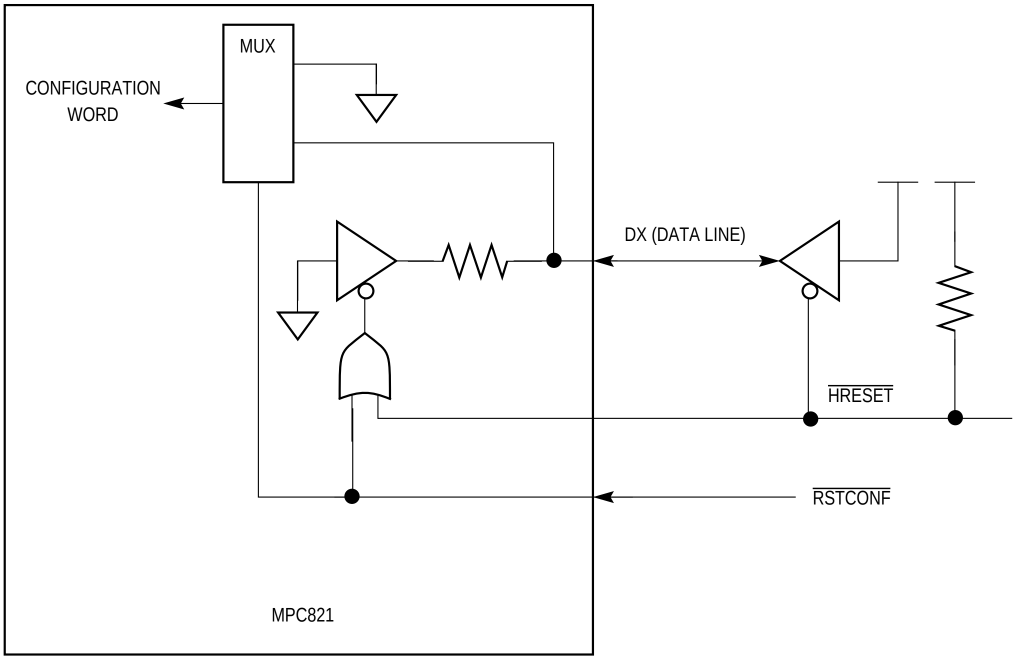


Figure 4-1. Reset Configuration Basic Scheme

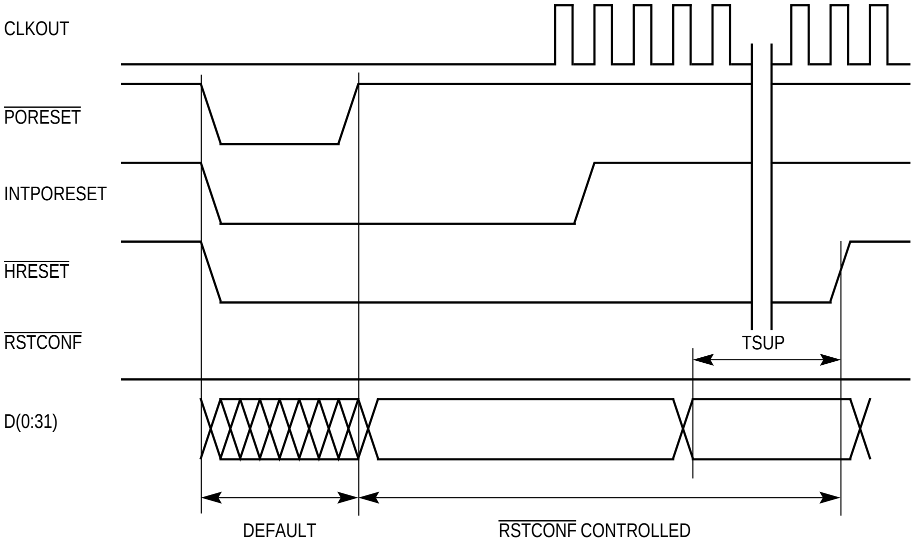


Figure 4-2. Reset Configuration Sampling Scheme
For Short PORESET Assertion

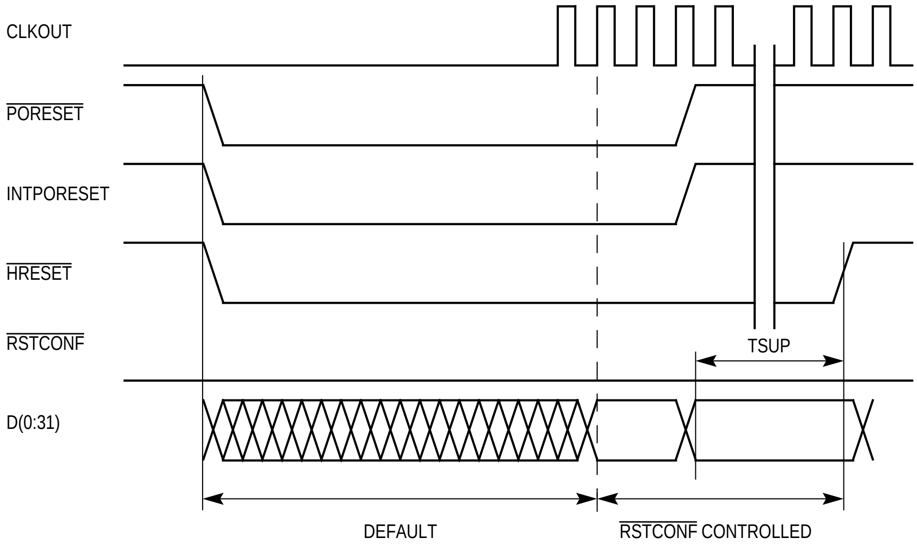


**Figure 4-3. Reset Configuration Sampling Scheme For Long PORESET Assertion**

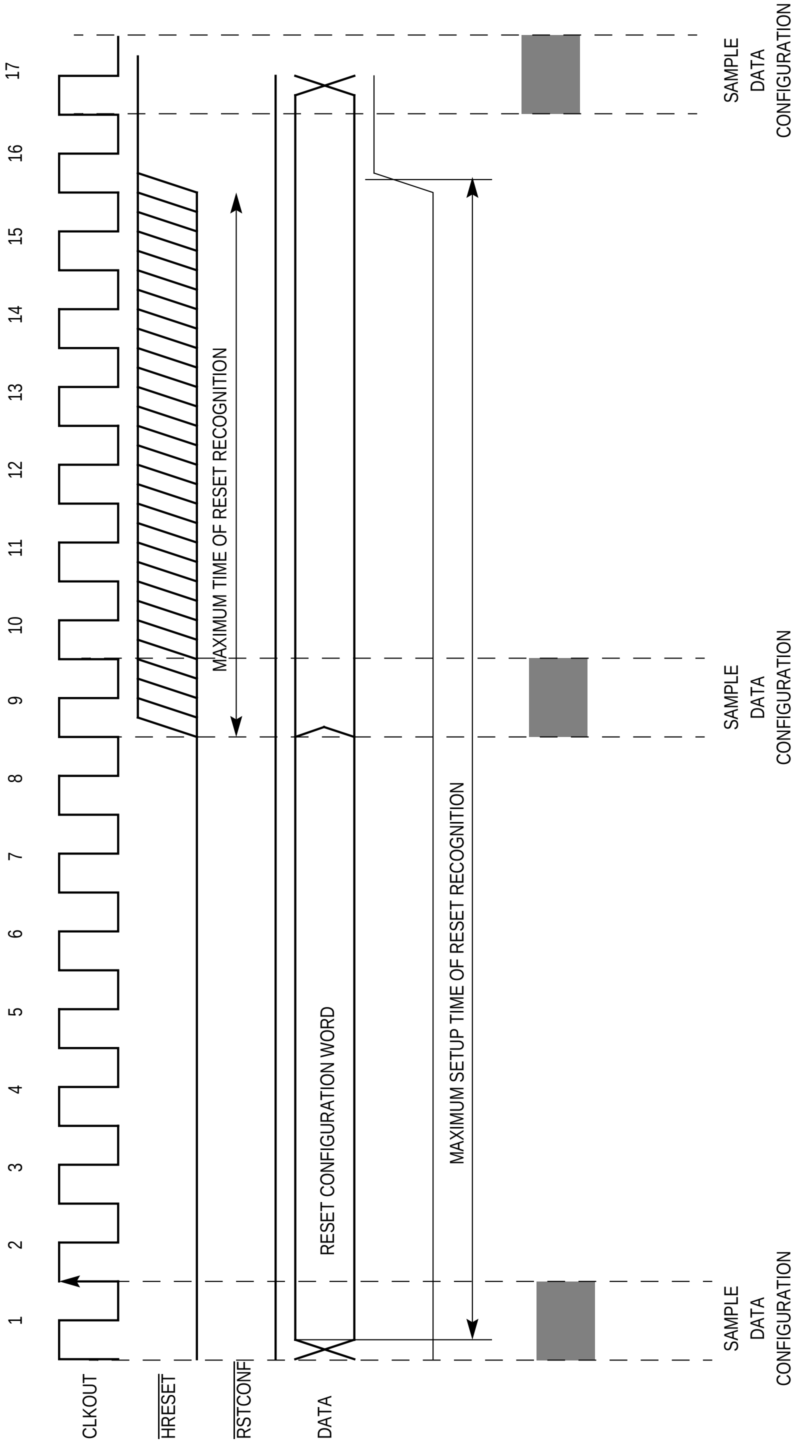

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
CLKOUT
HRESET
RSTCONF
DATA
MAXIMUM TIME OF RESET RECOGNITION
RESET CONFIGURATION WORD
MAXIMUM SETUP TIME OF RESET RECOGNITION
SAMPLE DATA CONFIGURATION
SAMPLE DATA CONFIGURATION
SAMPLE DATA CONFIGURATION

**Figure 4-4. Reset Configuration Sampling Timing Requirements**

### 4.3.1.1 HARD RESET CONFIGURATION WORD.

The following is the hard reset configuration word that is sampled from the data bus and default of the bits.

HARD RESET CONFIGURATION WORD

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | EARB | IP | RES | BDIS | BPS | | RES | ISB | | DBGC | | DBPC | | EBDF | | — |
| FIELD | | | BBE | | | | | | | | | | | | | CLES |
| RESET | 0 | 0 | 0 | 0 | 00 | | 0 | 00 | | 00 | | 00 | | 00 | | 0 |
| RESET | LAT | 1 | 1 | | | | | | | | | | | | | |
| R/W | | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | | |

| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | | | | | | | | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | | |

EARB—External Arbitration

If the EARB bit is set (1), external arbitration is assumed, but if it is cleared (0), then internal arbitration is performed. See **Section 12.4.1.1 SIU Module Configuration Register**.

$\overline{\text{IP}}$—Initial Interrupt Prefix

This bit defines the initial value of the $\text{MSR}_{\text{IP}}$ immediately after reset. The $\text{MSR}_{\text{IP}}$ bit defines the interrupt table location. If $\overline{\text{IP}}$ is zero (default), the $\text{MSR}_{\text{IP}}$ initial value is one, but if it is sampled one, the $\text{MSR}_{\text{IP}}$ initial value is zero. See **Section 6.4.1.2.1 Machine State Register Bits**.

BBE—Boot Burst Enable

If the BBE bit is set (1), the boot device supports bursting, but if it is cleared (0), the boot device is nonburstable.

BBE - Boot Burst Enable

If the BBE bit is set (1), then the boot device supports bursting. If it is cleared (0), then the boot device is non burstable. See **Section 15.3.2 General-Purpose Chip-Select Machine**.

BDIS—Boot Disable

If the BDIS bit is set (1), the memory controller is not activated after reset, but if it is cleared (0), the memory controller bank 0 is active immediately after reset so that it matches all addresses.

BPS—Boot Port Size

This field defines the port size of the boot device as shown in the following chart.

| BPS | PIN USAGE |
|---|---|
| 00 | 32-bit port size |
| 01 | 8-bit port size |
| 10 | 16-bit port size |
| 11 | Reserved |

ISB—Initial Internal Space Base Select

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

| ISB | IMMR INITIAL VALUE |
|---|---|
| 00 | $00000000 |
| 01 | $00F00000 |
| 10 | $FF000000 |
| 11 | $FFF00000 |

DBGC—Debug Pins Configuration

Refer to **Section 12.4.1.1 SIU Module Configuration Register** for this bit field's definition.

DBPC—Debug Port Pins Configuration

Refer to **Section 12.4.1.1 SIU Module Configuration Register** for the bit field's definition.

EBDF—External Bus Division Factor

Refer to **Section 5.8 System Clock Control** for the bit field's definition.

CLES—Core Little Endian Swap

If the LES bit is set (1), the little endian swapper at the EBI is activated for core accesses after reset. If it is cleared (0), the little endian swapper at the EBI is not activated for core accesses after reset. See **Section 14 Endian Modes** for more information.

## 4.3.2 Soft Reset Configuration

When a soft reset event occurs, the MPC821 reconfigures the development port. Refer to **Section 19.3.1.3 Entering Debug Mode** and **Section 19.3.2.3 Development Port Serial Communications–Clock Mode Selection** for more information.

# SECTION 5
# CLOCKS AND POWER CONTROL

## 5.1 OVERVIEW

The main timing reference for the MPC821 can be a high frequency crystal of 4 MHz, a low frequency crystal of 32 KHz, or an external frequency source at 4 MHz or at the system frequency. The on-chip PLL can multiply the output of the crystal circuit up to the final system frequency. A crystal circuit consists of a parallel resonant crystal, two capacitors, and two resistors. Notice that the values shown as example values are based on inhouse designs and your circuit might require slightly different values to operate properly. Crystals are typically much cheaper than similar speed oscillators, but they may not be as stable since they are affected by parameters like trace length, component quality, board layout, and MPC821 shrink level. For the most part, they are usually stable, but it is impossible to guarantee that they will remain that way because of MPC821 process changes or external component shifts.

**NOTE**

> The internal frequency of the MPC821 and the output of the CLKO pins is dependent on the quality of the crystal circuit and the multiplication factor (MF) used in the PLLCR. Please refer to the sections on phase-lock loop for a description of the PLL performance.

The system operating frequency is generated through a programmable phase-locked loop called the system PLL (SPLL). The SPLL is programmable in integer multiples of input oscillator frequency to generate the internal (VCO/2) operating frequency that should be at least 15 MHz. It can be divided by a power of two divider to generate the system operating frequencies. Another responsibility of the MPC821 and part of the clock section are the clocks to the timebase (TB), decrementer (DEC), real-time counter (RTC), and periodic interrupt counter (PIT).

The oscillator, TB, DEC, RTC, and PIT are all powered by the keep alive power supply (KAPWR) that allows the counters to continue counting (increment/decrement) at 32 KHz/4 MHz, even when the main power to the MPC821 is off. While the power is off, the PIT can be used to signal to the IC power supply that power should be sent to the system at specific intervals. This is the power-down wake-up feature.

When the chip is not in power-down low power mode, the KAPWR is powered to the same voltage value as that of the I/O buffers and logic. Therefore, if the internal power supply is 2 volts and the I/O buffers and logic voltage are 3.3 volts, the KAPWR is (2.9 ÷ 3.3) volts. For more details refer to **Section 5.2 Clock Unit Description** and **Section 5.11.1 Keep Alive Power Configuration**. Figure 5-1 illustrates the clock unit’s functional block diagram.

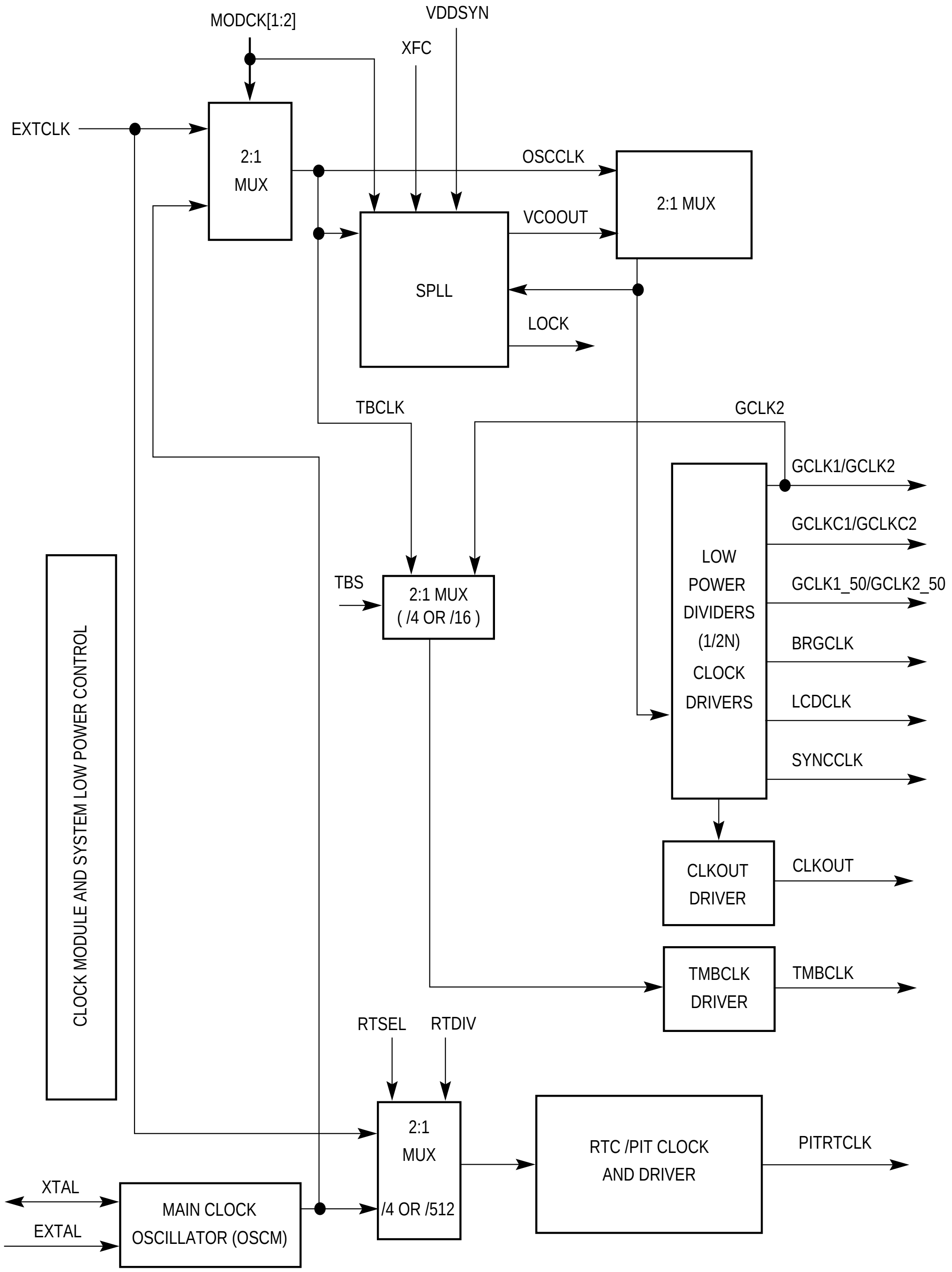


Figure 5-1. Clock Unit Block Diagram

## 5.2 CLOCK UNIT DESCRIPTION

The MPC821 clock module consists of the main crystal oscillator (OSCM), the SPLL, the low power divider, the clock generator/driver blocks, and the clock module/system low power control block. the clock module and system low power control block receive control bits from the system clock control register (SCCR), the PLL, the low power and reset control register (PLPRCR), and the reset control register (RSR).

To improve noise immunity, the charge pump and the VCO of the SPLL have their own set of power supply pins (VDDSYN and VSSSYN). KAPWR and VSS power the following clock unit modules—OSCM, pitrtclk and tmbclk generation logic, DB, DEC, RTC, PIT, SCCR, PLPRCR, and RSR. All other circuits are powered by the normal supply pins (VDDH/VDDL) and VSS. VDDH feeds the I/O buffers and logic and VDDL supplies the internal chip logic to reduce system power consumption. However, the power supply connected to VDDH should be larger or equal to the one connected to VDDL. The power supply for each block is listed in Table 5-1 below and described in **Section 5.10 Basic Power Structure**.

**Table 5-1. MPC821 Power Supply**

| | VDDH | VDDL | VDDSYN | KAPWR |
|---|---|---|---|---|
| IO Pad Logic | X | | | |
| CLKOUT | X | | | |
| SPLL (Digital) | X | | | |
| Clock Block | X | | | |
| Internal Logic | | X | | |
| Clock Drivers | | X | | |
| SPLL (Analog) | | | X | |
| MAIN OSC | | | | X |
| SCCR, PLLRCR and RSR | | | | X |
| RTC, PIT, TB, and DEC | | | | X |

NOTE: X Denotes That The Power Supply Is Used

The following are the relationships between different power supplies:

- VDDH = VDDSYN = 3.3 V ±10%
- VDDH $\geq$ VDDL $\geq$ 2.2 V ±10%
- VDDH $\geq$ KAPWR $\geq$ VDDH - 0.4 V (@ normal operation)
- KAPWR $\geq$ 2 V (@ power-down mode)

The timing diagram for the internal clocks generated in the MPC821 is illustrated in Figure 5-2.

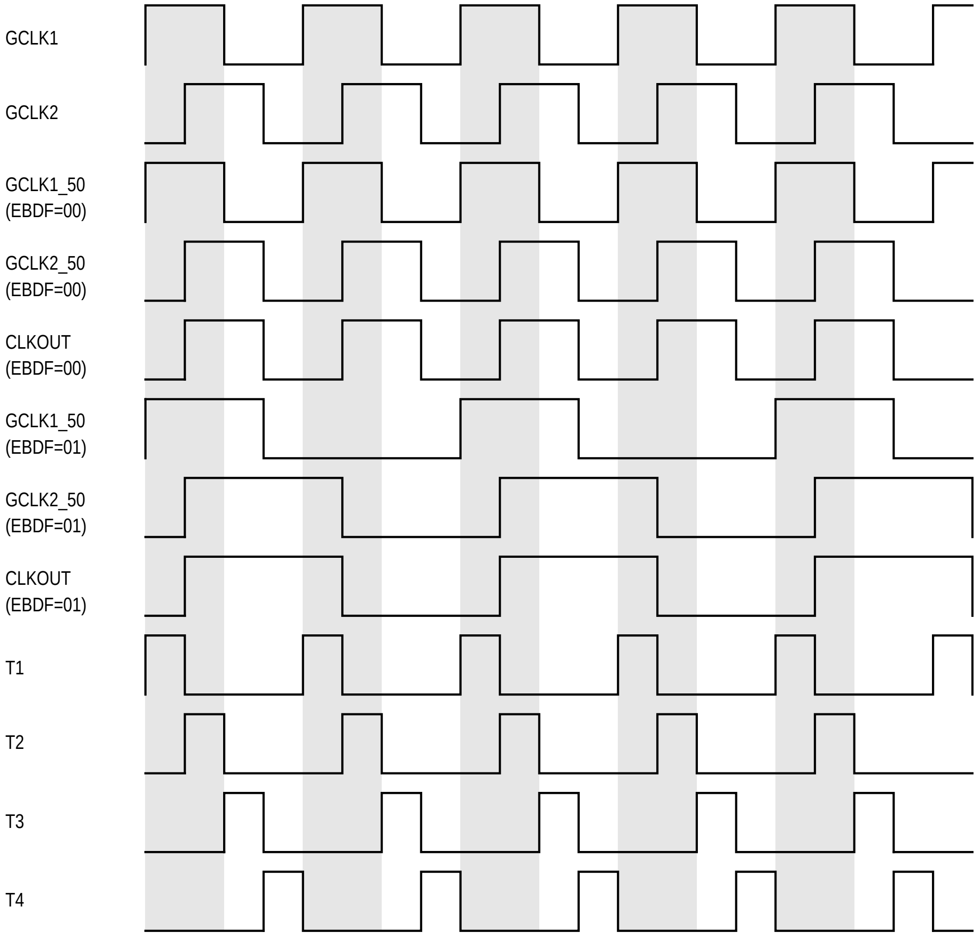


**Figure 5-2. MPC821 Clocks Timing Diagram**

Notice that GCLK1_50, GCLK2_50, and CLKOUT can have a lower frequency than GCLK1 and GCLK2. This allows the external bus to operate at lower frequencies as controlled by the EBDF bit in the SCCR. GCLK2_50 always rises simultaneously with GCLK2. If the MPC821 is working with DFNH = 0, GCLK2_50 has a 50% duty-cycle (with other values of DFNH or DFNL, the duty-cycle is less than 50%. GCLK1_50 rises simultaneously with GCLK1, but when the MPC821 is not in gear mode, the falling edge of GCLK1_50 occurs in the middle of the high phase of GCLK2_50 and EBDF determines the division factor between GCLK1/2 and GCLK1/2_50. Refer to Figure 5-6 for more information.

To configure the clock source for the SPLL and clock drivers, the MODCK1 and MODCK2 pins are sampled on the rising edge of the power-on reset line (PORESET). The configuration modes are shown in the table below. MODCK1 specifies the input source to the SPLL (OSCM or EXTCLK) and, combined with MODCK2, specifies the multiplication factor at reset. If the pitrtclk and tmbclk configuration and the SPLL multiplication factor must be unaffected in the power-down low power mode, the MODCK1 and MODCK2 bits should not be sampled on wake-up from this mode. In this case the PORESET pin should remain negated while the HRESET pin is asserted during the power-up wake-up stage.

### Table 5-2. Reset Clocks Source Configuration

| MODCK 1:2 | POR | DEFAULT MF + 1 @ POR | PITRTCLK DIVISION DEFAULTS @ POR | TMBCLK DIVISION DEFAULTS @ POR | SPLL OPTIONS |
|---|---|---|---|---|---|
| 00 | 0 | 513 | 4 | 4 | Normal operation, PLL enabled.<br>Main timing reference is freq$_{(OSCM)}$ = 32 KHz. |
| 01 | 0 | 5 | 512 | 4 | Normal operation, PLL enabled.<br>Main timing reference is freq$_{(OSCM)}$ = 4 MHz. |
| 11 | 0 | 5 | 512 | 4 | Normal operation, PLL enabled.<br>Main timing reference is freq$_{(EXTCLK)}$ = 4 MHz. |
| 10 | 0 | 1 | 512 | 16 | Normal operation, PLL enabled.<br>1:1 Mode (freq$_{clkout(max)}$ = freq$_{osc(EXTCLK)}$) |
| — | 1 | — | — | — | The configuration remains unchanged. |

When the MODCK1 bit is clear(O), the output of the OSCM (with a frequency of 4 MHz or 32 KHz) is selected to be the input of the SPLL, but when it is set, the external clock input (EXTCLK) is selected. In all cases, the system clock frequency (freq$_{gclk1}$) can be reduced by DFNH[0:2] and DFNL[0:2]) bits in the SCCR. Notice that the maximum system clock frequency occurs when the DFNH bits are set to $0.When the MODCK2 bit is set, a clock of 4 MHz is supplied as oscclk, but when it is clear(0), the input frequency is either 32 KHz (MODCK1=0) or the maximum system frequency (MODCK1=1). The last case is 1:1 mode.

If EXTCLK is the main timing reference (MODCK1=1 @POR) and OSCM is the timing reference to the RTC and PIT, the frequency of the oscillator connected to OSCM should be in the 32-KHz range. The timebase clock can be selected by the TBS bit in the SCCR to be either the SPLL input clock or gclk2. The periodic interrupt timer and real-time clock (PITRTCLK) frequency and source are specified by the RTDIV and RTSEL bits in the SCCR.

If the POR pin is negated (driven to the high value) the MODCK1 and MODCK2 reset values are not affected and they remain unchanged since the last power-on reset. The values of the pitrtclk and tmbclk clock divisions can be changed by the software. The RTDIV bit value in the SCCR defines the division of pitrtclk. All possible combinations of the tmbclk divisions are listed in Table 5-3.

Table 5-3. tmbclk Divisions

| TBS BIT IN SCCR | MODCK1 @ RESET | MF + 1 | TMBCLK DIVISION |
|---|---|---|---|
| 1 | — | — | 16 |
| 0 | 0 | — | 4 |
| 0 | 1 | 1, 2 | 16 |
| 0 | 1 | > 2 | 4 |

**NOTE**

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

## 5.3 ON-CHIP OSCILLATORS AND EXTERNAL CLOCK INPUT

The OSCM uses either a 3 MHz ÷ 5 MHz (4 MHz mode) or a 30 KHz ÷ 50 KHz (32 KHz mode) crystal to generate the PLL reference clock. When the OSCM output is the timing reference to the system, PLL skew elimination between the XTAL/EXTAL pins and CLKOUT is not guaranteed.

**NOTE**

The internal frequency of the MPC821 and the output of the CLKO pins is dependent on the quality of the crystal circuit and the multiplication factor (MF) used in the PLLCR. Please refer to the sections on phase-lock loop for a description of the PLL performance.

The external clock input receives a clock from an external source. The clock frequency can be either in the range of 3 MHz ÷ 5 MHz or it should be at the system frequency of at least 15 MHz (1:1 mode). When the external clock input is the timing reference to the system, PLL skew elimination between the EXTCLK pin and CLKOUT is less than ± 1 nanosecond.

For normal operation at least one clock source (EXTCLK or OSCM) should be active, but a configuration with both clock sources active is also possible. In this configuration, EXTCLK provides the oscclk and OSCM provides the pitrtclk. The input of an unused timing reference (EXTCLK or EXTAL) should be grounded.

## 5.4 SYSTEM PLL

The PLL performs frequency multiplication and skew elimination. It allows the processor to operate at a high internal clock frequency using a low frequency clock input, which is a feature with two immediate benefits. Lower frequency clock input reduces the overall electromagnetic interference generated by the system and oscillating at different frequencies reduces cost by eliminating the need to add another oscillator to the system. The MPC821 PLL block diagram is illustrated in Figure 5-3.

### 5.4.1 Frequency Multiplication

The PLL can multiply the input frequency by any integer between 1 and 4,096. The multiplication factor can be changed by changing the value of the MF[0:11] bits in the PLPRCR. While any integer value from 1 to 4,096 can be programmed, the resulting VCO output frequency must be in the range specified in **Section 21 Electrical Characteristics**. As defined in Table 5-6, the multiplication factor is set to a predetermined value during the power on reset (POR).

### 5.4.2 Skew Elimination

The PLL is capable of eliminating the skew between the external clock entering the chip (EXTCLK), the internal clock phases, and the CLKOUT pin. Therefore, the PLL is useful for tightening synchronous timings. Skew elimination is only active when the PLL is enabled and programmed with a multiplication factor of 1 or 2 (MF=0 or 1) and the timing reference to the system PLL is the external clock input EXTAL. With PLL disabled, the clock skew may be much larger.

### 5.4.3 PLL Block Diagram

The reference signal (oscclk) goes to the phase comparator that controls the direction (up or down) that the charge pump drives the voltage across the external filter capacitor (XFC). The direction selected depends on whether the feedback signal phase lags or leads the reference signal. The output of the charge pump drives the VCO whose output frequency is divided down and fed back to the phase comparator for comparison with the reference signal, oscclk. The MF values (0 to 4,095) are mapped to multiplication factors of 1 to 4,096. Also, when the PLL is operating in 1:1 mode, the multiplication factor is 1(MF=0) and the PLL output frequency is twice the maximum system frequency. This double frequency is required to generate the GCLK1 and GCLK2 clocks. Refer to the block diagram in Figure 5-3 for details.

On initial system power-up after KAPWR is lost, power-on reset ($\overline{\text{PORESET}}$) should be asserted by external logic for 3 microseconds after a valid level is reached on the KAPWR supply. Whenever power-on reset is asserted, the MF bits are set according to Table 5-2 and the DFNH and DFNL bits in SCCR are set to the value of 0 (÷1), respectively. This value then programs the SPLL to generate the default system frequency of approximately 16.7 MHz for a 32 KHz input frequency and 20 MHz for a 4 MHz input frequency.

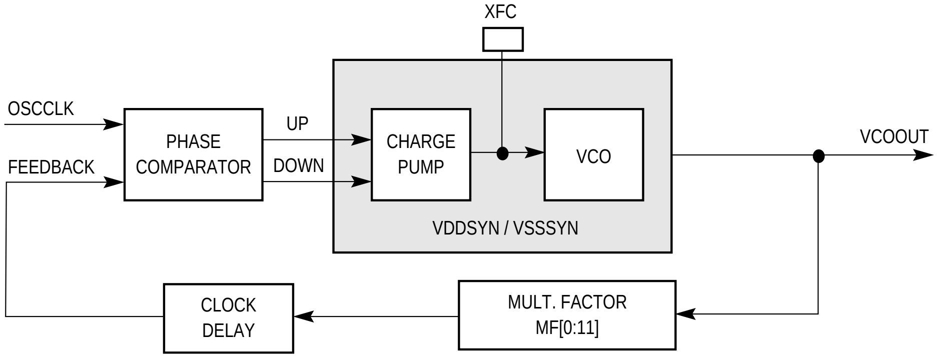


**Figure 5-3. System PLL Block Diagram**

## 5.5 LOW POWER DIVIDER

The output of the PLL is sent to a low power divider block. This block generates all other clocks in normal operation, but also has the ability to divide the output frequency of the VCO before it generates the SyncCLK, SyncCLKS, LCDCLK, LCDCLK50, BRGCLK, and general system clocks sent to the rest of the MPC821. GCLK1C and GCLK2C are the system timing references for the PowerPC core as well as the instruction and data caches and MMUs. GCLK1 and GCLK2 are the system timing references for all other modules. GCLK1_50 and GCLK2_50 can work at a frequency of half the GCLK1 and GCLK2 frequency. The frequency ratio between GCLK1/2 and GCLK1/2_50 is determined by the EBDF bit in the SCCR.

The purpose of the low power divider block is to allow the user to reduce and restore the operating frequencies of different sections of the MPC821 without losing the PLL lock. Using the low power divider block, the user can still obtain full chip operation, but at a lower frequency. This is called slow-go (gear) mode. The selection and speed of the slow-go mode can be changed at any time, with changes occurring immediately. The low power divider block is controlled in the SCCR and it’s default state is to divide all clocks by one. Thus, for a 40 MHz system, the SyncCLK, SyncCLKS, LCDCLK, LCDCLK50, BRGCLK, and general system clocks are each 40 MHz.

## 5.6 INTERNAL CLOCK SIGNALS

The internal logic of the MPC821 uses 11 internal clock lines:

- General system clocks (GCLK1C, GCLK2C, GCLK1, GCLK2, GCLK1_50, and GCLK2_50)
- Baud rate generator clock (BRGCLK)
- Synchronization clocks (SYNCCLK and SYNCCLKS)
- LCD clocks (LCDCLK and LCDCLK50)

The MPC821 also generates an external clock line called CLKOUT. The PLL synchronizes these clock signals to each other.

### 5.6.1 General System Clocks

The general system clocks (GCLK1C, GCLK2C, GCLK1, GCLK2, GCLK1_50, and GCLK2_50) are the basic clocks supplied to all modules and submodules on the MPC821. GCLK1C and GCLK2C are supplied to the PowerPC core, data and instruction caches, and MMUs and they can be stopped when the chip enters the doze low power mode. GCLK1 and GCLK2 are supplied to the SIU, clock module, RISC controller, and most other features in the CPM.

The external bus clock GCLK2_50 is the same as CLKOUT. The general system clock defaults to VCO/2 = 40 MHz (assuming a 40 MHz system frequency). In slow-go mode, the frequency of the general system clock can be dynamically changed with the SCCR. Refer to Figure 5-4 below for details.

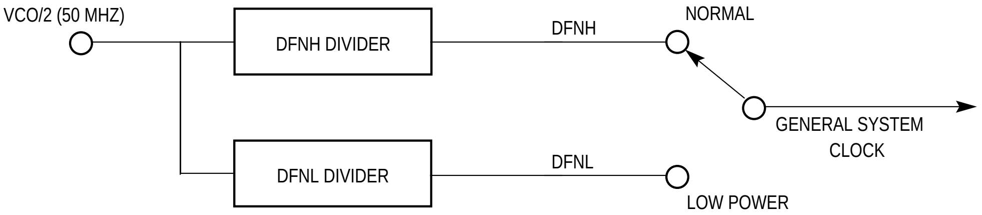


**Figure 5-4. General System Clocks Select**

The general system clock frequency can be switched between different values. The highest operational frequency can be achieved when the system clock frequency is determined by DFNH (CSRC bit in the PLPRCR is cleared) and DFNH=0 (divided by one). The general system clock can be operated at a low or high frequency. Low is defined by the DFNL bits in the SCCR and high is defined by the DFNH bits.

Software can change the frequency of the general system clock on-the-fly. The user can cause the general system clock to switch to it's low frequency. However, in some applications, a high frequency is needed during certain periods. For example, interrupt routines require more performance than the low frequency operation provides, but consumes less power than maximum frequency operation provides. The MPC821 is capable of automatically switching between low and high frequency operation whenever one of the following conditions exists:

- A pending interrupt from the interrupt controller occurs. This option is maskable by the PRQEN bit in the SCCR.
- The power management (POW) bit in the MSR of the PowerPC core is clear (normal operation). This option is maskable by the PRQEN bit in the SCCR.
- The CPM RISC controller has a pending request or is currently executing a routine (it is not idle). This option is maskable by the CRQEN bit in the SCCR.

When none of these conditions exist and the CSRC bit in PLPRCR is set, the general system clock automatically switches back to the low frequency. When the general system clock is divided, it's duty-cycle is changed. One phase remains the same (12.5 nanoseconds @ 40 MHz) while the other becomes longer. Notice that CLKOUT no longer has a 50% duty cycle when the general system clock is divided. The CLKOUT waveform is the same as that of GCLK2_50.

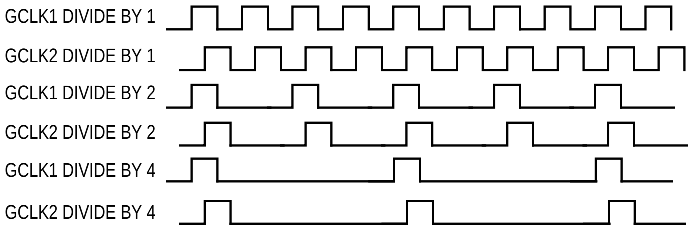


**Figure 5-5. Divided System Clocks Timing Diagram**

The frequency for system clocks GCLK1and GCLK2 is:

$$\text{FREQsys} = \frac{\text{FREQsysmax}}{\left(2^{\text{DFNH}}\right)\text{or}\left(2^{\text{DFNL}+1}\right)}$$

The frequency for clocks GCLK1_50 and GCLK2_50 is:

$$\text{FREQ50} = \frac{\text{FREQsysmax}}{\left(2^{\text{DFNH}}\right)\text{or}\left(2^{\text{DFNL}+1}\right)} \times \frac{1}{\text{EBDF}+1}$$

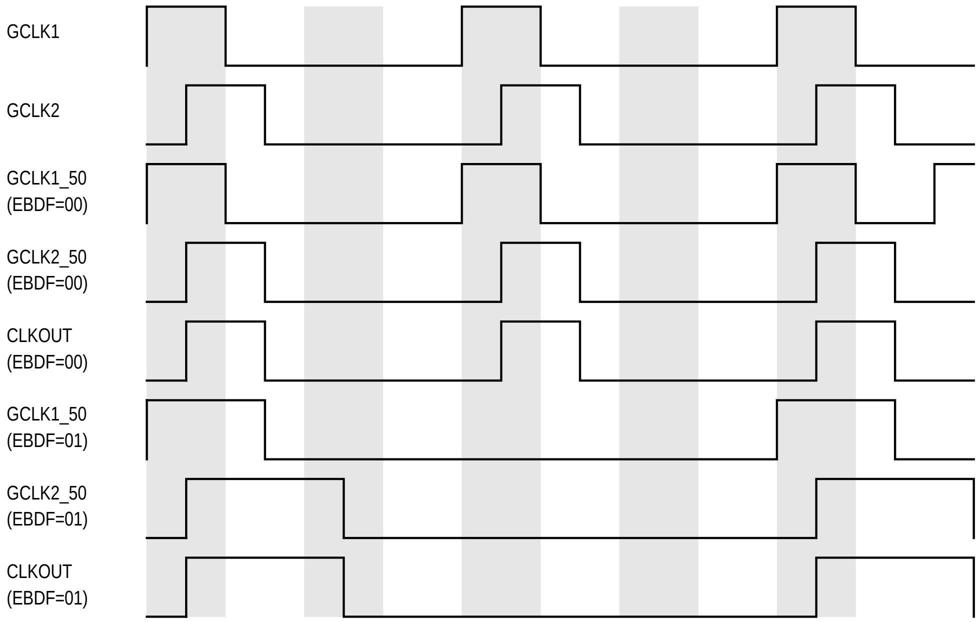


**Figure 5-6. MPC821 Clocks For DFNH = 1 (or DFNL = 0) Timing Diagram**

BRGCLK is used by the four baud rate generators of the CPM and by the memory controller refresh counter. It defaults to VCO/2 = 40 MHz, assuming a 40-MHz system frequency. The purpose of BRGCLK is to allow the four baud rate generators s to continue operating at a fixed frequency, even when the rest of the MPC821 is operating at a reduced frequency (the general system clock is divided). Refer to the baud rate generators description in **Section 16 Communication Processor Module** for more information on how to save power using the BRGCLK. The BRG clock frequency is:

$$\mathrm{FREQbrg} = \frac{\mathrm{FREQsysmax}}{\left(2^{2 \times \mathrm{DFBRG}}\right)}$$

syncCLK is used by the serial synchronization circuitry in the serial ports of the CPM and includes the SI, SCCs, and SMCs. The syncCLK performs the function of synchronizing externally generated clocks before they are used internally. syncCLK defaults to VCO/2 = 40 MHz, assuming a 40-MHz system frequency. syncCLKS is used by the SI internal logic.

The purpose of syncCLK is to allow the SI, SCCs, and SMCs to continue operating at a fixed frequency, even when the rest of the MPC821 is operating at a reduced frequency. Thus, the syncCLK allows the user to maintain the serial synchronization circuitry at the preferred rate, while lowering the general system clock to the lowest possible rate. However, syncCLK must always have a frequency at least as high as the general system clock frequency, be at least two times the preferred serial clock rate, and at least two and half times the preferred serial clock rate if the time-slot assigner in the SI is used. Refer to the SI description in **Section 16 Communication Processor Module** for more information on how to select an appropriate frequency for the syncCLK. The SYNC clock frequency is:

$$\mathrm{FREQsync} = \frac{\mathrm{FREQsysmax}}{\left(2^{2 \times \mathrm{DFSYNC}}\right)}$$

LCDCLK is used by the LCD controller circuitry to transfer the frame data to pixel format data. LCDCLK defaults to VCO/2 = 40 MHz, assuming a 40-MHz system frequency. When the PON bit in LP_HCR is set, the ratio between the system clock frequency value and the LCD clock value should be an integer value ($\mathrm{freq}_{\mathrm{syst}}$ / $\mathrm{freq}_{\mathrm{lcd}}$=Integer>0). The LCD clock frequency is the system frequency divided through two serial dividers. LCDCLK50 is a 50% duty-cycle clock at the same frequency as LCDCLK and it is used as a clock output to the LCD panel. DFLCD and DFALCD should be set in a way that the total LCD clock division factor never exceeds 64. The LCD clock frequency is:

$$\mathrm{FREQlcd} = \frac{\mathrm{FREQsysmax}}{\left(2^{\mathrm{DFLCD}}\right) \times (2 \times \mathrm{DFALCD} + 1)}$$

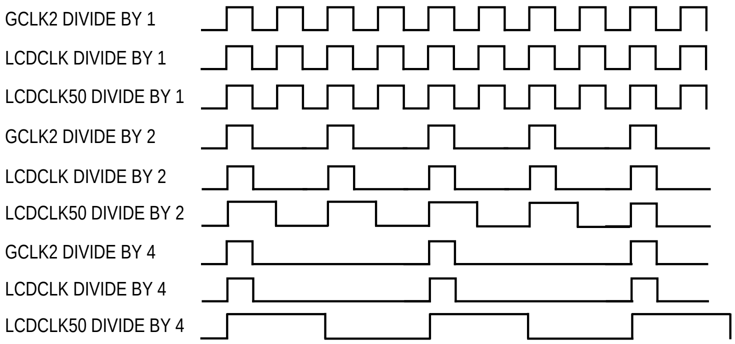


**Figure 5-7. LCD Clocks Timing Diagram**

CLKOUT is the same as GCLK2_50. It defaults to VCO/2 = 40 MHz, assuming a 40-MHz system frequency. CLKOUT can drive full strength, half strength, or be disabled. The drive strength is controlled in the SCCR. Disabling or decreasing the strength of CLKOUT can reduce power consumption, noise and electromagnetic interference on the printed circuit board. When the PLL is acquiring lock, the CLKOUT signal is disabled and remains in the low state.

## 5.7 PLL PINS

The following pins are dedicated to the PLL operation.

**NOTE**

> The internal frequency of the MPC821 and the output of the CLKO pins is dependent on the quality of the crystal circuit and the multiplication factor (MF) used in the PLLCR. Please refer to the sections on phase-lock loop for a description of the PLL performance.

VDDSYN—Drain Voltage

This is the VDD dedicated to the analog PLL circuits. The voltage should be well-regulated and the pin should be provided with an extremely low-impedance path to the VDD power rail. VDDSYN should be bypassed to VSSSYN by a 0.1µF capacitor located as close as possible to the chip package.

VSSSYN—Source Voltage

This is the VSS dedicated to the analog PLL circuits. The pin should be provided with an extremely low impedance path to ground and should be bypassed to VDDSYN by a 0.1µF capacitor located as close as possible to the chip package. It is requested that the user also bypass VSSSYN to VDDSYN with a 0.01uF capacitor as close as possible to the chip package.

VSSSYN1—Source Voltage

This is the VSS dedicated to the analog PLL circuits. The pin should be provided with an extremely low-impedance path to ground.

XFC—External Filter Capacitor

Connects to the off-chip capacitor for the PLL filter. One terminal of the capacitor is connected to XFC while the other terminal is connected to VDDSYN.

**NOTE**

> 30 MΩ is the minimum parasitic resistance value that ensures proper PLL operation when connected in parallel with the XFC capacitor.

**Table 5-4. XFC Capacitor Values**

| | MIN CAPACITANCE | MAX CAPACITANCE | UNIT |
|---|---|---|---|
| MF < = 4 | XFC = MF * 425 - 125 | XFC = MF * 590 - 175 | pF |
| MF > 4 | XFC = MF * 520 | XFC = MF * 920 | pF |

## 5.8 SYSTEM CLOCK CONTROL

The SPLL has a 32-bit control register that is powered by keep alive power. The SCCR is memory mapped into the MPC821 SIU register map. Table 5-5 provides an overview of this register and includes a quick reference of each bit field encoding.

**Table 5-5. System Clock and Reset Control Register**

| BITS | MNEMONIC | POWER-ON RESET DEFAULT VALUE | HARD RESET DEFAULT VALUE | DESCRIPTION | FUNCTION |
|---|---|---|---|---|---|
| 0 | Reserved | | | | |
| 1-2 | COM[0:1] | 00 | Unaffected | Clock output mode | 00 = Clock output enabled full-strength output buffer<br>01 = Clock output enabled half-strength output buffer<br>10 = Reserved<br>11 = Clock output disabled |
| 3-5 | Reserved | | | | |
| 6 | TBS | 0 | Unaffected | TBS: Timebase source | 0 = Source is oscclk divided by (4 or 16)<br>1 = Source is system clock divided by (16) |
| 7 | RTDIV | MODCK1 & MODCK2 | Unaffected | RTC clock divide | 0 = RTC and PIT clock divided by 4<br>1 = RTC and PIT clock divided by 512 |
| 8 | RTSEL | MODCK[1] | Unaffected | RTC circuit input source select | 0 = OSCM clock is selected as input to RTC and PIT<br>1 = EXTCLK clock is selected as input to RTC and PIT |
| 9 | CRQEN | 0 | 0 | CPM request enable | 0 = Remains in the lower frequency (defined by the DFNL) even if the CPM is active (not idle).<br>1 = Switches to high frequency (defined by DFNH) when the CPM needs to execute a routine. |
| 10 | PRQEN | 0 | 0 | Power management request enable | 0 = Remains in the lower frequency (defined by the DFNL) even if the power management bit in the MSR is reset (normal operational mode) or if there is a pending interrupt from the interrupt controller.<br>1 = Switches to high frequency (defined by the DFNH) when the power management bit in the MSR is reset (normal operational mode) or when there is a pending interrupt from the interrupt controller. |
| 11-12 | Reserved | | | | |
| 13-14 | EBDF[0:1] | | D(13:14) | CLKOUT frequency | 00 = CLKOUT is GCLK2 divided by 1<br>01 = CLKOUT is GCLK2 divided by 2<br>1x = Reserved |

Table 5-5. System Clock and Reset Control Register (Continued)

| BITS | MNEMONIC | POWER-ON RESET DEFAULT VALUE | HARD RESET DEFAULT VALUE | DESCRIPTION | FUNCTION |
|---|---|---|---|---|---|
| 15-16 | Reserved | | | | |
| 17-18 | DFSYNC[0:1] | 00 | 00 | Division factor of SyncCLK | 00 = Divide by 1 (normal operation)<br>01 = Divide by 4<br>10 = Divide by 16<br>11 = Divide by 64 |
| 19-20 | DFBRG[0:1] | 00 | 00 | Division factor of BRGCLK | 00 = Divide by 1 (normal operation)<br>01 = Divide by 4<br>10 = Divide by 16<br>11 = Divide by 64 |
| 21-23 | DFNL[0:2] | 000 | 000 | Division factor low frequency | 000 = Divide by 2<br>001 = Divide by 4<br>010 = Divide by 8<br>011 = Divide by 16<br>100 = Divide by 32<br>101 = Divide by 64<br>110 = Reserved<br>111 = Divide by 256 |
| 24-26 | DFNH[0:2] | 000 | 000 | Division factor high frequency | 000 = Divide by 1<br>001 = Divide by 2<br>010 = Divide by 4<br>011 = Divide by 8<br>100 = Divide by 16<br>101 = Divide by 32<br>110 = Divide by 64<br>111 = Reserved |
| 27-29 | DFLCD[0:2] | 000 | 000 | Division factor of LCD clock | 000 = Divide by 1<br>001 = Divide by 2<br>010 = Divide by 4<br>011 = Divide by 8<br>100 = Divide by 16<br>101 = Divide by 32<br>110 = Divide by 64<br>111 = Reserved |
| 30-31 | DFALCD[0:1] | 00 | 00 | Additional LCD clock division factor | 00 = Divide by 1<br>01 = Divide by 3<br>10 = Divide by 5<br>11* = Divide by 7 |

COM[0:1]—Clock Output Mode

These bits control the output buffer strength of the CLKOUT pin. When both bits are set, the CLKOUT pin is held in the high (1) state. These bits can be dynamically changed without generating spikes on the CLKOUT pin. If the CLKOUT pin is not connected to external circuits, set both bits (disabling CLKOUT) to minimize noise and power dissipation. The COM bits are cleared by a hard reset.

TBS—Timebase Source

This bit controls which clock source drives the timebase and decrementer. Refer to Table 5-4 for more information.

0 = TB frequency source is the crystal oscillator frequency divided by 4 or 16.
1 = TB frequency source is the system clock divided by 16.

RTDIV—RTC Clock Divide

This bit indicates if the clock to RTC and PIT is additionally divided by 128. At power-on reset this bit is cleared if both MODCK[1] and MODCK[2] are zeros, otherwise it is set.

RTSEL—RTC Circuit Input Source Select

This bit specifies the input source to the RTC. At the power-on reset, RTSEL receives the value of the MODCK[1] bit.

CRQEN—CPM Request Enable

This bit is cleared by power-on or hard reset and specifies if the general system clock returns to the high frequency (defined by the DFNH bits) while the CPM RISC controller is active and not idle.

0 = The system remains in the lower frequency (defined by the DFNL bits) even if the CPM is active.
1 = The system switches to the high frequency (defined by the DFNH bits) when CPM is active.

PRQEN—Power Management Request Enable

This bit specifies whether or not the general system clock returns to the high frequency (defined by the DFNH bits) while there is a pending interrupt from the interrupt controller or the power management (POW) bit in the MSR is clear (normal operational mode). The PRQEN bit is cleared by power-on or hard reset.

0 = The system remains in the lower frequency (defined by the DFNL bits) even if there is a pending interrupt from the interrupt controller or the power management bit in the MSR is cleared (normal operational mode).
1 = The system switches to high frequency (defined by the DFNH bits) when there is a pending interrupt from the interrupt controller or the power management bit in the MSR is cleared.

EBDF[0:1]—External Bus Division Factor

These bits define the frequency division factor betweenGCLK1/GCLK2 and GCLK1_50/GCLK2_50. CLKOUT is similar to GCLK2_50. The GCLK2_50 and GCLK1_50 are used by the external; the bus interface, the memory controller, and the PCMCIA interface to interface to the external system. The EBDF bits are initialized during $\overline{\text{HRESET}}$ using the hard reset configuration mechanism.

DFSYNC[0:1]—Division Factor for the SyncCLK

These bits define the SyncCLK and SyncCLKS frequencies. Changing the value of these bits does not result in a loss of lock condition. These bits are cleared by the power-on or hard reset.

DFBRG[0:1]—Division Factor for the BRGCLK

These bits define the BRGCLK frequency. Changing the value of these bits does not result in a loss of lock condition. These bits are cleared by the power-on or hard reset.

DFNL[0:2]—Division Factor Lowest Frequency

These bits are required in two cases—to reduce the general system clock to a frequency lower than that which can be obtained in DFNH and to automatically switch between the DFNL and DFNH rates. Refer to **Section 5.6.1 General System Clocks** and Figure 5-4 for more details. These bits are cleared by the power-on or hard reset.

The user can load these bits with the preferred divide value and then set the CSRC bit to change the frequency. Changing the value of these bits does not result in a loss of lock condition. These bits are cleared by system reset.

DFNH[0:2]—Division Factor High Frequency

Changing the value of these bits does not result in a loss of lock condition. These bits are cleared by the power-on or hard reset. The user can load these bits at any time to change the general system clock rate.

DFLCD[0:2]—Division Factor and DFALCD[0:]—Additional Division Factor for the LCDCLK

These bits define the LCDCLK and LCDCLK50 frequencies. The total division factor of the LCD clocks is defined in **Section 5.6.1 General System Clocks**. DFLCD and DFALCD should be set in a way that the total LCD clock division factor does not exceed 64. Changing the value of these bits does not result in a loss of lock condition. These bits are cleared by the power-on or hard reset.

## 5.9 PLL, LOW POWER, AND RESET CONTROL REGISTER

The PLL, low power, and reset control register (PLPRCR) is a 32-bit register powered by keep alive power supply. Table 5-6 provides an overview of this register, which is memory-mapped into the MPC821 SIU register map.

**Table 5-6. PLL, Low Power and Reset Control Register**

| BITS | MNEMONIC | POWER-ON RESET DEFAULT VALUE | HARD RESET DEFAULT VALUE | DESCRIPTION | FUNCTION |
|---|---|---|---|---|---|
| 0-11 | MF[0:11] | 0 or 4 or 512 | Unaffected | Multiplication factor bits | |
| 12-15 | Reserved | | | | |
| 16 | SPLSS | 0 | Unaffected | SPLL lock status sticky bits | 0 = SPLL has remained in lock<br>1 = SPLL has gone out of lock at least once (not due to change of the PLLEN or MF bits) |
| 17 | TEXPS | 1 | 1 | TEXP status bit | 0 = TEXP is negated<br>1 = TEXP is asserted |
| 18 | Reserved | | | | |
| 19 | TMIST | 0 | 0 | Timers interrupt status | 0 = No timer expiration event is detected<br>1 = A timer expiration event is detected |
| 20 | Reserved | | | | |
| 21 | CSRC | 0 | 0 | Clock source bit | 0 = General system clock is determined by the DFNH value<br>1 = General system clock is determined by the DFNL value |
| 22-23 | LPM[0:1] | 00 | 00 | Low power mode select bits | |
| 24 | CSR | 0 | Unaffected | Checkstop reset enable | 0 = Checkstop condition does not cause automatic reset<br>1 = Checkstop condition causes automatic reset |
| 25 | LOLRE | 0 | Unaffected | Loss of lock reset enable | 0 = Loss of Lock does not cause reset<br>1 = Loss of Lock does cause reset |
| 26 | FIOPD | 0 | Unaffected | Force I/O pull-down | 0 = The address and data pins are not driven by an internal pull-down device<br>1 = The address and data pins are driven by an internal pull-down device |
| 27-31 | Reserved | | | | |

MF[0:11]—Multiplication Factor

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

The MF bits can be read and written at any time. However, the MF bit field can be write-protected by setting the MF lock (MFL) bit in the PLPRCR. Changing the MF bits causes the SPLL to lose lock. All clocks are disabled until PLL reaches lock condition.

The normal reset value for the DFNH bits is $0 or divided by one. When the PLL is operating in 1:1 mode, the multiplication factor is set to 1 (MF=0). See Table 5-2 for details.

SPLSS—System PLL Lock Status Sticky Bit

SPLSS is not affected by hard reset. An out-of-lock indication sets the SPLSS bit and it remains set until the software clears it. At power-on reset, the state of the SPLSS bit is zero. The SPLSS bit is negated by writing a one to SPLSS (writing a zero has no effect on this bit). The SPLSS bit is not affected due to software-initiated loss of lock, which is defined as an MF change or entering deep sleep and power down modes.

TEXPS—Timer Expired Status Bit

The TEXPS is set by a reset. If enabled, the TEXPS is asserted when the periodic timer expires, the real-time clock alarm sets, timebase clock alarm is set, or the decrementer interrupt occurs. The bit stays set until the software clears it. The TEXPS bit is negated by writing a one to TEXPS (writing a zero has no effect on this bit). When TEXPS is set, the TEXP external signal is asserted and when TEXPS is reset the TEXP external signal is negated.

TMIST—Timers Interrupt Status Bit

TMIST is cleared at reset. TMIST is set when one of the following interrupts occur—RTC, PIT, TB, or DEC. It is cleared by writing a 1 to TMIST (writing a zero has no effect on this bit). The system clock frequency remains at a high frequency value (defined by DFNH) if the TMIST bit is set. The clock frequency remains high if the CSRC bit in the PLPRCR is set (DFNL enabled) and conditions to switch to normal low mode do not exist. For more information refer to the section on LPM(0:1)—Low Power Modes.

CSRC—Clock Source Bit

The CSRC bit specifies which bit will determine the general system clock—DFNH or DFNL. Setting this bit switches the general system clock to the DFNL value (for entering the slow-go low power mode). Clearing this bit switches the general system clock to the DFNH value. CSRC is cleared at hard reset.

LPM(0:1)—Low Power Modes

The LPM bits are encoded to provide one normal operating mode and four low power modes. In normal and doze modes the system can be in the high state defined by the DFNH bits or in the low state defined by the DFNL bits. The normal high operating mode is the state out of reset. This is also the state the bits defer to when the low power mode exit signal arrives. In addition, there are four low power modes—doze, sleep, deep sleep, and power-down. For more details, see the table below.

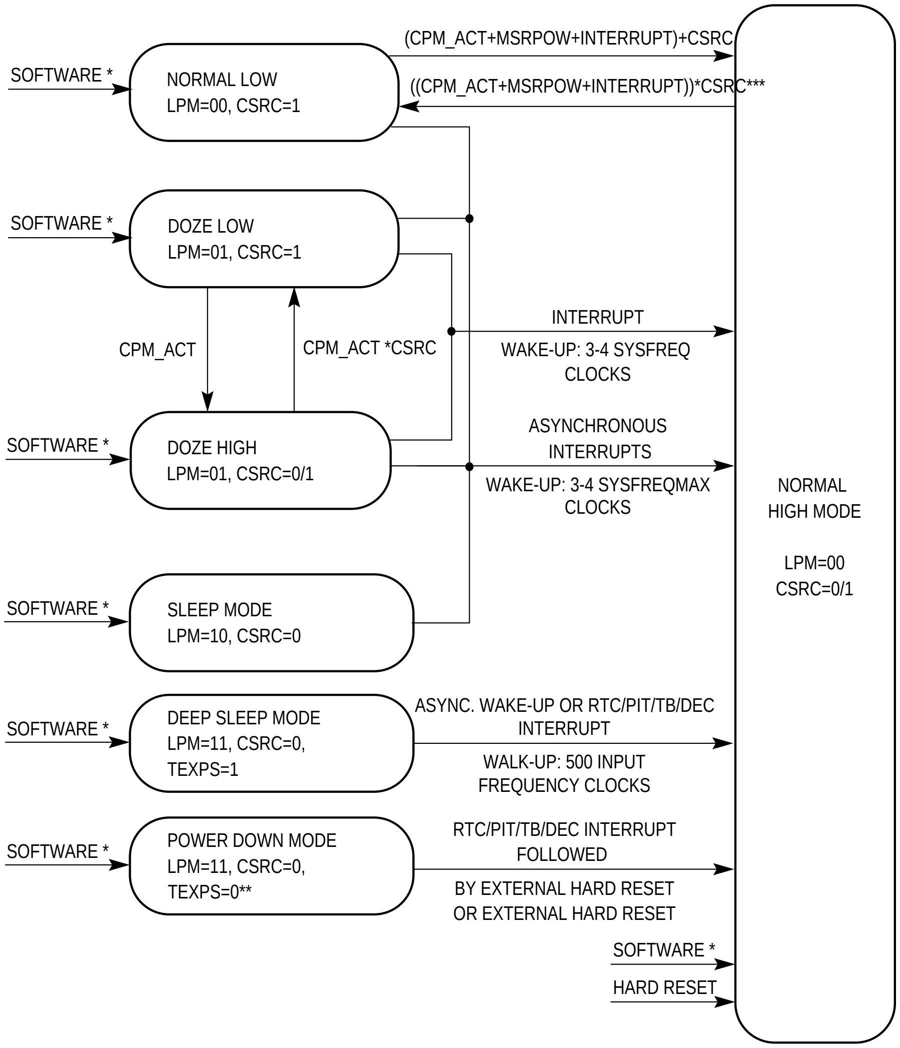
**Table 5-7. MPC821 Low Power Modes**

| OPERATION MODE | SPLL | CLOCKS | WAKE-UP METHOD | RETURN TIME FROM WAKE-UP EVENT TO NORMAL HIGH | MPC821 POWER CONSUMPTION @ 50 MHZ | FUNCTIONALITY |
|---|---|---|---|---|---|---|
| Normal High<br>LPM=00 | Active | Full<br>Freq. div<br>$2^{DFNH}$ | — | — | ≈ 20 mWatt+<br>$1/2^{DFNH}$ Watt | Full Functions Not In Use Are Shut Off |
| Normal Low<br>(“Gear”)<br>LPM=00 | Active | Full<br>Freq. div<br>$2^{DFNL+1}$ | Software<br>or<br>Interrupt | Asynchronous Interrupts:<br>3-4 Maximum System Cycles<br>Synchronous Interrupts:<br>3-4 Actual System Cycles | ≈ 20 mWatt+<br>$1/2^{(DFNL+1)}$ Watt | |
| Doze High<br>LPM=01 | Active | Full<br>Freq. div<br>$2^{DFNH}$ | Interrupt | | ≈ 20mWatt+<br>$0.4/2^{DFNH}$ Watt | Enabled: RTC, PIT, LCD, MEMC, CPM<br>Disabled: Extended Core |
| Doze Low<br>LPM=01 | Active | Full<br>Freq. div<br>$2^{DFNL+1}$ | Interrupt | | ≈ 20 mWatt+<br>$0.4/2^{(DFNL+1)}$ Watt | |
| Sleep<br>LPM=10 | Active | Not<br>Active | Interrupt | 3-4 Maximum System clocks | <10 mW | Enabled: RTC, PIT, TB and DEC |
| Deep Sleep<br>LPM=11<br>TEXPS=1 | Not<br>Active | Not<br>Active | Interrupt | <500 Oscillator Cycles<br>16msec-32 KHz<br>125 µsec-4 MHz | TBD | |
| Power Down<br>LPM=11<br>TEXPS=0 | Not<br>Active | Not<br>Active | Interrupt | (<500 Oscillator Cycles + Power Supply Wake-Up)<br>(PwSp_Wake+ 16msec) @ 32 KHz | 32 KHz- ~10 µA,<br>KAPWR = 3.0 V<br>Temperature=50 C | |

Table 5-7 describes all possible transfers between low power modes. The MPC821 enters a low power mode by setting the LPM bits appropriately. This can only be done in one of the normal modes and not in the doze mode. Exiting from low power modes occurs through an asynchronous or synchronous interrupt. An enabled asynchronous interrupt clears the LPM bits, but does not change the PLPRCR$_{CSRC}$ bit.

The exit from normal low, doze high, low, and sleep modes to normal high mode is done by the asynchronous interrupt. The sources of the asynchronous interrupt are:

- Asynchronous wake-up interrupt from the interrupt controller
- RTC, PIT, TB, or DEC interrupts (if enabled)



* SOFTWARE IS ACTIVE ONLY IN NORMAL HIGH/LOW MODES

** TEXPS RECEIVES THE ZERO VALUE BY WRITING ONE. WRITING A ZERO HAS NO EFFECT ON TEXPS.

*** THE SWITCH FROM NORMAL HIGH TO NORMAL LOW IS ENABLED ONLY IF THE CONDITIONS TO ASYNCHRONOUS INTERRUPT ARE CLEARED

**Figure 5-8. MPC821 Low Power Modes Flowchart**

The system responds quickly to an asynchronous interrupt. The wake-up time from normal low, doze high/low, and sleep mode due to an asynchronous interrupt is only 3 to 4 clocks of maximum system frequency. In a 40-MHz system, this wake-up takes 75 to 100 ns. The asynchronous wake-up interrupt from the interrupt controller is level sensitive. Therefore, it is negated only after the cause of the interrupt in the interrupt controller is cleared. The RTC, PIT, TB, or DEC interrupts set status bits in the PLPRCR (TMIST). The clock module views this interrupt as a pending asynchronous interrupt as long as the TMIST bit is set. Therefore, the TMIST status bit should be cleared before entering any low power mode other than normal high mode.

The wake-up time due to synchronous interrupt sources from the interrupt controller is measured in actual system clocks. It takes 2 to 4 system clocks from the interrupt event before the system reaches normal high mode. In a 50-MHz system with DFNL=111 (divided by 256) the wake-up time is 12.8 to 25.6 µs. In normal and doze modes, if the $\text{PLPRCR}_{\text{CSRC}}$ bit is set, the system toggles between low frequency (defined by the DFNL bit) and high frequency (defined by DFNL/DFNH) states. The conditions to switch from normal low mode to normal high state are:

- The CPM is active and not idle
- A pending interrupt from an interrupt controller occurs
- The power management enable bit in the MSR is cleared (normal operation mode)

When none of the above conditions exist, the $\text{PLPRCR}_{\text{CSRC}}$ bit is set, and the asynchronous interrupt status bits are reset, the system switches back to normal low mode. If the CPM is active and not idle, the system switches from doze low mode to doze high mode. On the other hand, when the CPM is idle, the $\text{PLPRCR}_{\text{CSRC}}$ bit is set and the system switches back to doze low mode. A pending interrupt from the interrupt controller transfers the system from doze mode to normal high mode. An exit from deep sleep mode is made by:

- An asynchronous wake-up interrupt from interrupt controller
- RTC, PIT, TB, or DEC interrupts (if enabled)

In deep sleep mode the PLL is disabled and, therefore, the wake-up time from this mode is up to 500 PLL input frequency clocks. In 1:1 mode the wake-up time can be up to 1,000 PLL input frequency clocks. If the PLL input frequency is 32 KHz the wake-up time is less then 15.6 ms and if it is 4 MHz the wake-up time is less then 125 µs.

The exit from power-down mode is accomplished with a hard reset that should be asserted by external logic in response to the TEXPS bit and TEXP pin assertion. The TEXPS bit is asserted by an enabled RTC, PIT, timebase, or decrementer interrupt. The hard reset should be longer than the time it takes the power supply to wake up, in addition to the PLL lock time. Hard reset assertion when the TEXPS bit and TEXP pin is clear, sets the bit and the pin values, causing an exit from power-down low power mode. For more details on power-down mode, refer to **Section 5.11.1 Keep Alive Power Configuration**.

**NOTE**

> The chip is only allowed to enter deep sleep low power mode and power-down mode if the main timing reference is 32 KHz crystal oscillator (MODCK@POR=00).

CSR—Checkstop Reset Enable

If this bit is set, then an automatic reset is generated when the CPU core signals that it has entered checkstop mode, unless debug mode is enabled at reset. If the bit is clear and debug mode is not enabled, then the SIU does nothing when a checkstop signal is received from the CPU core. If debug mode is enabled, then the part enters debug mode when entering checkstop mode. In this case, the CPU core does not assert the checkstop signal to the reset circuitry.

LOLRE—Loss of Lock Reset Enable

The LOLRE indicates how the clocks should handle a loss of lock indication. When LOLRE is clear, a hard reset is not asserted if a loss of lock indication occurs. However, when LOLRE is set, a hard reset ($\overline{\text{HRESET}}$) is asserted when a loss of lock indication occurs.

FIOPD—Force I/O Pull Down

The FIOPD indicates whether the address and data external pins are driven by an internal pull-down device at sleep and deep sleep low power modes. When this bit is set and the chip is either in sleep or in deep sleep low power mode, the A[0:31] and D[0:31] external pins are driven to zero. When this bit is set and the chip is not in sleep or deep sleep low power modes (or when FIOPD is cleared) there is no influence on the A[0:31] and D[0:31] external pins.

## 5.10 BASIC POWER STRUCTURE

The MPC821 provides a wide range of possibilities for power supply connection. Figure 5-9 illustrates the different power supply sources for each one of the basic units on the chip. For details on the relationships between the power supplies refer to **Section 5.2 Clock Unit Description**.

The I/O buffers, logic, and clock block are fed by a 3.3 V power supply that allows them to function in a TTL-compatible range of voltages. The internal logic can be fed by a 3.3 or 2 V source allowing a considerable reduction in the power consumption of the part. The PLL is fed by a 3.3 V power supply (VDDSYN) to achieve a high stability in it’s output frequency. The OSCM, TB, DEC, PIT, and RTC are all fed by the KAPWR rail allowing to the external power supply unit to disconnect all other subunits in the part at low power deep sleep mode. The TEXP pin (fed by the same rail) can be used by the external power supply unit to switch between sources.

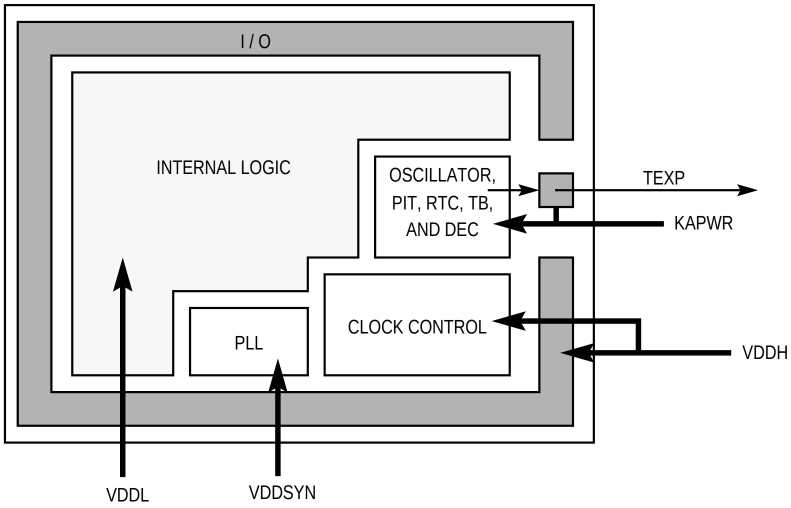


Figure 5-9. MPC821 Basic Power Supply Configuration

## 5.11 KEEP ALIVE POWER

### 5.11.1 Keep Alive Power Configuration

An example of a switching scheme for an optimized low power system is illustrated in Figure 5-10.

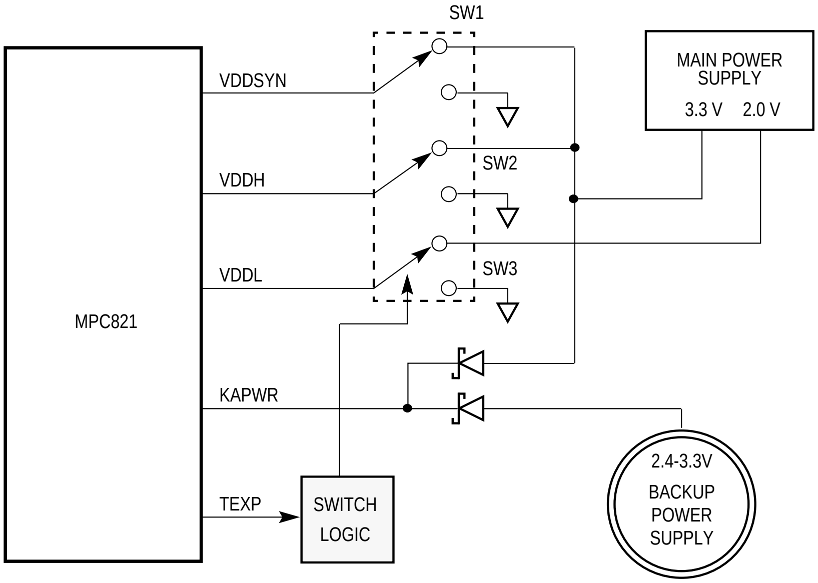


Figure 5-10. External Power Supply Scheme (2.0 V Internal Voltage)

SW1 and SW2 can be unified in one switch if VDDSYN and VDDH are supplied by the same source. If VDDL is fed with 3.3 V, SW2 and SW3 can be unified in one switch. The TEXP signal, if enabled, is asserted by the MPC821 when the RTC or TB time value matches the value programmed in it's associated alarm register or when the PIT or DEC decrements their value to zero. The TEXP signal is negated when writing to the TEXPS status bit with the corresponding data bit being 1. The KAPWR power supply feeds the OSCM. The condition for the main crystal oscillator stability is that the power supply value changes slowly. The maximum slope of the KAPWR power supply should be less than 1.7 V/mS for 32 KHz input frequency (an exponential model of voltage change on the KAPWR rail should ensure $\tau > 20/freq_{oscm}$).

### 5.11.2 Keep Alive Power Registers Lock Mechanism

All the registers defined in the system integration timers and the clocks and reset sections of Table 8-1, as well as the decrementer, and timebase timers, are powered by the KAPWR supply. When disconnecting the main power supply after entering the power-down mode, the value stored in any of these registers is preserved. For cases where this requirement cannot be met, there is a chance of data loss in these registers. To reduce the probability of data loss to a minimum, the MPC821 includes a key mechanism that ensures data retention as long as a register is locked. While a register is locked, writes to this register are ignored.

Each of the registers in the KAPWR region have a key that can be in open or locked state. At power-on reset, all keys are in the open state (except for the real-time clock related registers). Each key has an address associated with it in the internal memory map. A write of 0x55CCAA33 to this location changes the key to the open state. A write of any other data to this location changes the key to the locked state.

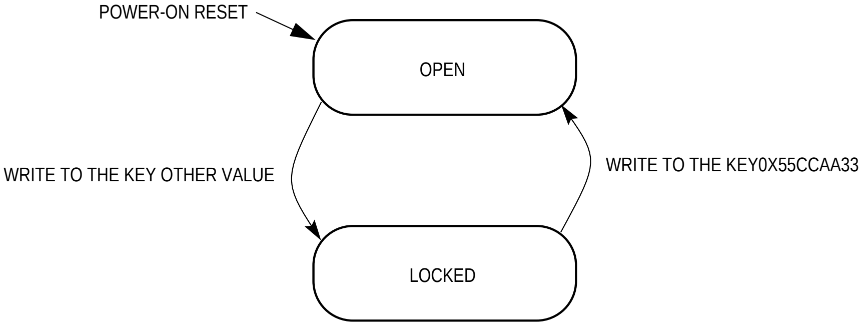


**Figure 5-11. Keep Alive Register Key State Diagram**

The key registers for those registers appearing in the system integration timers and the clocks and reset sections of Table 8-1 are defined in the follow-up sections of the same table.

# SECTION 6
# CORE

## 6.1 OVERVIEW

The core is a module within the MPC821 that is the implementation of the PowerPC™ architecture within the chip. As such, it has the functionality of the PowerPC branch processor and fixed-point processor and includes the implementation of all the PowerPC user mode (problem mode) instructions, except floating-point instructions and relevant privileged instructions as well as the registers associated with the processors and instructions. In addition, it contains part of the development support features of the MPC821, including the breakpoint and watchpoints support, program flow tracking data generation, and debug mode operation in which the chip is controlled by the development support system through the debug port module.

This section describes the functional specifications of the core. It is part of a set of documents describing the MPC821. This manual is based on the PowerPC architecture as defined in the *PowerPC Architecture™ Books (Books I, II, and III).*

**NOTE**

In general, this manual only refers to those sources and does not duplicate any information already specified in those documents.

## 6.2 FEATURES

The following list provides the key MPC821 core features:

- 32-bit PowerPC architecture
- Single-issue machine
- Variable pipeline depth architecture tailored to instruction complexity
- Out of order execution termination
- Branch prediction for prefetch
- 32 × 32 bits general-purpose register file
- Precise exception model
- Extensive debug/testing support

### 6.2.1 General Core Structure

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

- **Sequencer Unit**—Includes the branch processor (next address generation), the instruction prefetch queue, and the interrupt handling mechanism. It controls some data structures within the register unit.
- **Register Unit**—Includes all the user-visible registers, the register's scoreboard mechanism, and a history of previous operations to allow for a precise interrupt model. This module is physically split so that each data structure is implemented near the area where it is used.
- **Fixed-Point Unit**—Includes implementation of all fixed-point instructions except load/store instructions. This module is subdivided into the IMUL/IDIV which includes the implementation of the fixed-point multiply and divide instructions, and the ALU/BFU which includes implementation of all fixed-point logic, add, and subtract instructions, as well as the bit field instructions.
- **Load/Store Unit**—Includes implementation of all load and store instructions, except floating-point processor load and store instructions.

### 6.2.2 Instruction Flow Within the Core

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

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

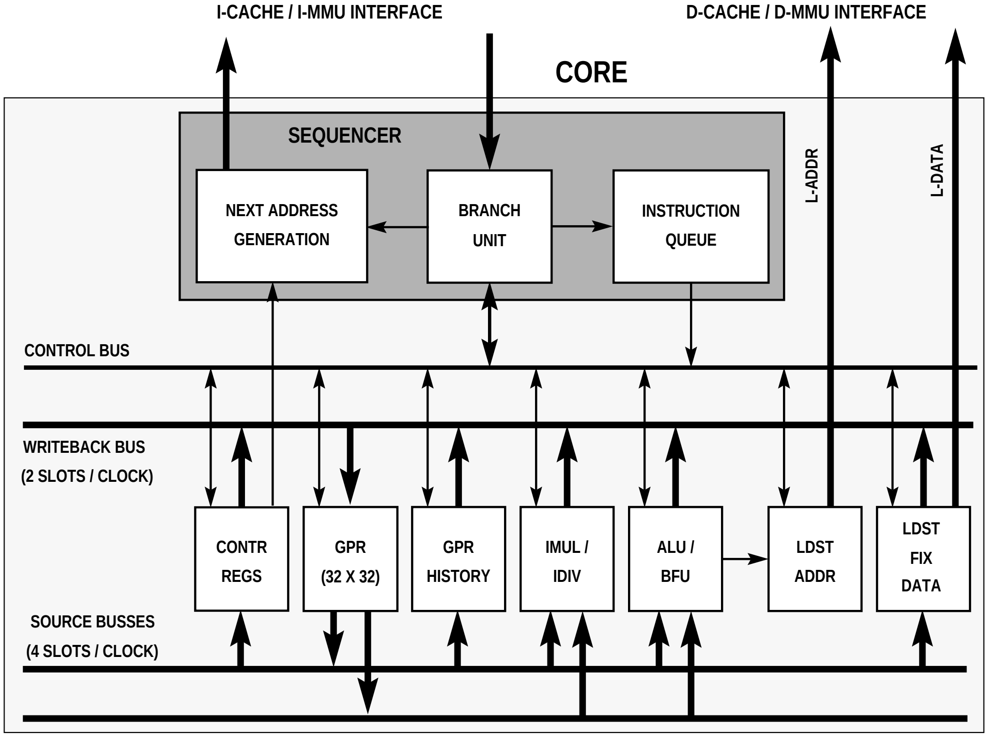


Figure 6-1. Core Block Diagram

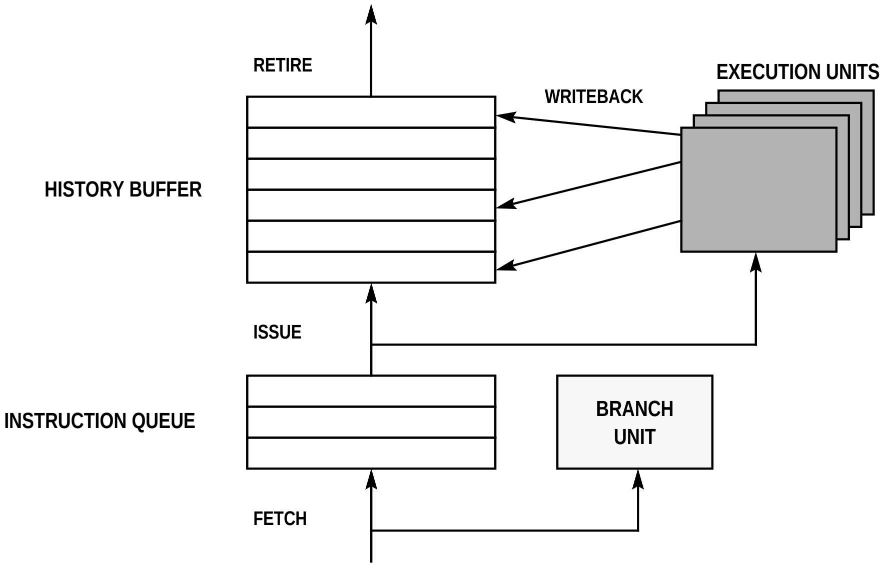


Figure 6-2. Instruction Flow Conceptual Diagram

### 6.2.3 Basic Instruction Pipeline

Figure 6-3 illustrates the basic instruction pipeline timing.

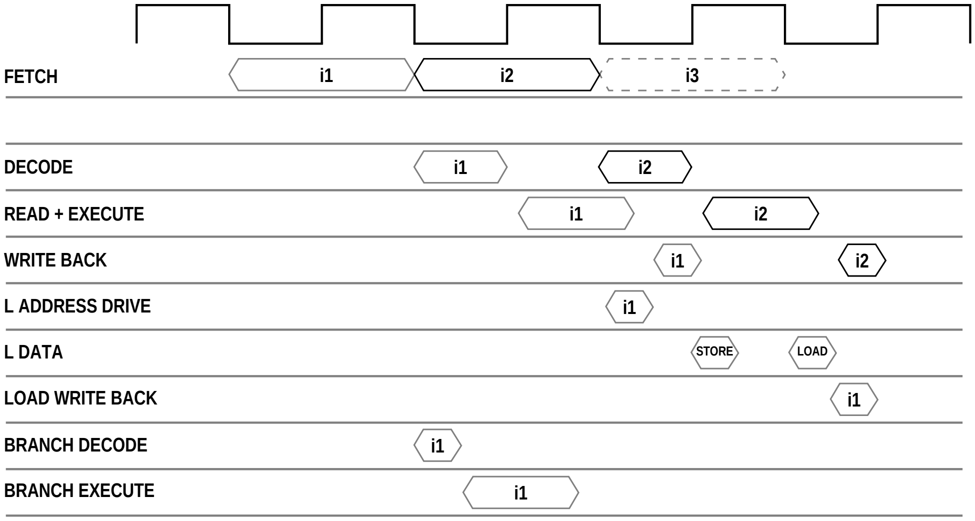


**Figure 6-3. Basic Instruction Pipeline Timing Diagram**

## 6.3 SEQUENCER UNIT

This section specifies the implementation details of the core’s sequencer. Features not addressed herein can be assumed to be the same as those defined for the PowerPC architecture as described in the *PowerPC Architecture™ Books (Books I, II, and III)*.

### 6.3.1 Overview

The instruction sequencer is the heart of the core. It provides centralized control over data flow among execution units and register files. The sequencer implements the basic instruction pipeline, fetches instructions from the memory system and issues them to available execution units, and maintains a state history so it can back up the machine in the event of an exception. Figure 6-4 illustrates the sequencer data path. In addition, the sequencer implements all branch processor instructions (refer to the *PowerPC Architecture™ Books (Books I, II, and III* for more information)) which include flow control and condition register instructions. For latency and blockage data, see Table 4-1 of this manual.

### 6.3.2 Flow Control

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

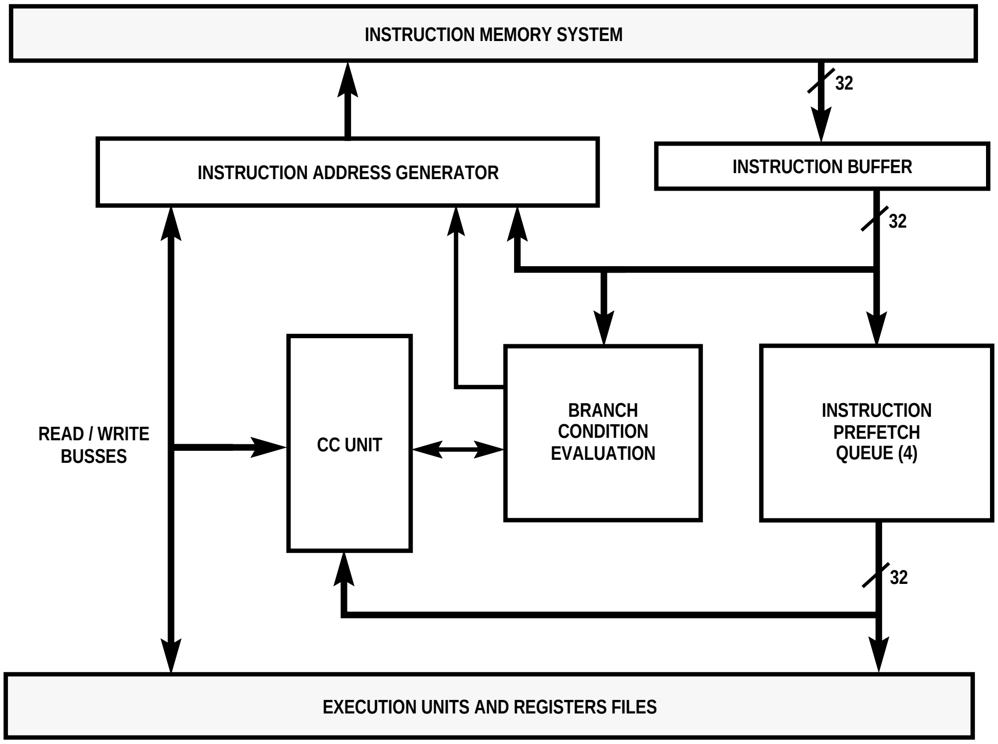

INSTRUCTION MEMORY SYSTEM
32
INSTRUCTION ADDRESS GENERATOR
INSTRUCTION BUFFER
32
READ / WRITE BUSSES
CC UNIT
BRANCH CONDITION EVALUATION
INSTRUCTION PREFETCH QUEUE (4)
32
EXECUTION UNITS AND REGISTERS FILES

**Figure 6-4. Sequencer Data Path**

The sequencer maintains a four instructions deep instruction prefetch queue in an attempt to execute branches in parallel with execution of sequential instructions. In the ideal case, a sequential instruction is issued every clock, even when branches are present in the code. This feature is referred to as branch folding. The instruction prefetch queue also eliminates stalls due to long latency instruction fetches. All instructions are fetched into the instruction prefetch queue, but only sequential instructions are issued to the execution units on reaching the head of the queue. The reason branches enter the queue is for watchpoint marking (refer to **Section 5 Development Support** for details). Since branches do not prevent issue of sequential instructions (unless they come in pairs) the performance impact of entering branches in the instruction prefetch queue is negligible.

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

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

**Table 6-1. Branch Prediction Policy**

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

### 6.3.3 Instruction Issue

The sequencer attempts to issue a sequential instruction on each clock, whenever possible. For an instruction to issue, the execution unit must be available and be able to determine that the required source data is available and no other instruction still in execution targets the same destination register. The sequencer broadcasts to the execution units the existence of the instruction on the instruction bus. The execution units then decode the instruction, interrogate the register unit (if the operands and targets are free), and inform the sequencer that it accepts the instruction for execution.

### 6.3.4 Interrupts

The core interrupts can be generated when an exception occurs. An exception can result from the execution of an instruction or occurrence of some asynchronous external event. There are five exception sources in the MPC821:

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

The handling of interrupts is transparent to user mode code. The core uses the same mechanism to handle all types of exceptions. When a user mode instruction experiences an exception, the machine is placed into privileged state and control is transferred to a software exception handler routine located at some offset within a memory-based vector table. Each interrupt generated in the machine transfers control to a different address in the vector table. For more information on initializing the base address of the vector table, refer to Table 6-13 as well as the PowerPC definition of the MSR. When the exception has been dealt with, the handler can resume execution of the user program without the knowledge that such an event ever occurred. As specified in the *PowerPC™ Operating Environment Architecture (Book III)*, the core implements a precise interrupt model (actually, Book III allows some impreciseness on imprecise mode of FP errors). This means that when an interrupt occurs:

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

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

**Table 6-2. Before and After Interrupts**

| INTERRUPT TYPE | INSTRUCTION TYPE | BEFORE / AFTER | CONTENTS OF SRR0 |
|---|---|---|---|
| Hard Reset | Any | NA | Undefined |
| System Reset | Any | Before | Next Instruction to Execute |
| Machine Check Interrupt | Any | Before | Faulting Instruction |
| Implementation Specific Instruction / Data TLB Miss / Error Interrupts | Any | Before | Faulting Fetch or Load/Store |
| Other Asynchronous Interrupts (Noninstruction Related Interrupts) | Any | Before | Next Instruction to Execute |
| Alignment Interrupt | Load / Store | Before | Faulting Instruction |
| Privileged Instruction | Any Privileged Instruction | Before | Faulting Instruction |
| Trap | tw, twi | Before | Faulting Instruction |
| System Call Interrupt | sc | After | Next Instruction to Execute |

Table 6-2. Before and After Interrupts (Continued)

| INTERRUPT TYPE | INSTRUCTION TYPE | BEFORE / AFTER | CONTENTS OF SRR0 |
|---|---|---|---|
| Trace | Any | After | Next Instruction to Execute |
| Debug I- Breakpoint | Any | Before | Faulting Instruction |
| Debug L- Breakpoint | Load / Store | After | Faulting Instruction + 4 |
| Implementation Dependent<br>Software Emulation Interrupt | NA | Before | Faulting Instruction |
| Floating-Point Unavailable | Floating-Point | Before | Faulting Instruction |

### 6.3.5 Precise Exception Model Implementation

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

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

An exception can be detected at any time during instruction execution and is recorded in the history buffer when the instruction finishes execution. The exception is not recognized until the faulting instruction reaches the head of the history queue. When the exception is recognized, an interrupt process begins. The queue is reversed and machine is restored to it’s state at the time the instruction is issued. Machine state is restored at a maximum rate of two floating-point and two fixed point instructions per clock.

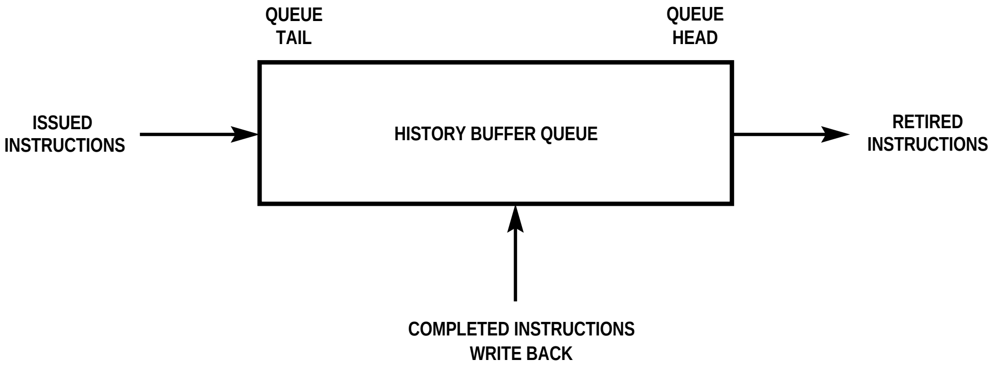


**Figure 6-5. History Buffer Queue**

To correctly restore the architectural state, the history buffer must record the value of the destination prior to instruction execution. 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 completed execution without exception or the store has reached the head of the history buffer.

The history buffer has enough storage to hold six instructions worth of state, but no more than four fixed-point instructions. The other two instructions can be condition code or branch instructions. In the event of a long latency instruction, it is possible (if a data dependency does not occur first) that issued instructions fill up the history buffer. If so, instruction issue halts until the long latency operation (blocking retirement) finishes. The following kinds of instructions can potentially cause the history buffer to fill up:

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

**6.3.5.1 RESTARTABILITY AFTER AN INTERRUPT.** Most of the interrupt cases in the core are always restartable. Some interrupts may be nonrestartable since they can be recognized when the machine status save/restore 0 and 1 registers (SRR0 and SRR1) are busy. Such interrupts in the PowerPC architecture are:

- System reset
- Machine check interrupt

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

Also external interrupts are masked in these areas. In the core, two implementation specific interrupt types can have this phenomena—debug port unmaskable interrupt and breakpoint interrupt (when in nonmaskable mode). Since there might be a case where it is preferable to be restartable, such as in the mentioned implementation specific interrupts, a mechanism is defined to notify the interrupt handler code whether it is in restartable state.

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

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

**Table 6-3. Special Ports to MSR Bits**

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

### 6.3.6 Interrupt Timing

The following table provides the significant events in interrupt processing.

**Table 6-4. Interrupt Latency**

| TIME POINT | FETCH | ISSUE | INSTRUCTION COMPLETE | KILL PIPELINE |
|---|---|---|---|---|
| A | | Faulting Instruction Issue | | |
| B | | | Instruction Complete and All Previous Instructions Complete | |
| C | Start Fetch Handler | | | Kill Pipeline |
| D ≤ B + 3 Clocks | | | | |
| E | | 1st Instruction of Handler Issued | | |

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

At time A the excepting instruction issues and begins execution. During the interval A-B, previously issued instructions are finishing execution. The interval A-B is equivalent to the time required for all instructions currently in progress to complete (the time to serialize the machine as described below). At time point B, the exception is recognized and during the interval B-D the machine state is being restored. This time is up to 10 cycles (zero word state memory). At time point C, the core starts fetching the first instruction of the exception handler if the interrupt handler is external and 5 cycles if it is in I-cache and NO SHOW mode is on.

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

## 6.3.7 Serialization

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

- **Execution serialization**

  Instruction issue is halted until all instructions currently in progress have completed execution (all internal pipeline stages and instruction buffers have emptied and all outstanding memory transactions are completed.)

- **Fetch serialization**

  Instruction fetch is halted until all instructions currently in the processor have completed execution (all issued prefetched instructions waiting to be issued.) The machine after fetch serialization is said to be completely synchronized.

An attempt to issue a serializing instruction causes the machine to serialize before the instruction issues. For details, see Table 8-1. Only the sync (synchronize) instruction guarantees serialization across PowerPC implementations. Fetching an **i**sync (storage control) instruction causes fetch serialization. Also, when the serialize mode bit ($CTRL_{SER}$) is asserted, or when in debug mode, any instruction can cause fetch serialization.

**6.3.7.1 SERIALIZATION LATENCY.** The time required to serialize the machine is the amount of time needed to complete the instructions currently in progress. This time is heavily dependent on the instructions in progress and the memory system latency. It is impossible to put an absolute upper bound on this time because the memory system design is not under CPU control. The time to complete the current instruction is generally the machine serialization time and the specific instruction execution time determines how long serialization takes. This can be either divide, load, or store a multiple, string, or pair of simple load/store instructions. See Table 8-1 for more information.

## 6.3.8 External Interrupt

The core provides one external interrupt line: the architectural maskable external interrupt. In the MPC821,this interrupt is generated by the on-chip interrupt controller. It is software acknowledged and maskable by the $MSR_{EE}$ bit. $MSR_{EE}$ is automatically cleared by the hardware to disable external interrupts when any interrupt is taken.

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

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

Any instruction that does not meet these criteria is discarded (with all of it's side effects) and the execution at the end of the interrupt handler resumes with the first instruction that was discarded. If all the instructions in the history buffer were allowed to complete, execution at the end of the interrupt handler resumes with the next instruction. External interrupt latency depends on the time required to reference memory. The measurement is equal to one of three things, in addition to interval E-B in Table 6-4.

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

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

### 6.3.9 Interrupt Ordering

There are two major types of interrupts:

- Instruction-related interrupts
- Asynchronous (noninstruction-related) interrupts

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

**Table 6-5. Instruction-Related Interrupt Detection Order**

| NUMBER | INTERRUPT TYPE | CAUSED BY |
|---|---|---|
| 1 | Trace | Trace Bit Asserted[1] |
| 2 | Implementation Dependent Instruction TLB Miss | Instruction MMU TLB Miss |
| 3 | Implementation Dependent Instruction TLB Error | Instruction MMU Protection / Translation Error |
| 4 | Machine Check Interrupt | Fetch Error |
| 5 | Debug I- Breakpoint | Match Detection |
| 6 | Implementation Dependent Software Emulation Interrupt | Attempt to Invoke Unimplemented Feature |
| 1 | Floating-Point Unavailable | Attempt to is Made to Execute Floating-Point Instruction and $MSR_{FP}$ =0 |

Table 6-5. Instruction-Related Interrupt Detection Order (Continued)

| NUMBER | INTERRUPT TYPE | CAUSED BY |
|---|---|---|
| 7[2] | Privileged Instruction | Attempt to Execute Privileged Instruction in Problem Mode |
| | Alignment Interrupt | Load/Store Checking (refer to the **Alignment Interrupt** section) |
| | System Call Interrupt | SC Instruction |
| | Trap | Trap Instruction |
| 8 | Implementation Dependent Data TLB Miss | Data MMU TLB Miss |
| 9 | Implementation Dependent Data TLB Error | Data MMU TLB Protection / Translation Error |
| 10 | Machine Check Interrupt | Load or Store Access Error |
| 11 | Debug L- Breakpoint | Match Detection |

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

2. All the cases listed on item 7 are exclusive for any one instruction.

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

Table 6-6. Interrupt Priorities Mapping

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

## 6.4 REGISTER UNIT

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

- Decoding of the operand fields of all sequential instructions
- Drives the operand busses, as requested by the execution unit
- Performs scoreboard checking and signing
- Samples the result data from the write-back bus

### 6.4.1 Control Registers

The following tables provide the CPU control registers implemented within the MPC821.

**Table 6-7. Standard Special Purpose Registers**

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

## Table 6-8. Standard Timebase Register Mapping

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

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

## Table 6-9. Added Special Purpose Registers

| SPR | | | REGISTER NAME | PRIVILEGED | SERIALIZE ACCESS |
|---|---|---|---|---|---|
| DECIMAL | SPR $_{5:9}$ | SPR $_{0:4}$ | | | |
| 80 | 00010 | 10000 | EIE[1] | Yes | Write |
| 81 | 00010 | 10001 | EID | Yes | Write |
| 82 | 00010 | 10010 | NRI | Yes | Write |
| 144 | 00100 | 10000 | CMPA[1] | Debug[3] | Fetch Sync on Write |
| 145 | 00100 | 10001 | CMPB | Debug | Fetch Sync on Write |
| 146 | 00100 | 10010 | CMPC | Debug | Fetch Sync on Write |
| 147 | 00100 | 10011 | CMPD | Debug | Fetch Sync on Write |
| 148 | 00100 | 10100 | ICR | Debug | Fetch Sync on Write |
| 149 | 00100 | 10101 | DER | Debug | Fetch Sync on Write |
| 150 | 00100 | 10110 | COUNTA | Debug | Fetch Sync on Write |
| 151 | 00100 | 10111 | COUNTB | Debug | Fetch Sync on Write |
| 152 | 00100 | 11000 | CMPE | Debug | Write: Fetch Sync<br>Read: Synch Relative to Load/Store Operations |
| 153 | 00100 | 11001 | CMPF | Debug | Write: Fetch Sync<br>Read: Synch Relative to Load/Store Operations |
| 154 | 00100 | 11010 | CMPG | Debug | Write: Fetch Sync<br>Read: Synch Relative to Load/Store Operations |

### Table 6-9. Added Special Purpose Registers (Continued)

| SPR | | | REGISTER NAME | PRIVILEGED | SERIALIZE ACCESS |
|---|---|---|---|---|---|
| DECIMAL | SPR $_{5:9}$ | SPR $_{0:4}$ | | | |
| 155 | 00100 | 11011 | CMPH | Debug | Write: Fetch Sync<br>Read: Synch Relative to Load/Store Operations |
| 156 | 00100 | 11100 | LCTRL1 | Debug | Write: Fetch Sync<br>Read: Synch Relative to Load/Store Operations |
| 157 | 00100 | 11101 | LCTRL2 | Debug | Write: Fetch Sync<br>Read: Synch Relative to Load/Store Operations |
| 158 | 00100 | 11110 | ICTRL | Debug | Fetch Sync on Write |
| 159 | 00100 | 11111 | BAR | Debug | Write: Fetch Sync<br>Read: Synch Relative to Load/Store Operations |
| 630 | 10011 | 10110 | DPDR | Debug | Read and Write |
| 631 | 10011 | 10111 | DPIR[4] | Yes | Fetch |
| 638 | 10011 | 11110 | IMMR | Yes | Write - As a Store |
| 560 | 10001 | 10000 | IC_CST | Yes | Write - As a Store |
| 561 | 10001 | 10001 | IC_ADR | Yes | Write - As a Store |
| 562 | 10001 | 10010 | IC_DAT | Yes | Write - As a Store |
| 568 | 10001 | 11000 | DC_CST | Yes | Write - As a Store |
| 569 | 10001 | 11001 | DC_ADR | Yes | Write - As a Store |
| 570 | 10001 | 11010 | DC_DAT | Yes | Write - As a Store |
| 784 | 11000 | 10000 | MI_CTR | Yes | Write - As a Store |
| 786 | 11000 | 10010 | MI_AP | Yes | Write - As a Store |
| 787 | 11000 | 10011 | MI_EPN | Yes | Write - As a Store |
| 789 | 11000 | 10101 | MI_TWC<br>(MI_L1DL2P) | Yes | Write - As a Store |
| 790 | 11000 | 10110 | MI_RPN | Yes | Write - As a Store |
| 816 | 11001 | 10000 | MI_DBCAM | Yes | Write - As a Store |
| 817 | 11001 | 10001 | MI_DBRAM0 | Yes | Write - As a Store |
| 818 | 11001 | 10010 | MI_DBRAM1 | Yes | Write - As a Store |
| 792 | 11000 | 11000 | MD_CTR | Yes | Write - As a Store |
| 793 | 11000 | 11001 | M_CASID | Yes | Write - As a Store |

**Table 6-9. Added Special Purpose Registers (Continued)**

| SPR | | | REGISTER NAME | PRIVILEGED | SERIALIZE ACCESS |
|---|---|---|---|---|---|
| DECIMAL | SPR $_{5:9}$ | SPR $_{0:4}$ | | | |
| 794 | 11000 | 11010 | MD_AP | Yes | Write - As a Store |
| 795 | 11000 | 11011 | MD_EPN | Yes | Write - As a Store |
| 796 | 11000 | 11100 | M_TWB<br>(MD_L1P) | Yes | Write - As a Store |
| 797 | 11000 | 11101 | MD_TWC<br>(MD_L1DL2P) | Yes | Write - As a Store |
| 798 | 11000 | 11110 | MD_RPN | Yes | Write - As a Store |
| 799 | 11000 | 11111 | M_TW<br>(M_SAVE) | Yes | Write - As a Store |
| 824 | 11001 | 11000 | MD_DBCAM | Yes | Write - As a Store |
| 825 | 11001 | 11001 | MD_DBRAM0 | Yes | Write - As a Store |
| 826 | 11001 | 11010 | MD_DBRAM1 | Yes | Write - As a Store |

NOTES:
1. Refer to **Section 6.3.5.1 Restartability After An Interrupt**.
2. Refer to **Section 19.5.1 Development Support Registers List**.
3. Protection of Registers designated with “debug” privilege is described in **Section 19.5.2 Development Support Registers Protection**.
4. This register is a fetch-only register, using mtspr is ignored and using mfspr gives an undefined value.

**Table 6-10. Other Control Registers**

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

**6.4.1.1 PHYSICAL LOCATION OF SPECIAL REGISTERS.** Some of the special registers in the CPU are physically located outside of the core. Access to these registers is gained as it is to any other special register, via the appropriate mtspr, mfspr instructions through the internal chip busses. Apart from the PowerPC timebase counter and the decrementer (which are implemented within the SIU), in the current implementation the following encoding is reserved for special registers not located within the core (specific registers encoding are given in Table 6-9).

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

| SPR | | RESERVED FOR |
|---|---|---|
| $\text{SPR}_{5:9}$ | $\text{SPR}_{0:4}$ | |
| 100xx<br>110xx | xxxxx | External to the Core Registers |
| 1x0xx | x0xxx | Reserved |
| 10011 | x0xxx | SIU Internal Registers |
| 0xxxx | xxxxx | If Appears on the Internal Bus Signifies DEC or TB |
| 10000 | x0xxx | Reserved |
| 10000 | x1xxx | Reserved |
| 1100x | x0xxx | I-MMU Implementation Specific Control |
| 1100x | x1xxx | D-MMU Implementation Specific Control |
| 10001 | x00xx | I-Cache Registers |
| 10001 | x10xx | D-Cache Registers |

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

### Table 6-12. Address of Special Registers Located Outside the Core

| 0:17 | 18:22 | 23:27 | 28:31 |
|---|---|---|---|
| 0. .0 | $\text{spr}_{0:4}$ | $\text{spr}_{5:9}$ | 0000 |

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

### 6.4.1.2 CONTROL REGISTERS BIT ASSIGNMENT.

**6.4.1.2.1 Machine State Register Bits.** The following table provides the bit assignments for the MSR.

**Table 6-13. Machine State Register Bit Assignments**

| BIT | MSR NAME | SRR1 FUNCTION | MSR BIT DESCRIPTION |
|---|---|---|---|
| 0 | Reserved | ➦ | |
| 1-4 | Reserved | Instruction Storage Interrupt Status | |
| 5-9 | Reserved | ➦ | |
| 10 | Reserved | Instruction Storage Interrupt Status | |
| 11:12 | Reserved | Program Interrupt Status | |
| 13 | POW | | Power Management Enable |
| 14 | ISF | | Implementation Specific Function |
| 15 | ILE | | Interrupt Little Endian Mode |
| 16 | EE | ➦ | External Interrupt Enable |
| 17 | PR | ➦ | Problem State |
| 18 | FP | ➦ | Floating-Point Available |
| 19 | ME | ➦ | Machine Check Enable |
| 20 | FE0 | ➦ | Floating-Point Exception Mode 0<br>(Has No Effect) |
| 21 | SE | ➦ | Single Step Trace Enable |
| 22 | BE | ➦ | Branch Trace Enable |
| 23 | FE1 | ➦ | Floating-Point Exception Mode 1<br>(Has No Effect) |
| 24 | Reserved | ➦ | |
| 25 | IP | ➦ | Interrupt Prefix<br>0->0x000n_nnnn<br>1->0xFFFn_nnnn |
| 26 | IR | ➦ | Instruction Relocate |
| 27 | DR | ➦ | Data Relocate |

Table 6-13. Machine State Register Bit Assignments (Continued)

| BIT | MSR NAME | SRR1 FUNCTION | MSR BIT DESCRIPTION |
|---|---|---|---|
| 28,29 | Reserved | ➦ | Reserved |
| 30 | RI | ➦ | Recoverable Interrupt |
| 31 | LE (➥) | ➦ | Little Endian Mode |

NOTES: 1. ➦ Means this bit is loaded from the corresponding bit in MSR when an interrupt is taken. The appropriate bit in MSR is loaded from this bit when an RFI is executed.

2. ➥ Means this bit is loaded from the ILE bit (bit 15) when an interrupt is taken.

3. Reserved bits in the MSR are set from the source value on write and return the value last set for it on read. Note 1 applies for reserved bits as well.

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

**6.4.1.2.3 Fixed-Point Exception Cause Register.** The following table provides the bit assignments for the fixed-point exception cause register (XER).

Table 6-14. Fixed-Point Exception Cause Register

| BIT | NAME | DESCRIPTION |
|---|---|---|
| 0 | SO | Summary Overflow |
| 1 | OV | Overflow |
| 2 | CA | Carry |
| 3:24 | Reserved | |
| 25:31 | BCNT | Byte Count For Load/Store String Operations |

### 6.4.1.3 INITIALIZATION OF CONTROL REGISTERS.

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

**6.4.1.3.2 Hard/Soft Reset.** When a hard or soft reset occurs, the registers affected by system reset are set similarly. The following list provides the differences between how each register is set:

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

## 6.5 FIXED-POINT UNIT

The fixed-point unit (FXU) implements all fixed-point processor instructions (refer to the *PowerPC™ User Instruction Set Architecture (Book 1)*), except the fixed-point storage access instructions, which are implemented by the load/store unit.

### 6.5.1 XER Update In Divide Instructions

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

## 6.6 LOAD/STORE UNIT

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

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

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

- Supports many instructions
  - Fixed-point load/store instructions
  - Storage synchronization instructions
  - Storage control instructions
  - Off-core special registers move instructions
  - Big-endian mode and little-endian mode
- A two entry load/store instruction address queue
- Pipelined operation
- Minimal load latency–2 clocks (using 1 clock on-chip data RAM)
- Minimal store latency–1 clock (the load/store unit ends the store execution in 2 clocks, using 1 clock on-chip data RAM)
- Load/store multiple and string instructions synchronize
- Support of load/store breakpoint/watchpoint detection (address and data)

Figure 6-6 illustrates a conceptual block diagram of the load/store unit that contains two queues:

- Address queue—A 2-entry queue shared by all load/store instructions.
- Fixed-point data queue—A 2-entry, 32-bit wide queue that holds fixed-point data.

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

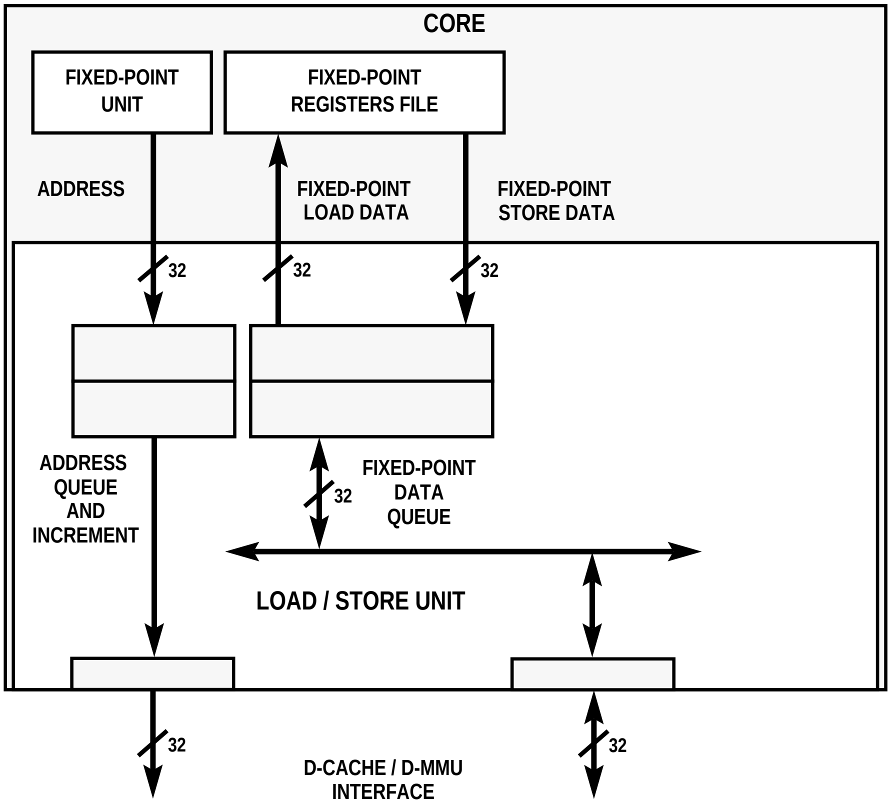


**Figure 6-6. Load/Store Unit Functional Block Diagram**

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

### 6.6.1 Load/Store Instruction Issue

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

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

If all operands are available, the load/store unit takes the instruction and enables the sequencer to issue a new instruction. Then, using a dedicated interface, the load/store unit notifies the integer unit of the need to calculate the effective address. All load/store instructions are executed and terminated in order. If there are no prior instructions waiting in the address queue, the load/store instruction is issued to the data cache immediately at the time the instruction is taken. Otherwise, if there are prior instructions remaining whose addresses have not yet been issued to the data cache, the instruction is inserted into the address queue and data is inserted into the respective store data queue. For load/store with update instructions, the destination address register is written back on the following clock regardless of the state of the address queue.

### 6.6.2 Load/Store Synchronizing Instructions

For the following load/store instructions, they are not taken until all previous instructions have terminated.

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

For the following load/store instructions, issuance of further instructions is stalled until they terminate:

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

### 6.6.3 Instructions Issued to the Data Cache

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

### 6.6.4 Store Instructions Cycles Issue

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

### 6.6.5 Nonspeculative Load Instructions

Load instructions targeted at a nonspeculative memory region are identified as nonspeculative one clock cycle after the previous load/store bus cycle termination, but only if all prior instructions have terminated normally and without an exception.

The nonspeculative identification relates to the state of the cycle's associated instruction. In case of lmw, although the cycles are pipelined into the bus they are all marked as nonspeculative as the instruction is nonspeculative. In case of a single register load instruction for which more than one bus cycle is generated (unaligned load instructions), some of the cycles can be marked as speculative while later cycles can be marked as nonspeculative after all prior instructions terminate. When executing speculative load cycles to nonspeculative external memory region, no external cycles are generated until the load instruction becomes nonspeculative.

### 6.6.6 Unaligned Instructions Execution

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

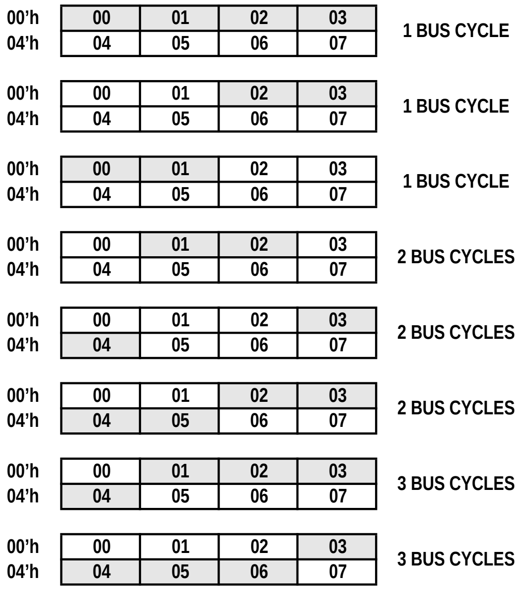


**Figure 6-7. Number of Bus Cycles Needed For Unaligned, Single Register Fixed-Point Load/Store Instructions**

### 6.6.7 Little Endian Support

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

### 6.6.8 Atomic Update Primitives

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

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

The MPC821 storage reservation supplies hooks for the support of storage reservation implementation in a hierarchical bus structure. Refer to **Section 7.5.9 Storage Reservation** for a full description of the storage reservation mechanism. In case of storage reservation on internal memory, a lwarx indication causes the on-chip snoop logic to latch the address. This logic notifies the load/store unit in the case of an internal master store access, then the reservation is reset. If a new lwarx instruction address phase is successfully executed it replaces any previous storage reservation address at the appropriate snoop logic. However, when an stwcx instruction is executed, the storage reservation is canceled, unless an alignment interrupt condition is detected.

### 6.6.9 Instruction Timing

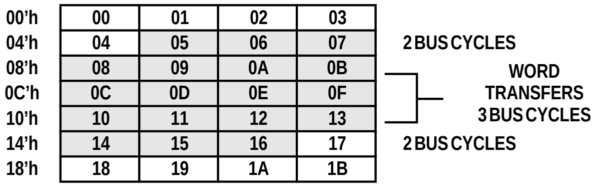
The following table summarizes the different load/store instructions timing in the case of a zero wait state memory references on a parked bus. In case of external memory accesses, pipelined external accesses are assumed.

**Table 6-15. Load/Store Instructions Timing**

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

NOTE: N denotes the number of registers transferred.

String instructions are broken into a series of aligned bus accesses. Figure 6-8 illustrates the maximum number of bus cycles needed for string instruction execution (where the beginning and end of the string is unaligned.)



| | | | | |
|---|---|---|---|---|
| 00'h | 00 | 01 | 02 | 03 |
| 04'h | 04 | 05 | 06 | 07 |
| 08'h | 08 | 09 | 0A | 0B |
| 0C'h | 0C | 0D | 0E | 0F |
| 10'h | 10 | 11 | 12 | 13 |
| 14'h | 14 | 15 | 16 | 17 |
| 18'h | 18 | 19 | 1A | 1B |

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

## 6.6.10 Storage Control Instructions Stall

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

## 6.6.11 Off-Core Special Registers Access

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

## 6.6.12 Storage Control Instructions

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

## 6.6.13 Exceptions

**6.6.13.1 DAR, DSISR, AND BAR OPERATION.** The load/store unit keeps track of all instructions and bus cycles. In the case of a bus error (an error detected by the MMU) the DAR is loaded with the cycle's effective address. In the case of a multicycle instruction the effective address of the first offending cycle is loaded.

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

**Table 6-16. DAR, BAR and DSISR Values Summary**

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

# SECTION 7
# POWERPC ARCHITECTURE COMPLIANCE

This section describes implementation dependent choices made for the core on issues that are optional by the PowerPC™ architecture as defined in the *PowerPC Architecture™ Books (Books I, II, and III)*. It also describes features that exist in the architecture, but are not supported by the core. The *User Programming Model of the Architecture* as defined in the *PowerPC™ User Instruction Set Architecture (Book I)* is fully supported by the MPC821.

## 7.1 POWERPC USER INSTRUCTION SET ARCHITECTURE (BOOK I)

### 7.1.1 Computation Modes

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

### 7.1.2 Reserved Fields

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

### 7.1.3 Classes of Instructions

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

### 7.1.4 Exceptions

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

### 7.1.5 The Branch Processor

### 7.1.6 Instruction Fetching

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

### 7.1.7 Branch Instructions

The coreimplements all the instructions defined for the branch processor by the *PowerPC™ User Instruction Set Architecture (Book I)* in the hardware. For performance of various instructions, refer to Table 8-1 of this manual.

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

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

### 7.1.8 The Fixed-Point Processor

**7.1.8.1 FIXED-POINT INSTRUCTIONS.** The core implements the following instructions:

- Fixed-point arithmetic instructions
- Fixed-point compare instructions
- Fixed-point trap instructions
- Fixed-point logical instructions
- Fixed-point rotate and shift instructions
- Move to/from system register instructions

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

**7.1.8.1.1 Move To/From System Register Instructions.** Move to/from invalid special registers in which spr0 =1 yields invocation of the privilege instruction error interrupt handler if the processor is in problem state. For a list of all implemented special registers, refer to **Section 6.4.1 Control Registers**.

**7.1.8.1.2 Fixed-Point Arithmetic Instructions.** If an attempt is made to perform any of the divisions in the divw[o][.] instruction:

0x80000000 ÷ -1

<anything> ÷ 0

Then, the contents of RT are 0x80000000 and if Rc =1, the contents of bits in CR field 0 are LT = 1, GT = 0, EQ = 0, and SO is set to the correct value. If an attempt is made to perform any of the divisions in the divw[o][.] instruction, <anything> ÷ 0. Then, the contents of RT are 0x80000000 and if Rc =1, the contents of bits in CR field 0 are LT = 1, GT = 0, EQ = 0, and SO is set to the correct value. In cmpi, cmp, cmpli, and cmpl instructions, the L-bit is applicable for 64-bit implementations. In 32-bit implementations, if L = 1 the instruction form is invalid. The core ignores this bit and therefore, the behavior when L = 1 is identical to the valid form instruction with L = 0.

## 7.1.9 Load/Store Processor

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

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

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

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

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

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

**7.1.9.5 OPTIONAL INSTRUCTIONS.** No optional instructions are supported.

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

## 7.2 POWERPC VIRTUAL ENVIRONMENT ARCHITECTURE (BOOK II)

### 7.2.1 Storage Model

The MPC821 caches are defined as follows:

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

**7.2.1.1 MEMORY COHERENCE.** Hardware memory coherence is not supported in the MPC821hardware and should be performed in software or by defining storage as cache inhibited, if required. In addition, the MPC821 does not provide any data storage attributes to an external system.

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

The MPC821 does not cause the system data storage error handler to be invoked if the storage accessed by the lwarx and stwcx instructions is in the write-through required mode. Also, the MPC821 does not provide support for snooping an external bus activity outside the chip. The provision is made to cancel the reservation inside the MPC821 by using the CR_B and KR_B input pins. Data cache has a snoop logic to monitor the internal bus for CPM accesses of the address associated with the last lwarx instruction.

### 7.2.2 Effect of Operand Placement on Performance

The load/store unit hardware supports all of the PowerPC load/store instructions. An optimal performance is obtained for naturally aligned operands. These accesses result in optimal performance (one bus cycle) for up to 4 bytes size and good performance (two bus cycles) for double precision floating-point operands. 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 *PowerPC™ Virtual Environment Architecture (Book II),* except for the case of 8-byte operands. In case, since the MPC821 uses a 32-bit wide data bus, the performance is good rather than optimal. Refer to **Section 6.6.6 Unaligned Instructions Execution** for a description of the fixed-point unaligned instruction execution and timing and to **Section 6.6.9 Instruction Timing** for a description of the string instruction timing.

### 7.2.3 Storage Control Instructions

The MPC821 interprets the cache control instructions (icbi, isync, dcbt, dcbi, dcbf, dcbz, dcbst, eieio, and dcbtst) as if they pertain only to the MPC821 cache. These instructions do not broadcast. Any bus activity caused by these instructions is a direct result of performing the operation on the MPC821 cache.

**7.2.3.1 INSTRUCTION CACHE BLOCK INVALIDATE (icbi) INSTRUCTION.** The effective address is translated by the MMU (according to the $MSR_{IR}$) and the associative block in the instruction cache is invalidated if hit.

**7.2.3.2 INSTRUCTION SYNCHRONIZE (isync) INSTRUCTION.** The isync instruction causes a reflect which waits for all prior instructions to complete and then executes the next sequential instruction. Any instruction after an isync will see all effects of prior instructions.

**7.2.3.3 DATA CACHE BLOCK TOUCH (dcbt) INSTRUCTION.** The block associated with this instruction is checked for hit in the cache. If its a miss, the instruction is treated as a regular miss, except that the bus error does not cause an interrupt. If no error occurs, the line is written into the cache.

**7.2.3.4 DATA CACHE BLOCK TOUCH FOR STORE (dcbtst) INSTRUCTION.** The block associated with this instruction is checked for a hit in the cache. If its a miss, the instruction is treated as a regular miss, except that bus error does not cause an interrupt. If no error occurs, the line is written into the cache.

**7.2.3.5 DATA CACHE BLOCK SET TO ZERO (dcbz) INSTRUCTION.** This instruction is executed according to the definition in the *PowerPC™ Virtual Environment Architecture (Book II)*.

**7.2.3.6 DATA CACHE BLOCK STORE (dcbst) INSTRUCTION.** This instruction is executed according to the definition in the *PowerPC™ Virtual Environment Architecture (Book II)*.

**7.2.3.7 DATA CACHE BLOCK INVALIDATE (dcbi) INSTRUCTION.** The effective address is translated by the MMU (according to the $MSR_{IR}$ bit) and the associative block in the data cache is invalidated if hit.

**7.2.3.8 DATA CACHE BLOCK FLUSH (dcbf) INSTRUCTION.** This instruction is executed according to the definition in the *PowerPC™ Virtual Environment Architecture (Book II)*.

**7.2.3.9 ENFORCE IN-ORDER EXECUTION OF I/O (eieio) INSTRUCTION.** When executing an eieio instruction, the load/store unit will wait until all previous accesses have terminated before issuing cycles associated with load/store instructions following the eieio instruction.

### 7.2.4 Timebase

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

## 7.3 POWERPC OPERATING ENVIRONMENT ARCHITECTURE (BOOK III)

The MPC821 has an internal memory space that includes memory-mapped control registers and internal 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 master.

### 7.3.1 Branch Processor

#### 7.3.1.1 BRANCH PROCESSOR REGISTERS.

**7.3.1.1.1 Machine State Register.** The floating-point exception mode is ignored by the MPC821.

The IP bit initial state after reset is set as programmed by the reset configuration as specified by the **Section 12 System Interface Unit**.

**7.3.1.1.2 Processor Version Register.** The value of the version field of register PVR is x'0050'. The value of the revision field is x'0000' and it is incremented each time that the software distinguishes between the core revisions.

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

### 7.3.2 Fixed-Point Processor

#### 7.3.2.1 SPECIAL PURPOSE REGISTERS.

**7.3.2.1.1 Unsupported Registers.** The following registers are not supported by the MPC821. Refer to **Section 7.3.3 Storage Model** for more details.

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

**7.3.2.1.2 Added Registers.** For a list of added special purpose registers, refer to Table 6-9.

### 7.3.3 Storage Model

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

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

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

### 7.3.4 Reference and Change Bits

No reference bit is supported by the MPC821. However, the change bit is supported by using the data TLB error interrupt mechanism on a write attempt to an unmodified page.

### 7.3.5 Storage Protection

Two main protection modes are supported by the MPC821:

- Domain manager mode
- PowerPC mode

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

### 7.3.6 Storage Control Instructions

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

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

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

**7.3.6.4 TLB SYNCHRONIZE (tlbsync) INSTRUCTION.** The tlbsync instruction is implemented and functions as a regular mtspr instruction with regard to engine synchronization with no further effects.

## 7.3.7 Interrupts

**7.3.7.1 INTERRUPT CLASSES.** The core implements all storage-associated interrupts as precise interrupts. This means that a load/store instruction is not complete until all possible error indications have been sampled from the load/store bus. This also implies that a store, or a nonspeculative load instruction is not issued to the load/store bus until all previous instructions have completed. In case of a late error, a store cycle (or a nonspeculative load cycle) can be issued and then aborted.

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

**7.3.7.3 INTERRUPT DEFINITIONS.** The following table defines the offset value by interrupt type.

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

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

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

| OFFSET (HEX) | INTERRUPT TYPE |
|---|---|
| 01400 | Implementation Dependent Data TLB Error |
| 01500 - 01BFF | Reserved |
| 01C00 | Implementation Dependent Data Breakpoint |
| 01D00 | Implementation Dependent Instruction Breakpoint |
| 01E00 | Implementation Dependent Peripheral Breakpoint |
| 01F00 | Implementation Dependent Non Maskable Development Port |

**7.3.7.3.1 System Reset Interrupt.** A system reset interrupt occurs when the $\overline{\text{IRQ0}}$ pin is asserted and the following registers are set.

Save/Restore Register 0 (SRR0)

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

Save/Restore Register 1 (SRR1)

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

Machine State Register (MSR)

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

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

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

As defined in the *PowerPC™ Operating Environment Architecture (Book III)*, machine check interrupts are enabled when $\text{MSR}_{\text{ME}}$ =1. If $\text{MSR}_{\text{ME}}$ = 0 and a machine check interrupt indication is received, the processor enters the checkstop state. The behavior of the core in checkstop state is dependent on the working mode as defined in **Section 19.3.1.1 Debug Mode Enable vs. Debug Mode Disable**. When the processor is in debug mode enable, it enters the debug mode instead of the checkstop state. When in debug mode disable, instruction processing is suspended and cannot be restarted without resetting the core.

An indication is sent to the SIU which may generate an automatic reset in this condition. Refer to the **Section 12 System Interface Unit** for more details. If the machine check interrupt is enabled, MSR$_{ME}$ =1, it is taken. If SRR1 Bit 30 =1, the interrupt is recoverable and the following registers are set.

SRR0

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

SRR1

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

MSR

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

For load/store bus cases, these registers are also set:

Data/Storage Interrupt Status Register (DSISR)

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

Data Address Register (DAR)

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

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

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

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

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

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

**7.3.7.3.6 Program Interrupt.** Floating-point enabled exception type program interrupt is not generated by the MPC821. Likewise, illegal instruction type program interrupt is not generated by the core, but an implementation dependent software emulation interrupt is generated instead. Aprivileged instruction program interrupt generated for an on-core valid special purpose register (SPR) field or any SPR encoded as an external special register if $SPR_0$ =1 and $MSR_{PR}$ =1, as well as if an attempt to execute privileged instruction occurred when $MSR_{PR}$ =1. See Table 6-11 for details.

**7.3.7.3.7 Floating-Point Unavailable Interrupt.** The floating-point unavailable interrupt is not generated by the MPC821. An implementation dependent software emulation interrupt will be taken on any attempt to execute floating-point instruction, regardless of $MSR_{FP}$.

**7.3.7.3.8 Trace Interrupt.** A trace interrupt occurs if $MSR_{SE}$ = 1 and any instruction except rfi is successfully completed or $MSR_{BE}$ = 1 and a branch is completed. Notice that the trace interrupt does not occur after an instruction that caused an interrupt (for instance, sc). A monitor/debugger software must change the vectors of other possible interrupt addresses to single-step such instructions. If this is unacceptable, other debug features can be used. Refer to **Section 19 Development Support** for more information. The following registers are set:

SRR0

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

SRR1

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

MSR

| | |
|---|---|
| IP | No change. |
| ME | No change. |
| LE | Bits is copied from ILE. |
| Other | Set to 0. |

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

**7.3.7.3.9 Floating-Point Assist Interrupt.** The floating-point assist interrupt is not generated by the MPC821.An implementation dependent software emulation interrupt will be taken on any attempt to execute floating-point instruction.

**7.3.7.3.10 Implementation Dependent Software Emulation Interrupt.** An implementation dependent software emulation interrupt occurs in the following instances:

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

In addition, the following registers are set:

SRR0

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

SRR1

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

MSR

| | |
|---|---|
| IP | No change. |
| ME | No change. |
| LE | Bits is copied from ILE. |
| Other | Set to 0. |

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

**7.3.7.3.11 Implementation Specific Instruction TLB Miss Interrupt.** The implementation specific instruction TLB miss interrupt occurs when $MSR_{IR}$ =1 and there is an attempt to fetch an instruction from a page whose effective page number cannot be translated by TLB. The following registers are set:

SRR0

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

SRR1

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

MSR

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

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

**7.3.7.3.12 Implementation Specific Instruction TLB Error Interrupt.** The implementation specific instruction TLB error interrupt occurs in the following instances:

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

The following registers are set:

SRR0

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

SRR1

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

MSR

| | |
|---|---|
| IP | No change. |
| ME | No change. |
| LE | Bits is copied from ILE. |
| Other | Set to 0. |

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

**7.3.7.3.13 Implementation Specific Data TLB Miss Interrupt.** The implementation specific data TLB miss interrupt occurs when $MSR_{DR}$ =1 and there is an attempt to access a page whose effective page number cannot be translated by TLB. The following registers are set:

SRR0

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

SRR1

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

MSR

| | |
|---|---|
| IP | No change. |
| ME | No change. |
| LE | Bits is copied from ILE. |
| Other | Set to 0. |

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

**7.3.7.3.14 Implementation Specific Data TLB Error Interrupt.** The implementation specific data TLB error interrupt occurs in the following instances:

- The effective address of a load, store, icbi, dcbz, dcbst, dcbf or dcbi instruction cannot be translated (either segment or page valid bit of this page is cleared in the translation table).
- The access violates the storage protection.
- An attempt to write to a page with a negated change bit.

The following registers are set:

SRR0

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

SRR1

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

MSR

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

DSISR

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

DAR

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

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

**7.3.7.3.15 Implementation Specific Debug Register.** An implementation specific debug interrupt occurs in the following instances:

- When there is an internal breakpoint match (for more details, refer to **Section 19.2 Watchpoints and Breakpoints Support**).
- When a peripheral breakpoint request is presented to the interrupt mechanism.
- When the development port request is presented to the interrupt mechanism. Refer to **Section 19 Development Support** for details on how to generate the development port request.

The following registers are set:

SRR0

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

SRR1

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

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

MSR

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

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

BAR

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

DAR and DSISR

Do not change.

Execution resumes at offset from the base address indicated by $\text{MSR}_{\text{IP}}$ as follows:

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

**7.3.7.4 PARTIALLY EXECUTED INSTRUCTIONS.** In general, the architecture permits instructions to be partially executed when an alignment or data storage interrupt occurs. In the core, instructions are not executed at all if an alignment interrupt condition is detected and data storage interrupt is never generated by the hardware. In the MPC821, the instruction can be partially executed only in the case of the load/store instructions that cause multiple access to the memory subsystem. These instructions are:

- Multiple/string instructions
- Unaligned load/store instructions

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

### 7.3.8 Timer Facilities

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

### 7.3.9 Optional Facilities and Instructions

Any other *PowerPC™ Operating Environment Architecture (Book III)* optional facilities and instructions (except those that are discussed here) are not implemented by the MPC821 hardware. Attempting to execute any of these instructions causes an implementation dependent software emulation interrupt to be taken.

# SECTION 8
# INSTRUCTION EXECUTION TIMING

## 8.1 INSTRUCTIONS TIMING LIST

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

**Table 8-1. Instruction Execution Timing**

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

### Table 8-1. Instruction Execution Timing (Continued)

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

## Table 8-1. Instruction Execution Timing (Continued)

| INSTRUCTIONS | LATENCY | BLOCKAGE | EXECUTION UNIT | SERIALIZING INSTRUCTION |
|---|---|---|---|---|
| Order Storage Access:<br>eieio | 1 | 1 | LDST | Next Load or Store is Synchronized Relative to All Prior Load or Store |
| Cache Control:<br>icbi | 1 | 1 | LDST,<br>I-Cache | No |

NOTES:
1. Refer to Table 6-11.
2. Refer to **Section 6.4.1 Control Registers**.
3. Refer to Table 6-10.
4. $$\mathrm{DivisionLatency} = \begin{bmatrix} \mathrm{NoOverflow} \Rightarrow 3 + \left(\dfrac{34 - \mathrm{divisorLength}}{4}\right) \\ \mathrm{Overflow} \Rightarrow 2 \end{bmatrix}$$

   Where: $\mathrm{Overflow} = \left(\frac{x}{0}\right) \mathrm{or} \left(\frac{\mathrm{MaxNegativeNumber}}{-1}\right)$
5. $\mathrm{DivisionBlockage} = \mathrm{DivisionLatency}$
6. Blockage of the multiply instruction is dependent on the subsequent instruction. For subsequent multiply instruction the blockage is 1 clock and for subsequent divide it is 2 clocks.
7. Assuming nonspeculative aligned access, on-chip memory and available bus. For details, refer to **Section 6.6.5 Nonspeculative Load Instructions**, **Section 6.6.6 Unaligned Instructions Execution**, and **Section 6.6.9 Instruction Timing**.
8. Although store (as well as mtspr for special registers external to the core) issued to the load store unit buffer frees the CPU pipeline, next load or store will not actually be performed on the bus until the bus is free.

## 8.2 INSTRUCTION EXECUTION TIMING EXAMPLES

All examples assume an instruction cache hit.

### 8.2.1 Load From Data Cache Example

| | |
|---|---|
| l | r12,64 (SP) |
| sub | r3,r12,3 |
| addic | r4,r14,1 |
| mulli | r5,r3,3 |
| addi | r4,3(r0) |

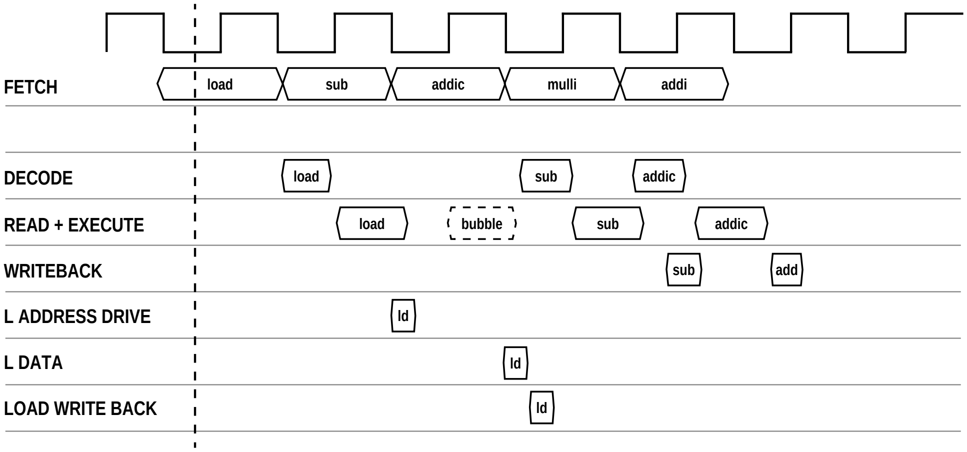


**Figure 8-1. Load From Data Cache Example**

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

## 8.2.2 Writeback Examples

### 8.2.2.1 WRITEBACK ARBITRATION EXAMPLES.

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

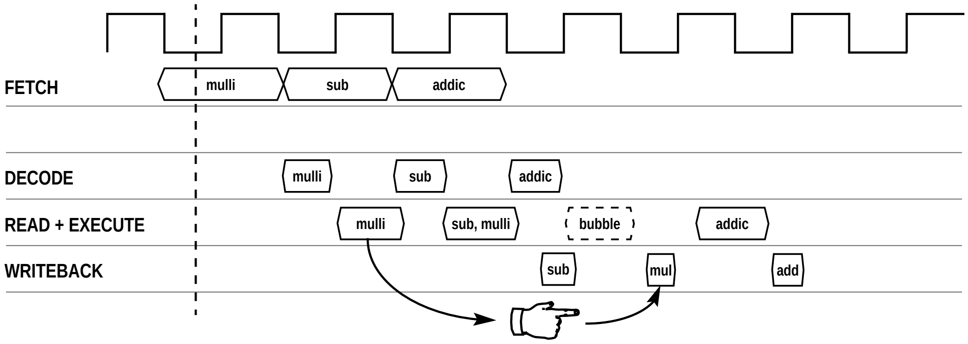


**Figure 8-2. Writeback Arbitration Example I**

The addic is dependent on the mulli result. Since 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   r4,r3,1
```

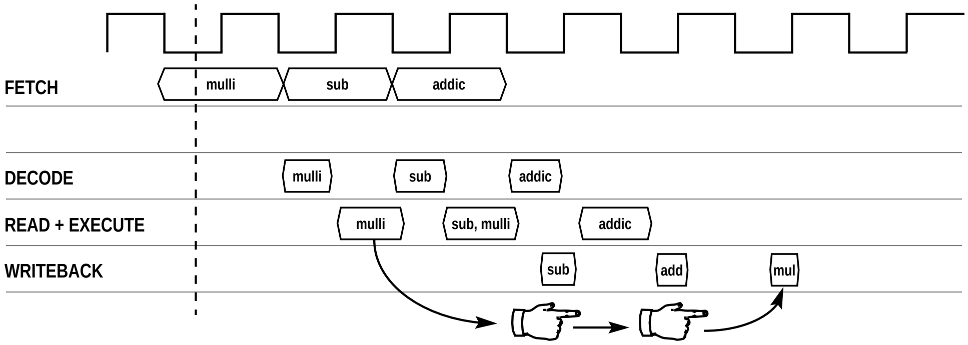


**Figure 8-3. Writeback Arbitration Example II**

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

### 8.2.2.2 LOAD PRIVATE WRITEBACK BUS.

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

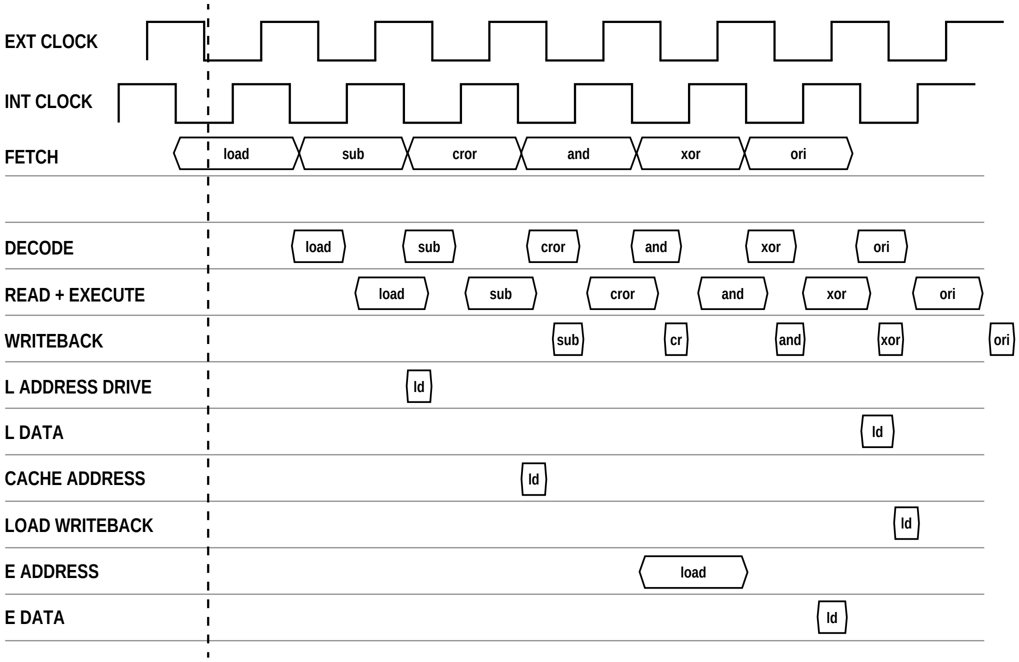


**Figure 8-4. Load Private Writeback Bus Example**

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

### 8.2.3 Fastest External Load (Data Cache Miss) Example

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

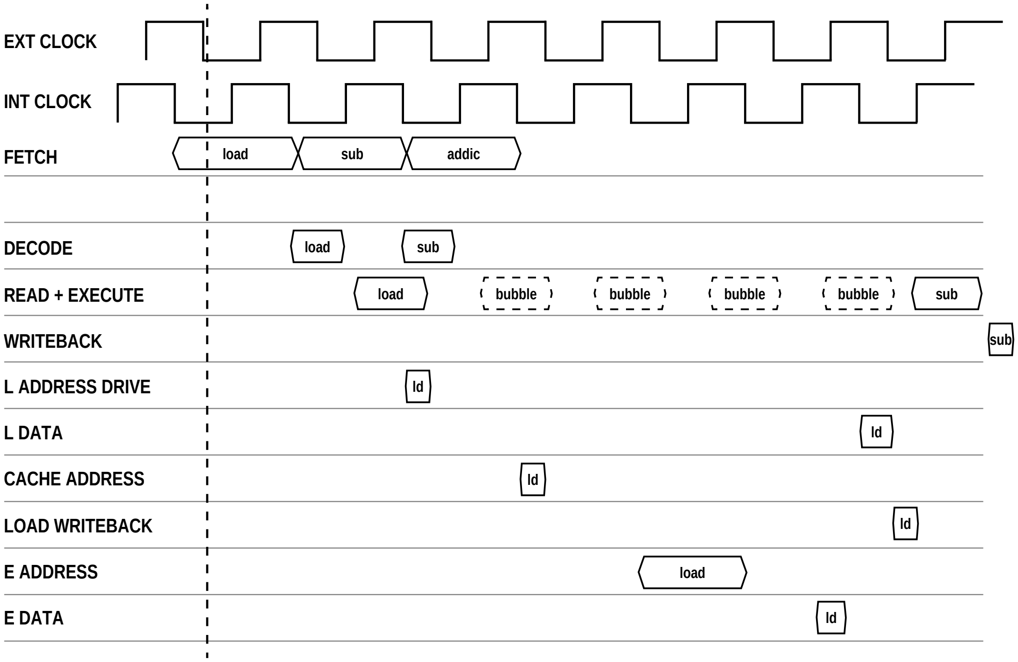


**Figure 8-5. External Load Example**

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

### 8.2.4 History Buffer Full Example

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

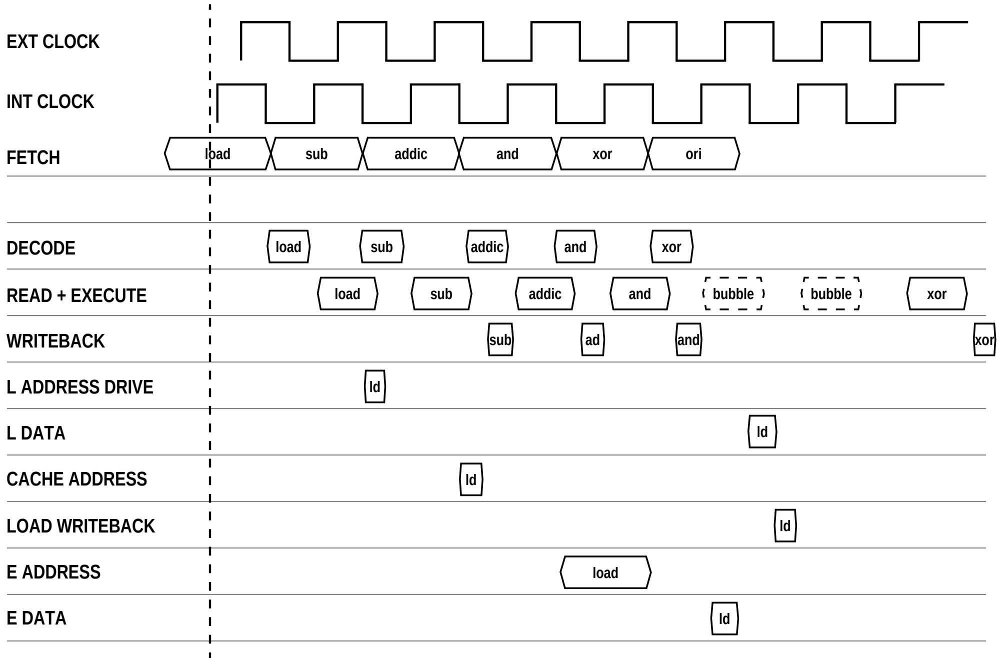


Figure 8-6. History Buffer Full Example

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

### 8.2.5 Branch Folding Example

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

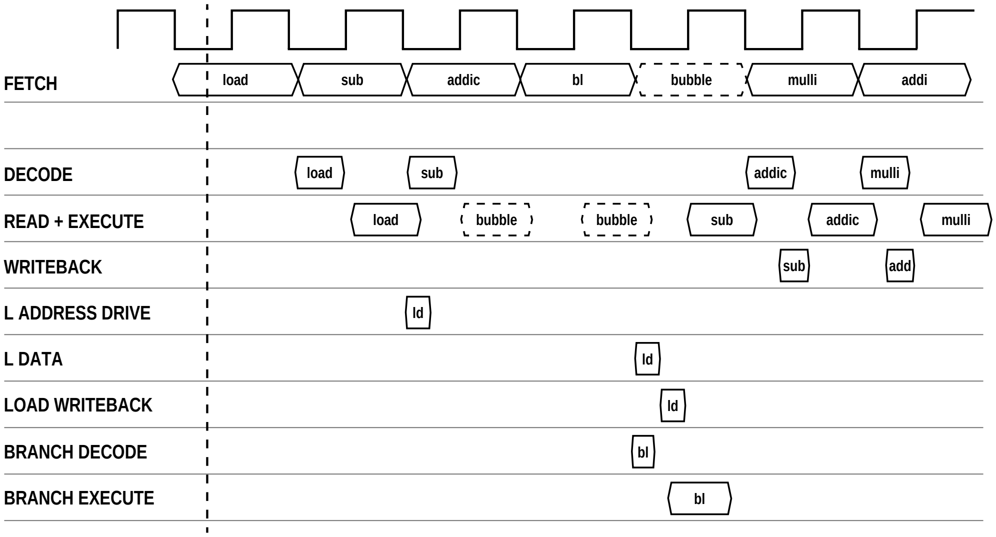


**Figure 8-7. Branch Folding Example**

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

### 8.2.6 Branch Prediction Example

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

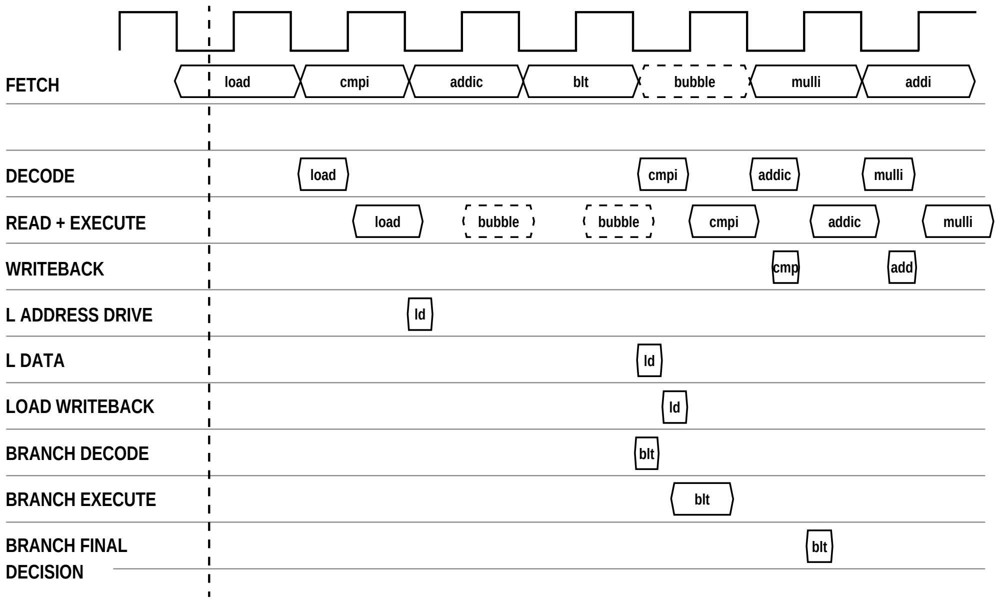


**Figure 8-8. Branch Prediction Example**

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

# SECTION 9
# INSTRUCTION CACHE

## 9.1 OVERVIEW

The MPC821 instruction cache (I-cache) is a 4 kbyte two-way set associative cache. The cache organization is 128 sets, two lines per set, and four words per line. Cache lines are aligned on 4-word boundaries in memory. The cache access cycle begins with an instruction request from the instruction unit in the core. In the case of a cache hit, the instruction is delivered to the instruction unit. In case of a cache miss, the cache initiates a burst read cycle on the internal bus with the address of the requested instruction. The first word received from the bus is the requested instruction. The cache forwards this instruction to the instruction unit of the core as soon as it is received from the internal bus. A cache line is then selected to receive the data which will be coming from the bus. A least recently used (LRU) replacement algorithm is used to select a line when no empty lines are available.

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

## 9.2 FEATURES

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

- 4-kbyte two-way set associative, four words in a line
- LRU replacement policy
- Parked on the internal bus
- Lockable SRAM (cache line granularity)
- "Critical word first", burst access
- Stream hit (allows fetch from the burst buffer and of the word currently on the internal bus)
- Serves the core request in parallel to bring the tail of the previous missed line (hit under miss)

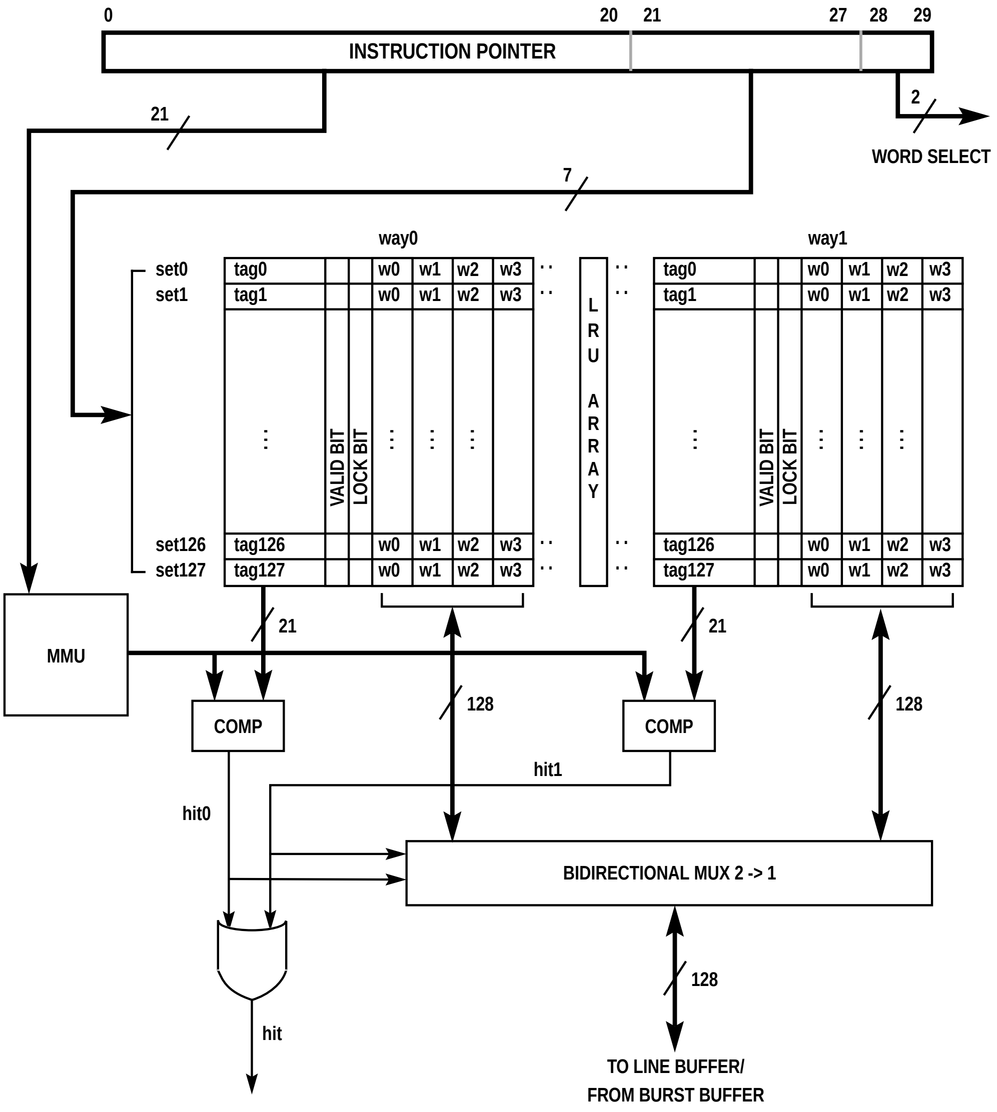


**Figure 9-1. Instruction Cache Organization Block Diagram**

- Cache control
  - — Supports PowerPC™ invalidate instruction
  - — Supports load and lock (cache line granularity)
- Supports cache inhibit
  - — As a cache mode of operation (cache disable)
  - — On memory regions (supported by the MMU)

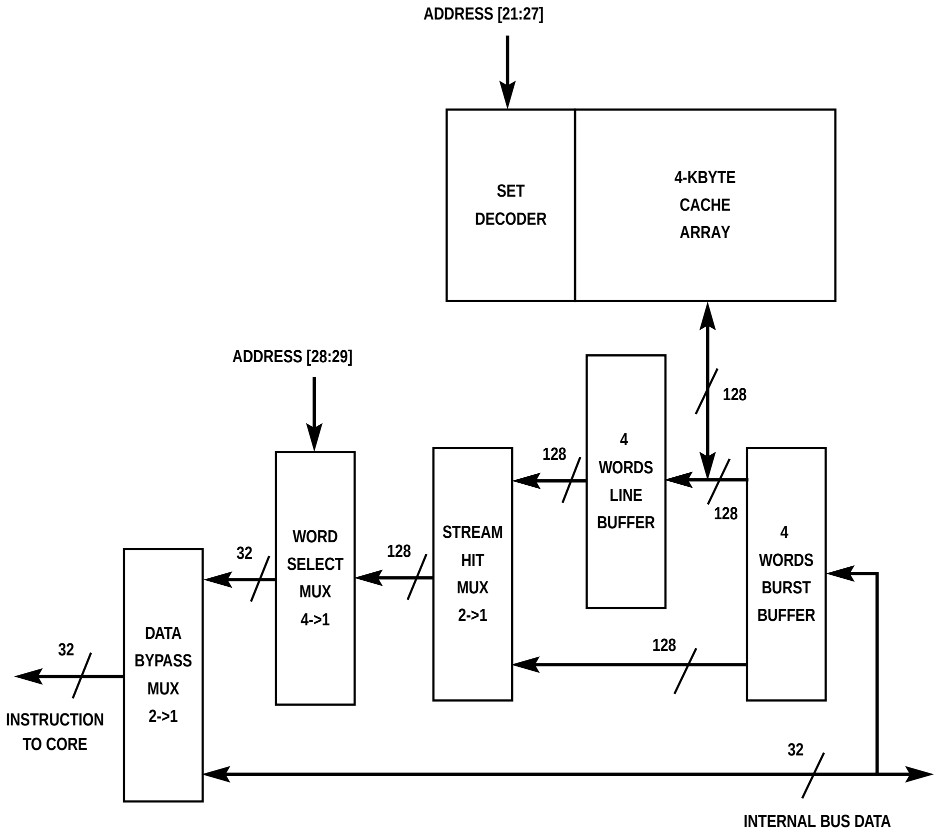


**Figure 9-2. Cache Data Path Block Diagram**

- Efficiently uses the pipeline of the internal bus by initiating a new burst cycle (if miss is detected) while bringing the tail of the previous missed line
- Performance enhanced for cache-inhibited regions by fetching a full line to the internal burst buffer. Instructions stored in the burst buffer and those originated in a cache-inhibited region, are only used once before being refetched.
- Instruction unit request has priority over a burst buffer write to array (burst buffer holds last missed data), thus increasing the overall CPU performance
- Miss latency is reduced by sending address to the cache and internal bus simultaneously and aborting when hit before cycle goes external
- Minimum operational power consumption
- Read/write capability of tags (including all attributes) and data arrays (for debugging and testing purposes
- Special support when the MPC821 processor is under debug. Refer to **Section 9.10 Debug Support** for more information

## 9.3 PROGRAMMING MODEL

Three of the MPC821 special purpose control registers are used to control the I-cache:

- IC_CST—I-cache control and status register
- IC_ADDR—I-cache address register
- IC_DAT—I-cache data port register (read-only)

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

**Table 9-1. I-Cache Control and Status Register**

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0 | IEN | I-Cache Enable Status Bit | 0 - I-Cache is Disabled<br>1 - I-Cache is Enabled<br>This Bit is a Read-Only Bit. Any Attempt to Write it is Ignored |
| 1 - 3 | | Reserved | |
| 4 - 6 | CMD | I-Cache Commands, Reading These Bits Always Delivers a ‘0’ | 000 - Reserved<br>001 - Cache Enable<br>010 - Cache Disable<br>011 - Load & Lock<br>100 - Unlock Line<br>101 - Unlock All<br>110 - Invalidate All<br>111 - Reserved |
| 7 - 9 | | Reserved | |
| 10 | CCER1 | I-Cache Error Type 1 | 0 - No Error<br>1 - Error<br>These Bits Are Sticky, Set By the Hardware. These Bits Are Read-Only. Reading These Bits Clears Them. |
| 11 | CCER2 | I-Cache Error Type 2 | |
| 12 | CCER3 | I-Cache Error Type 3 | |
| 13 - 31 | | Reserved | |

NOTE: Reset value: 0x00000000.

Table 9-2. I-Cache Address Register

| BITS | MNEMONIC | DESCRIPTION |
|---|---|---|
| 0-31 | ADR | The Address to be Used in the Command Programmed in the Control and Status Register. |

NOTE: Reset value: Undefined.

Table 9-3. I-Cache Data Port Register

| BITS | MNEMONIC | DESCRIPTION |
|---|---|---|
| 0-31 | DAT | The Data Received When Reading Information From the I-Cache. |

NOTE: Reset value: Undefined.

## 9.4 INSTRUCTION CACHE REGULAR OPERATION

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

### 9.4.1 Instruction Cache Hit

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

### 9.4.2 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. A cache line is then selected to receive the data which will be coming from the bus. The selection algorithm gives first priority to invalid lines. If neither of the two lines in the selected set are invalid, then the least recently used line is selected for replacement. Locked lines are never replaced. The transfer begins with the word requested by the instruction unit (critical word first), followed by the remaining words (if any) of the line, then by the word at the beginning of the lines (wraparound).

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

Together with the missed word, an indication from the internal bus on a noncacheable device might arrive. In case such an indication is received, the line is written only to the burst buffer and not to the cache. Instructions stored in the burst buffer and originated in a cache-inhibited memory region, are only used once before being refetched. Refer to **Section 9.5.6 Instruction Cache Read** for more information.

### 9.4.3 Instruction Fetch On A Predicted Path

The core implements branch prediction to allow branches to issue as early as possible. This mechanism allows instruction prefetch to continue while an unresolved branch is being computed and the condition is being evaluated. Instructions fetched after unresolved branches are 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 MPC821 I-cache does not initiate a miss sequence in most cases when the instruction is inside a predicted path. The MPC821 I-cache evaluates fetch requests to see if they are inside a predicted path. If a hit is detected, the requested data 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.

## 9.5 INSTRUCTION CACHE COMMANDS

The MPC821 instruction cache supports the PowerPC invalidate instruction with some additional commands that help control the cache and debug the information stored in it. The additional commands are implemented using the three special purpose control registers mentioned previously in **Section 9.3 Programming Model**. Most of the commands are executed immediately after the control register is written and cannot generate any errors. Therefore, when executing these commands there is no need to check the error status in the IC_CST register.

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

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

## 9.5.1 Instruction Cache Invalidate Commands

**9.5.1.1 INSTRUCTION CACHE BLOCK INVALIDATE.** The MPC821 implements the PowerPC instruction cache block invalidate (icbi) as if they pertain only to the MPC821 instruction cache. This instruction does not broadcast on the external bus and the MPC821 does not snoop this instruction if broadcast by other masters. This command is not privileged and has no associated error cases. The I-cache performs this instruction in one clock cycle. To accurately calculate the latency of this instruction, bus latency should be taken into consideration.

**9.5.1.2 INVALIDATE ALL INSTRUCTION CACHE.** An additional invalidate operation is supported by the MPC821, the invalidate all instruction cache. This operation is privileged and any attempt to perform it when the CPU is in the problem state ($MSR_{PR}$ =1) results in a program interrupt. When invoked and if $MSR_{PR}$ =0 all valid lines in the cache, except lines that are locked, are made invalid. As a result of this command, the LRU of all lines points to an unlocked way or to way 0 if both lines are unlocked. This last feature is useful to initialize the I-cache out of reset. For more information refer to **Section 9.9 Reset Sequence**. To invalidate the whole cache, set the invalidate all command in the IC_CST register. This command has no associated error cases. The I-cache performs this instruction in one clock cycle. In order to accurately calculate the latency of this instruction, bus latency should be taken into consideration.

## 9.5.2 Load & Lock

The load & lock operation is used to lock critical code segments in the instruction cache. This operation is privileged and any attempt to perform it when the CPU is in the problem state ($MSR_{PR}$ =1) results in a program interrupt. The load & lock operation is performed on a cache line granularity. After a line is locked, it operates as a regular instruction SRAM. It is not replaced during misses and it is not affected by invalidate commands. The hardware correct operation relies on the software following the exact steps mentioned in **Section 9.8 Updating Code And Memory Regions Attributes**. To load & lock one line, the following sequence should be followed:

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

After the load & lock command is written to the IC_CST register, the cache checks if the line containing the byte addressed by the IC_ADR is in the cache (hit). If it is in the cache, the line is locked and the command terminates with no exception. If it is not in the cache, a regular miss sequence is initiated. After the whole line is placed in the cache, the line is locked. The user must check the error type bits in the IC_CST register to determine if the load & lock operation completed properly. The load & lock command can generate two possible errors:

- Type 1—Bus error in one of the cycles that fetches the line.
- Type 2—No place to lock. It is the responsibility of the user to make sure that there is at least one unlocked way in the appropriate set.

### 9.5.3 Unlock Line

The unlock line operation is used to unlock previously locked cache lines. This operation is privileged and any attempt to perform it when the CPU is in the problem state ($MSR_{PR}$ =1) results in a program interrupt. The unlock line operation is performed on a cache line granularity. In case the line is found in the cache (hit), it is unlocked and starts to operate as a regular valid cache line. In case the line is not found in the cache (miss), no operation is done and the command terminates with no exception. To unlock one line the following unlock line sequence should be followed:

1. Write the address of the line to be unlocked into the IC_ADR.
2. Set the unlock line command in the IC_CST register.

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

### 9.5.4 Unlock All

The unlock all operation is used to unlock the whole cache. This operation is privileged and any attempt to perform it when the CPU is in the problem state ($MSR_{PR}$ =1) results in a program interrupt. This operation is performed on all cache lines. In case a line is locked, it is unlocked and starts to operate as a regular valid cache line. In case a line is not locked or if it is invalid, no operation is performed. To unlock the whole cache, set the unlock all command in the IC_CST register. This command has no associated error cases. The I-cache performs this instruction in one clock cycle. To accurately calculate the latency of this instruction, bus latency should be taken into consideration.

### 9.5.5 Instruction Cache Inhibit

Two levels of cache inhibit are supported in the MPC821—as a cache mode of operation (cache disable) and on memory regions (supported by the MMU). To disable the instruction cache, set the cache disable command in the IC_CST register. This operation is privileged and any attempt to perform it when the CPU is in the problem state ($MSR_{PR}$ =1) results in a program interrupt. This command has no error cases that the user needs to check.

To enable the instruction cache, set the cache enable command in the IC_CST register. This operation is privileged and any attempt to perform it when the CPU is in the problem state ($MSR_{PR}$ =1) results in a program interrupt. This command has no error cases that the user needs to check. When fetching from cache-inhibited regions the full line is brought to the internal burst buffer. Instructions that are stored in the burst buffer and originate from a cache-inhibited region, can be sent to the MPC821 core, at most, once before being refetched. A memory region can be programmed in the memory management unit (MMU) to be cache-inhibited. When changing a memory region to be cache inhibited, the user must unlock all previously locked lines containing code that originated in this memory region, invalidate all lines containing code that originated in this memory region and execute an isync instruction.

**NOTE**

Failing to follow these steps may result in code from cache-inhibited regions to be left inside the cache and, therefore, a reference to a cache-inhibited region may result in a cache hit. In case a reference to a cache-inhibited region results in a cache hit, the data is delivered to the MPC821 core from the cache and not from memory.

When the freeze signal is asserted by the MPC821, indicating that the MPC821 is under debug, all fetches from the cache are treated as if they were from cache-inhibited memory region. For more information on the cache debug support, refer to **Section 9.10 Debug Support**.

## 9.5.6 Instruction Cache Read

The MPC821 allows the user to read all data stored in the instruction cache, including the content of the tags array. This operation is privileged and any attempt to perform it when the CPU is in the problem state ($MSR_{PR}$ =1) results in a program interrupt. To read the data stored in the I-cache, the following sequence should be followed:

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

To be able to access all parts of the I-cache the IC_ADR is divided into the fields in Table 9-4.

**Table 9-4. IC_ADR Bits Function for the Cache Read Command**

| 0-17 | 18 | 19 | 20 | 21-27 | 28-29 | 30-31 |
|---|---|---|---|---|---|---|
| Reserved | 0 - Tag<br>1 - Data | 0 - Way 0<br>1 - Way 1 | Reserved | Set Select | Word Select (Used Only For Data Array) | Reserved |

When read from the data array, the 32 bits of the word selected by the IC_ADR is placed in the target general-purpose register. Likewise, when read from the tag array, the 21 bits of the tag selected by the IC_ADR and other relative information of the tag are placed in the target general-purpose register. The following table provides the bit layout of the I-cache data register when reading a tag.

**Table 9-5. IC_DAT Bits Layout When Reading a Tag**

| 0-20 | 21 | 22 | 23 | 24 | 25-31 |
|---|---|---|---|---|---|
| Tag Value | Reserved | 0 - Not Valid<br>1 - Valid | 0 - Not Locked<br>1 - Locked | LRU Bit | Reserved |

### 9.5.7 Instruction Cache Write

Instruction cache write is only enabled when the MPC821 is in the test mode.

## 9.6 RESTRICTIONS

Zero wait state devices that are placed on the internal bus are considered as cache-inhibited memory region and the hardware correct operation relies on the software following the exact steps mentioned in **Section 9.8 Updating Code And Memory Regions Attributes**. It is not advised to perform load & lock from zero wait state devices that are placed on the internal bus, since the data is not guaranteed to be fetched from the I-cache. In most cases, it is fetched from the device, but found in the I-cache.

## 9.7 INSTRUCTION CACHE COHERENCY

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

## 9.8 UPDATING CODE AND MEMORY REGIONS ATTRIBUTES

When updating code or changing memory regions programming (in the chip-select logic) the user must perform the following steps:

- Update code/change memory region programming in the chip-select logic
- Execute the sync instruction to ensure the update/change operation finished
- Unlock all locked lines containing code that was updated
- Invalidate all lines containing code that was updated
- Execute the isync instruction

## 9.9 RESET SEQUENCE

To simplify the debug task of the system the I-cache is only forced to be disabled during hardware reset (IC_CST$_{EN}$ = 0). This feature enables the user to investigate the exact state of the I-cache prior to the event that asserted the reset. To ensure proper operation of the instruction cache after reset, unlock all, invalidate all, and instruction cache enable must be performed.

## 9.10 DEBUG SUPPORT

### 9.10.1 General

The MPC821 can be debugged either in debug mode or by a software monitor debugger. In both cases, the core of the MPC821 CPU asserts the internal freeze signal. When freeze is asserted the I-cache treats all misses as if they were from cache-inhibited regions (misses are loaded ONLY to the burst buffer) and, therefore, assuming the debug routine is not in the I-cache, the cache state remains exactly the same. When freeze is asserted, hits are still read from the array and the LRU bits are updated. Therefore, in the simple case of the debug routine, if it is not already in the I-cache, it is read from memory like any other miss. However, for performance reasons, it might be preferable to run the debug routine from the cache and to do that follow these steps (there could be some variations):

1. Save both ways of the sets that are needed for the debug routine by reading the tag value, LRU bit value, valid bit value, and lock bit value.
2. Unlock the locked ways in the selected sets.
3. Use load & lock to load and lock the debug routine into the I-cache (load & lock operates the same when freeze is asserted).
4. Run the debug routine, all accesses to it will result in hits.

After the debug routine is finished, it is possible to restore the old state of the I-cache by following these steps:

1. Unlock and invalidate all the sets that are used by the debug routine (both ways)
2. Use load & lock to restore the old sets
3. Unlock the ways that were not locked before
4. To restore the old state of the LRU make sure the last access is done in the MRU way (not the LRU way). An access in this description is either load & lock or unlock.

### 9.10.2 Instruction Fetch From The Development Port

When the MPC821 is in debug mode, all instructions are fetched from the development port, regardless of the address generated by the MPC821 core. Therefore, the I-cache is practically bypassed when the MPC821 is in debug mode.

# SECTION 10
# DATA CACHE

## 10.1 OVERVIEW

The MPC821 data cache is a 4 kbyte two-way set associative cache. The cache organization is 128 sets, two lines per set, and four words per line. Cache lines are aligned on 4-word boundaries in memory. Two state bits are included in each cache line and implement invalid, modified-valid and unmodified-valid states of the data cache. The cache coherency in a multiprocessor environment is maintained by the software and supported by a fast hardware invalidation capability. The cache is designed for both writeback and writethrough modes of operation and a least recently used (LRU) replacement algorithm is used to select a line when no empty lines are available.

## 10.2 FEATURES

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

- 4-kbyte two-way set associative physically addressed data cache
- Single-cycle cache access on hit and 1 clock latency added for miss
- Four word line size
- "Critical word first" and four words burst line fill
- LRU replacement policy
- 32-bit interface to load/store unit
- One word write buffer
- Cache is lockable online granularity
- Copyback/writethrough operation programmed per MMU page
- Coherency is maintained by the software only and no snoop is supported
- Cache operation is blocked under miss, until the critical word is delivered to the core
- Hit under miss operation
- Full data cache PowerPC™ control operations
- Implementation specific single operation
  - — INVALIDATE all
  - — UNLOCK all
  - — FLUSH by cache location

## 10.3 DATA CACHE ORGANIZATION

The data cache is a 4-kbyte, two-way set associative physically addressed cache. The caches have 16-byte line size and a 32-bit data path to and from the load/store unit, allowing for a 4-byte transfer per cycle.

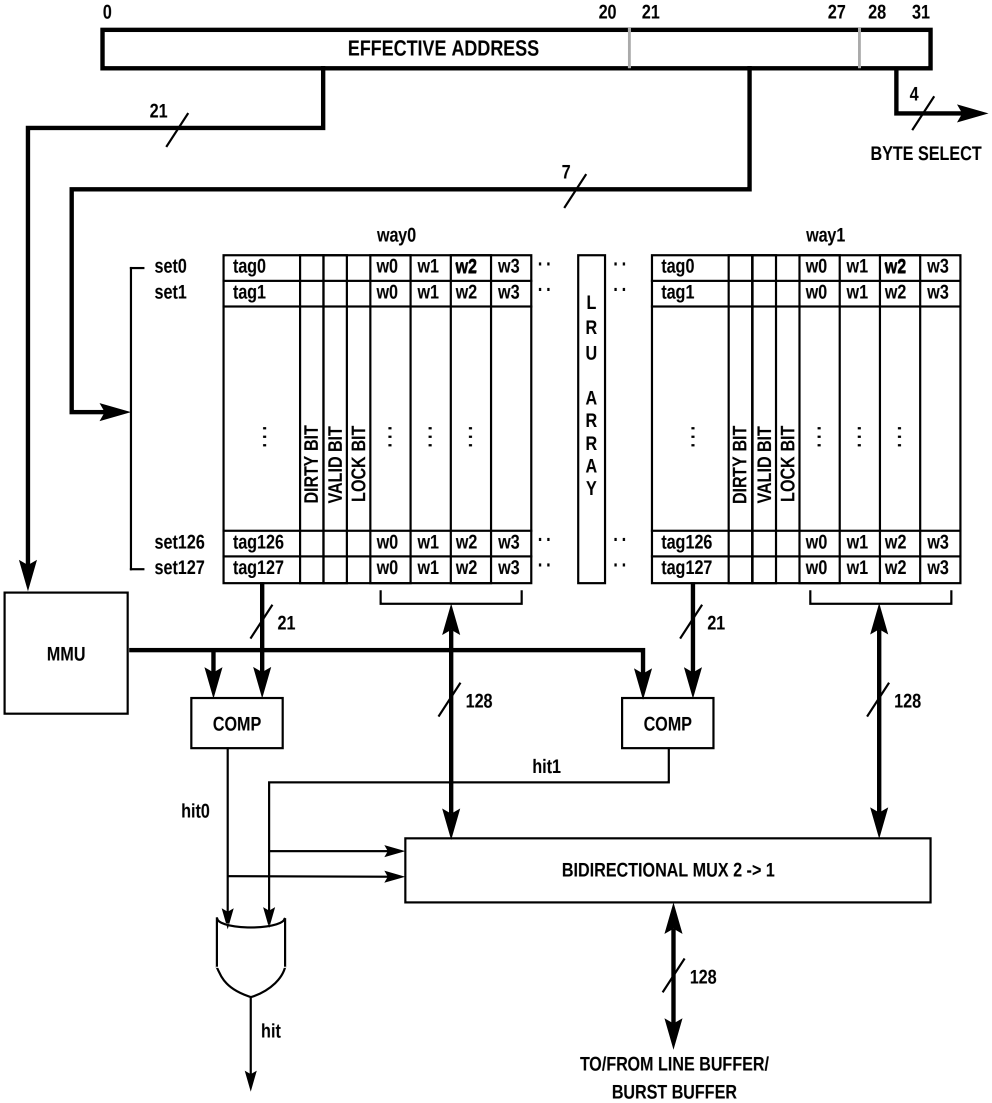


**Figure 10-1. Data Cache Organization Block Diagram**

## 10.4 PROGRAMMING MODEL

### 10.4.1 PowerPC Architecture Instructions

The following PowerPC instructions are supported by the data cache.

**10.4.1.1 POWERPC USER INSTRUCTION SET ARCHITECTURE (BOOK I).** The data cache supports the sync instruction through a cache pipe clean indication to the core.

**10.4.1.2 POWERPC VIRTUAL ENVIRONMENT ARCHITECTURE (BOOK II).** The data cache supports the following instructions:

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

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

### 10.4.2 Implementation Specific Operations

The MPC821 data cache include some extended features in addition to the definition of the PowerPC architecture. The following operations are implementation specific operations supported by the MPC821 data cache:

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

### 10.4.3 Data Cache Special Registers

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

- DC_CST—Data cache control and status register
- DC_ADR—Data cache address register
- DC_DAT—Data cache data register

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

### Table 10-1. Data Cache Control and Status Registers

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0 | DEN | Data Cache Enable Status Bit | 0 - Data Cache is Disabled<br>1 - Data Cache is Enabled<br>This Bit is a Read-Only bit. Any Attempt To Write To It Is Ignored. |
| 1 | DFWT | Data Cache Force Writethrough | 0 - Data Cache Mode Determined By MMU<br>1 - Data Cache is Forced Writethrough<br>This Bit is a Read-Only Bit. Any Attempt To Write To It Is Ignored. |
| 2 | LES | Little Endian Swap | 0 - Address of the Data and the Instruction Caches is the Unchanged Address From the Core. No Byte Swap is Done on the Data and Instruction Caches' External Accesses.<br>1 - Address Munging Performed By the Core is Reversed Before Accessing the Data Cache, the Instruction Cache and Storage. Byte Swap is Performed For the Instruction and Data Cache's External Accesses. This bit is a Read-Only Bit. Any Attempt to Write to it is Ignored. |
| 3 | | Reserved | |
| 4 - 7 | CMD | Data Cache Commands<br>Reading These Bits Always Delivers a "0" | 0000 - Reserved<br>0010 - Data Cache Enable<br>0100 - Data Cache Disable<br>0110 - Lock Line<br>1000 - Unlock Line<br>1010 - Unlock All<br>1100 - Invalidate All<br>1110 - Flush Data Cache Line<br>0001 - Set Force Writethrough Mode<br>0011 - Clear Force Writethrough Mode<br>0101 - Set Little Endian Swap Mode<br>0111 - Clear Little Endian Swap Mode<br>Others - Reserved |
| 8 - 9 | | Reserved | |

### Table 10-1. Data Cache Control and Status Registers (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 10 | CCER1 | Data Cache Error Type 1<br>Copyback Error. Machine Check Interrupt is Generated When This Bit is Set. | 0 - No Error<br>1 - Error<br>These Bits Are Sticky, Set by the Hardware. These Bits are Read-Only. Reading These Bits Clears Them. |
| 11 | CCER2 | Data Cache Error Type 2<br>Load and Lock/Unlock Error and Implementation Specific Flush Cache Line Error Indication. Set if Both Ways of the Needed Set Are Already Locked (For Load and lock) or When Bus Error was Asserted During the Load and Lock/Unlock or Flush Cache Line Operation. No Machine Check Interrupt is Generated When This Bit is Set Due to the Execution of the Load and Lock/Unlock Commands. | |
| 12 | CCER3 | Data Cache Error Type 3<br>Reserved | |
| 13 - 31 | — | Reserved | |

NOTE: Refer to **Section 14** Endian Modes for details regarding how the LES bit is used to achieve the required endian behavior.

### Table 10-2. Data Cache Address Register

| BITS | MNEMONIC | DESCRIPTION |
|---|---|---|
| 0-31 | ADR | The Address to be Used in the Command Programmed in the Control and Status Register. Also the Internal Address for the Read Tags and Registers Operation. |

**10.4.3.1 CACHE STRUCTURES READ.** To read the data stored in the data cache tags or registers, the following sequence should be followed:

1. Write to the DC_ADR (the address of the data to be read). Notice that it is also possible to read this register for debugging purposes.
2. Read the DC_DAT register.

The DC_ADR is divided into the following fields when the internal parts of the data cache are read.

**Table 10-3. DC_ADR Bits Function for the Cache Read Operation**

| 0-17 | 18 | 19 | 20 | 21-27 | 28-31 |
|---|---|---|---|---|---|
| Reserved | 0- Tags | 0 - Way 0<br>1 - Way 1 | Reserved | Set Number | Reserved |
| | 1- Registers | Reserved | | Register Number | Reserved |

When Bit 18 = 1, the register number field specifies which register is to be read. The following registers and their encoding are supported:

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

When reading from the DC_DAT register, the 21 bits of the tag selected by the DC_ADR, with relevant information of the tag, are placed in the target general-purpose register. The following table illustrates the bit layout of the DC_DAT register when reading a tag. Writing to DC_DAT is illegal. A write to DC_DAT results in an undefined data cache state.

**Table 10-4. DC_DAT Bits Layout When Reading a Tag**

| 0-20 | 21 | 22 | 23 | 24 | 25 | 25-31 |
|---|---|---|---|---|---|---|
| Tag Value | Reserved | 0 - Not Valid<br>1 - Valid | 0 - Not Locked<br>1 - Locked | LRU Bit of this Set | 0 - Clean<br>1 - Dirty | Reserved |

## 10.5 DATA CACHE OPERATION

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

### 10.5.1 Data Cache Read

- Read Hit—On a cache hit, the requested word is immediately transferred to the load/store unit and the LRU state of the set is updated, but no state transition occurs and the access time is 1 clock (zero wait state).
- Read Miss—A line in the cache is selected to hold the data which will be fetched from memory. The selection algorithm gives first priority to invalid lines and if both lines are invalid the line in way zero is selected first. If neither of the two candidate lines in the selected set is invalid, then one of the lines is selected by the LRU algorithm for

replacement. If the selected line is valid-modified (dirty), then it is kept in a special buffer to be written out (flushed) to memory later.

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

After the line with the requested data has been brought from memory, the dirty line kept in the buffer is sent to the SIU to be written out (flushed) to memory. If a bus error is detected during the fetch of the missed “critical word”, a machine check interrupt is generated and if a bus error occurs on any other word in the line transfer, the line is marked invalid. On the other hand, if no bus error is encountered, the cache line is marked unmodified-valid. If a bus error is detected during the dirty line flush, a machine check interrupt is generated (the dirty line flush error is an imprecise interrupt). The address and data of the line can be read as specified in **Section 10.4.3.1 Cache Structures Read**.

## 10.5.2 Data Cache Write

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

**10.5.2.1 COPYBACK MODE.** In copyback mode, write operations do not necessarily update the external memory. For this reason the copyback mode is the preferred mode of operation when it is necessary to minimize bus bandwidth utilization and operational power consumption.

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

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

After the line with the requested data has been brought from memory, the dirty line kept in the buffer is sent to the SIU to be written out (flushed) to memory. The data cache can support further requests, as long as they hit in the cache, while flushing the dirty line to memory. If a bus error is detected during the fetch of the missed line, even on a word not accessed by the load/store unit, the cache line is not modified and a machine check interrupt is generated. If a bus error is detected during the dirty line flush, a machine check interrupt is generated (the dirty line flush error is an imprecise interrupt). The address and data of the line can be read as specified in **Section 10.4.3.1 Cache Structures Read**.

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

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

## 10.5.3 Data Cache-Inhibited Accesses

If the cache access is to a page which has the cache inhibit (CI) bit set in the MMU, the following action is taken:

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

## 10.5.4 Data Cache Freeze

The MPC821 can be debugged either in debug mode or by a software monitor debugger. In both cases the core of the MPC821 asserts the internal freeze signal. For a detailed description of the MPC821 debug support refer to **Section 19 Development Support**.

When freeze is asserted, the data cache assumes the following behavior:

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

### 10.5.5 Data Cache Coherency Support

The MPC821 data cache provides no support for snooping external bus activity. All coherency between the internal caches and memory/devices external to the extended core must be controlled by the software. In addition, there is no mechanism provided for DMA or other internal masters to access the data cache directly.

## 10.6 DATA CACHE CONTROL

### 10.6.1 Data Cache Flushing And Invalidation

The MPC821 allows flushing and invalidation of the data cache under software control. The data cache may be invalidated through writing unlock all and invalidate all commands to the DC_CST register. The data cache is not automatically invalidated on reset. It must be invalidated under software control. Flushing of the data cache can be performed by a software loop either by using the PowerPC architecture instructions dcbst, dcbf, or the implementation specific data cache flush cache line command. Notice that the PowerPC architecture instructions flushes a line indexed by the address it represents, while the implementation specific command indexes the line by the physical location within the data cache.

When there is a need to restrict the flushing to a specific memory area or to keep architecture compliant, it is recommended to use the PowerPC architecture instructions. When there is a need to flush the entire data cache and there is no concern for compatibility, using the implementation specific command is more efficient. If a bus error occurs while executing the dcbf and dcbst instructions or the flush cache line implementation specific command, the user should retrieve the data of the cache line specified by these operations from the copyback data register [0:3] rather than from the data cache array.

### 10.6.2 Data Cache Disabling

The data cache may be enabled or disabled through the use of data cache enable and data cache disable written to the DC_CST register. In the disabled state, the cache tag state bits are ignored and all accesses are propagated to the bus as single beat transactions. The default after reset state of the data cache is disabled. Disabling the data cache does not affect the data address translation logic and translation is still controlled by the $MSR_{DR}$ bit.

Any write to the DC_CST register must be preceded by a sync instruction. This prevents the data cache from being disabled or enabled in the middle of a data access. When the data cache generates an interrupt as a result of the bus error on the copyback or on the implementation specific flush cache line command, it enters the disable state. Operation of the cache when it is disabled is similar to cache-inhibit operation.

### 10.6.3 Data Cache Locking

Each line of the data cache may be independently locked through the use of the lock line command written to the DC_CST register, but replacement line fills are not performed to a locked line. A flush or invalidate of a locked line cache is ignored by the data cache. Any write to the DC_CST register must be preceded by a sync instruction. This prevents a cache from being locked during a line fill.

### 10.6.4 Data Cache Control Instructions

**10.6.4.1 DCBI, DCBST, DCBF AND DCBZ INSTRUCTIONS.** The dcbz, dcbi, dcbst, and dcbf instructions will operate on a block basis of cache line, which is 16 bytes (4 words) in length. A data TLB miss exception is generated if the effective address of one of these instructions cannot be translated and data address relocation is enabled.

**10.6.4.2 TOUCH INSTRUCTIONS.** The dcbt and dcbtst instructions in the MPC821 will operate on a block basis of cache line which is 16 bytes (4 words) in length. They are treated as a no-operation if the effective address of one of these instructions cannot be translated and relocation is enabled.

**10.6.4.3 STORAGE SYNCHRONIZATION/RESERVATION IMPLEMENTATION.** The lwarx and stwcx instructions are implemented according to the PowerPC architecture requirements. When the storage accessed by the lwarx and stwcx instructions is in the cache-allowed mode it is assumed that the system works with the single master in this storage region, therefore, in a case of the data cache miss, the access on the internal and external busses does not have a reservation attribute.The MPC821 does not cause the system data storage error handler to be invoked if the storage accessed by the lwarx and stwcx instructions is in the writethrough required mode. The MPC821 does not provide support for snooping an external bus activity outside the chip. The provision is made to cancel the reservation inside the MPC821 by using the CR_B and KR_B input pins. The data cache has a snoop logic to monitor the internal bus for CPM accesses of the address associated with the last lwarx instruction. Refer to **Section 13.5.9 Storage Reservation**.

### 10.6.5 Data Cache Structures Read

In order to allow debug and recovery actions, the MPC821 allows the user to read the content of the tags array as well as the last copyback address and data buffers (refer to **Section 10.4.3 Data Cache Special Registers**). This operation is privileged and any attempt to perform it when the CPU is in the problem state ($MSR_{PR}$ =1) results in a program interrupt.

# SECTION 11
# MEMORY MANAGEMENT UNIT

## 11.1 OVERVIEW

The MPC821 implements a virtual memory management scheme that provides cache control, storage access protections, and effective to real address translation. The implementation includes separate instruction and data memory management units. The MPC821 MMU is compliant with the *PowerPC™ Operating Environment Architecture (Book III)* in relation to the supported types of attributes. A few new modes of operation have been added. The MMU has two modes of operation:

- PowerPC mode with extended encoding
- Domain manager mode

Available protection granularity sizes are page (either 4, 16, 512 kbyte, or 8 Mbyte) or 1-kbyte subpage (1-kbyte subpage resolution is supported for 4-kbyte pages only). Hereafter, the prefix MX_ appearing before a MMU control register name corresponds to both the MI_ and MD_ conditions.

## 11.2 FEATURES

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

- 32-entry fully associative data translation lookaside buffer (TLB)
- 32-entry fully associative instruction translation lookaside buffer
- Multiple page sizes
  - 4 kbyte
  - 16 kbyte
  - 512 kbyte
  - 8 Mbyte
  - 1-kbyte subpage protection granularity for 4-kbyte page size
- High performance
  - 1 clock (zero wait state) access for data cache hit
  - 1 clock (zero wait state) access for instruction cache hit when access is performed from the same 1-kbyte subpage as the previous access
  - One clock penalty for other TLB hit instruction accesses
- Supports up to 16 virtual address spaces
- Supports 16 access protection groups (the group protection overrides the page protection)

- Each entry can be programmed to match either problem accesses, privileged accesses, or both.
- Generates the following interrupts
  - Implementation specific instruction TLB miss interrupt
  - Implementation specific data TLB miss interrupt
  - Implementation specific instruction TLB error interrupt
  - Implementation specific data TLB error interrupt
- Accomplishes software tablewalk updates
  - Data TLB miss and instruction TLB miss interrupts
  - Special purpose registers located in the data MMU
- Supports the following attributes:
  - Changed bit support through the data TLB error interrupt on a write attempt to a nonmodified page
  - Per page writethrough attribute for data storage accesses
  - Per page cache-inhibit attribute
  - Per page guarded attribute for memory-mapped I/O and other nonspeculative regions
- PowerPC $MSR_{IR}$ and $MSR_{DR}$ control MMU translation and protection
- Supports PowerPC tlbie and tlbia instructions. No tlbsync instruction is supported, but it is implemented as a NOP instruction.
- Programming is accomplished by using the PowerPC mtspr**/**mfspr instructions to/from the implementation specific special purpose registers
- One special register is available as a scratch register for software tablewalks
- Designed for minimum power consumption

## 11.3 ADDRESS TRANSLATION

The MPC821 core generates 32-bit effective addresses and when enabled, the MMU translates the effective address to a real address that is used for cache or memory access. If disabled, the effective address is passed directly as the real address to the memory, bypassing the appropriate TLB. Conceptually, the effective address is searched for in tables residing in the memory to provide the real address mapping and storage attributes. For performance reasons, a TLB implemented in each hardware cache to hold recently used address translations. In the MPC821, the table lookup and TLB reload are performed by a software routine with little hardware assistance. This partition simplifies the hardware and gives the system the flexibility to choose the translation table structure.

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

The MPC821 MMU supports a multiple virtual address space model and when enabled, each translation is associated with an ASID. In this case, for the translation to be valid, it’s ASID must be equal to the current address space ID (CASID) that is in effect when an access is performed.

### 11.3.1 Translation Lookaside Buffer Operation

**11.3.1.1 MAINSTREAM OPERATION.** Two TLBs are provided in the MPC821—one for instruction fetches and one for data accesses. The TLB contains pointers to pages in the real memory where data is indexed by the effective page number and it can hold entries with different page sizes. The entry page size controls the number of effective address bits to be compared and the number of least-significant effective address bits that remain untranslated and are passed as least-significant real address bits.

For a 4-kbyte page size, four subpage validity flags are supported, thus 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 pages other than 4-kbyte pages with different valid bits are considered a programming error. The subpage validity flags can be manipulated to implement effective page sizes of 1, 2, 3, 4 kbyte, or any other combination of 1-kbyte subpages. However, subpages of an effective page frame must all map to the same real page. During the translation process, the effective address, the processor problem state ($MSR_{PR}$), and the CASID are provided to the TLB. Refer to Figure 11-1 for more information. In the TLB, the effective address and the CASID are compared with the EPN and the ASID of each entry. The CASID is only compared when the matching entry was programmed as nonshared. See Table 11-11 and Table 11-15 for details.

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

## 11.4 PROTECTION

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

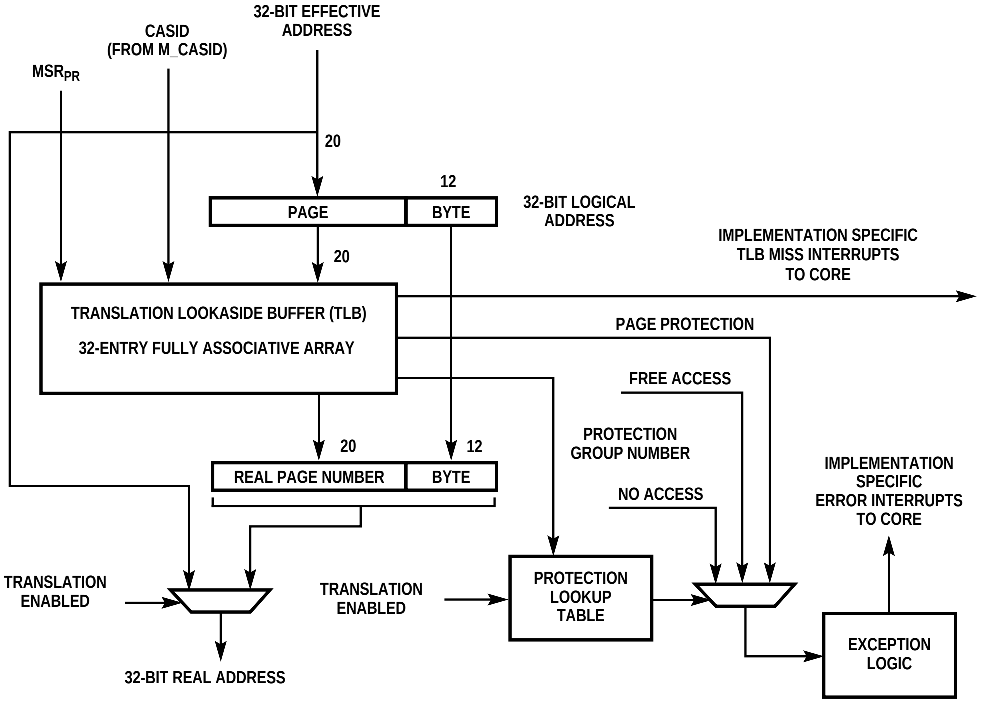


**Figure 11-1. Effective to Real Address Translation For 4-kbyte Pages Block Diagram**

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

## 11.5 STORAGE ATTRIBUTES

### 11.5.1 Reference and Change Bit Updates

The MPC821 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 C (changed) bit updates are implemented by the software, but the hardware treats the C bit (negated) as a write-protect attribute. Therefore, if a write is attempted to a page marked unmodified, that entry is invalidated and an implementation specific data TLB error interrupt is generated.

### 11.5.2 Storage Control Attributes

Each page can have different storage control attributes. The MPC821 supports cache inhibit (CI), writethrough (WT), and guarded (G) attributes, but not the memory coherence (M) attribute. A page that needs to be memory coherent must be programed cache-inhibited. Refer to the definition of these attributes in the *PowerPC™ Virtual Environment Architecture (Book II)*. The effects of the CI and WT attributes in the MPC821 are described in **Section 9 Instruction Cache**.

The G attribute is used to map I/O devices that are sensitive to speculative accesses. An attempt to access a page marked guarded (G bit asserted), forces the access to stall until either the access is nonspeculative or is canceled by the core. Fetching from a guarded storage is prohibited and if it is attempted, an implementation specific instruction storage interrupt is generated. When $MSR_{IR}$ or $MSR_{DR}$ for instruction or data address translation, respectively, are negated, default attributes are used. See Table 11-6 and Table 11-7 for details.

## 11.6 TRANSLATION TABLE STRUCTURE

The MPC821 MMU includes special hardware to assist in a two-level software tablewalk. Other table structures are not precluded. Figure 11-2 and Figure 11-3 illustrate the two levels of translation table structures supported by MPC821 special hardware. When $MD\_CTR_{TWAM}$ = 1, the tablewalk begins at the level one base address in the M_TWB register. The level one table is indexed by the ten most-significant bits ([0:9]) of the effective address to get the level one page descriptor. For 8-Mbyte pages, there must be two identical entries in the level one table for either bit [9]=0 or bit [9]=1. See Table 11-2 for details. The level two base address from the level one descriptor is indexed by the next ten lesser significant bits (bits [10:19]) to find the level two page descriptor. For pages larger than 4 kbytes, the entry in the level two table must be duplicated according to the page size. See Table 11-3 for more information.

During the address translation by the MMU, the most-significant bits of the missed effective address are replaced by the real page address bits from the level two page descriptor and the number of replaced bits depends upon the page size. The rest of the real address bits are taken directly from the effective address. When $MD\_CTR_{TWAM}$ = 0, the tablewalk begins at the level one base address placed in the M_TWB register. The level one table is indexed by the 12 most-significant bits (bits [0:11]) of the effective address to get the level one page descriptor. For 8-Mbyte pages, there must be eight identical entries in the level one table for bits [9:11] of the effective address.

The level two base address from the level one descriptor is indexed by the next ten lesser significant bits (bits [12:21]) to find the level two page descriptor. For pages larger than 1-kbyte, the entry in the level two table must be duplicated according to the page size.

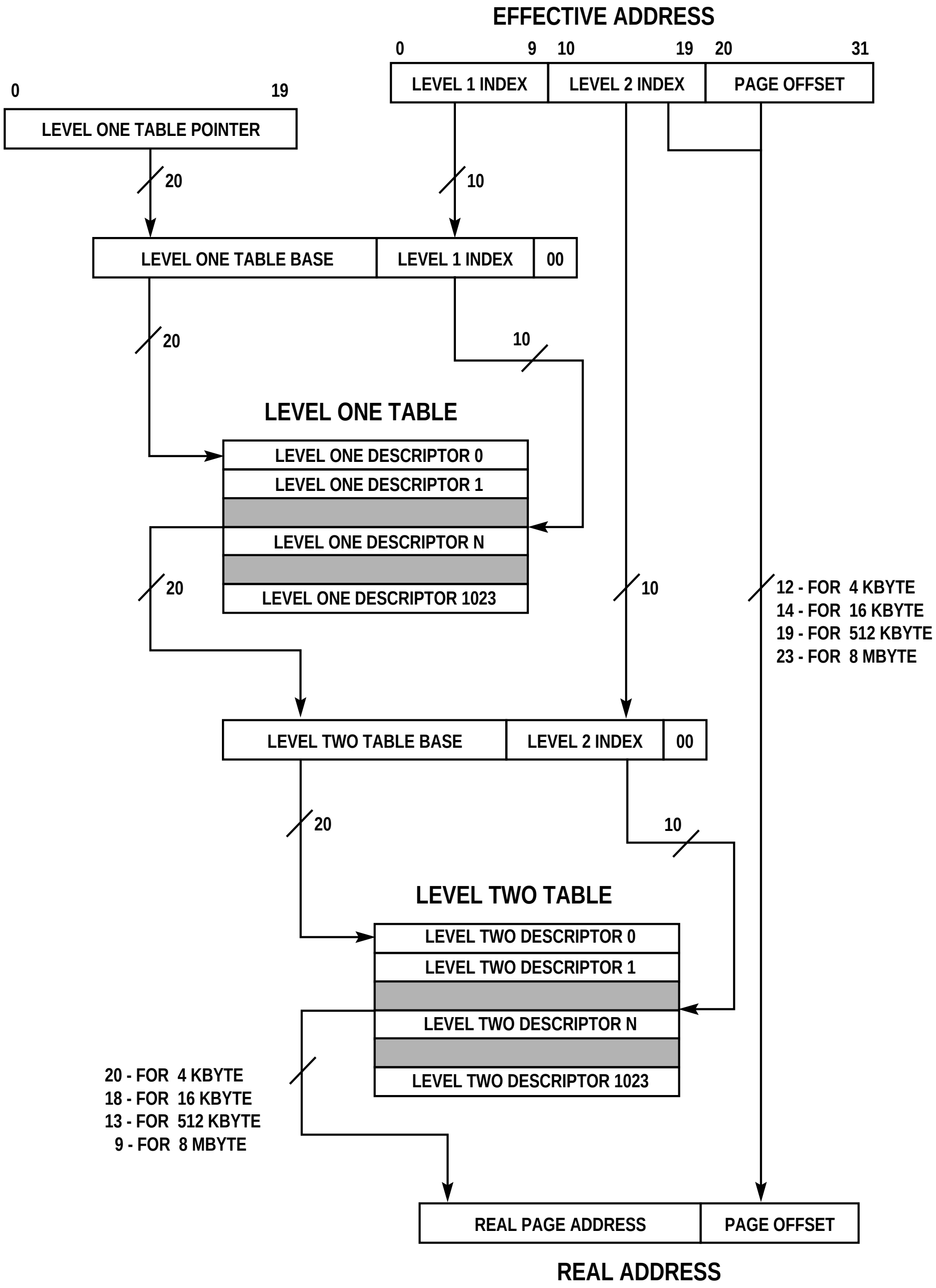


Figure 11-2. Two Level Translation Table When MD_CTR(TWAM) = 1

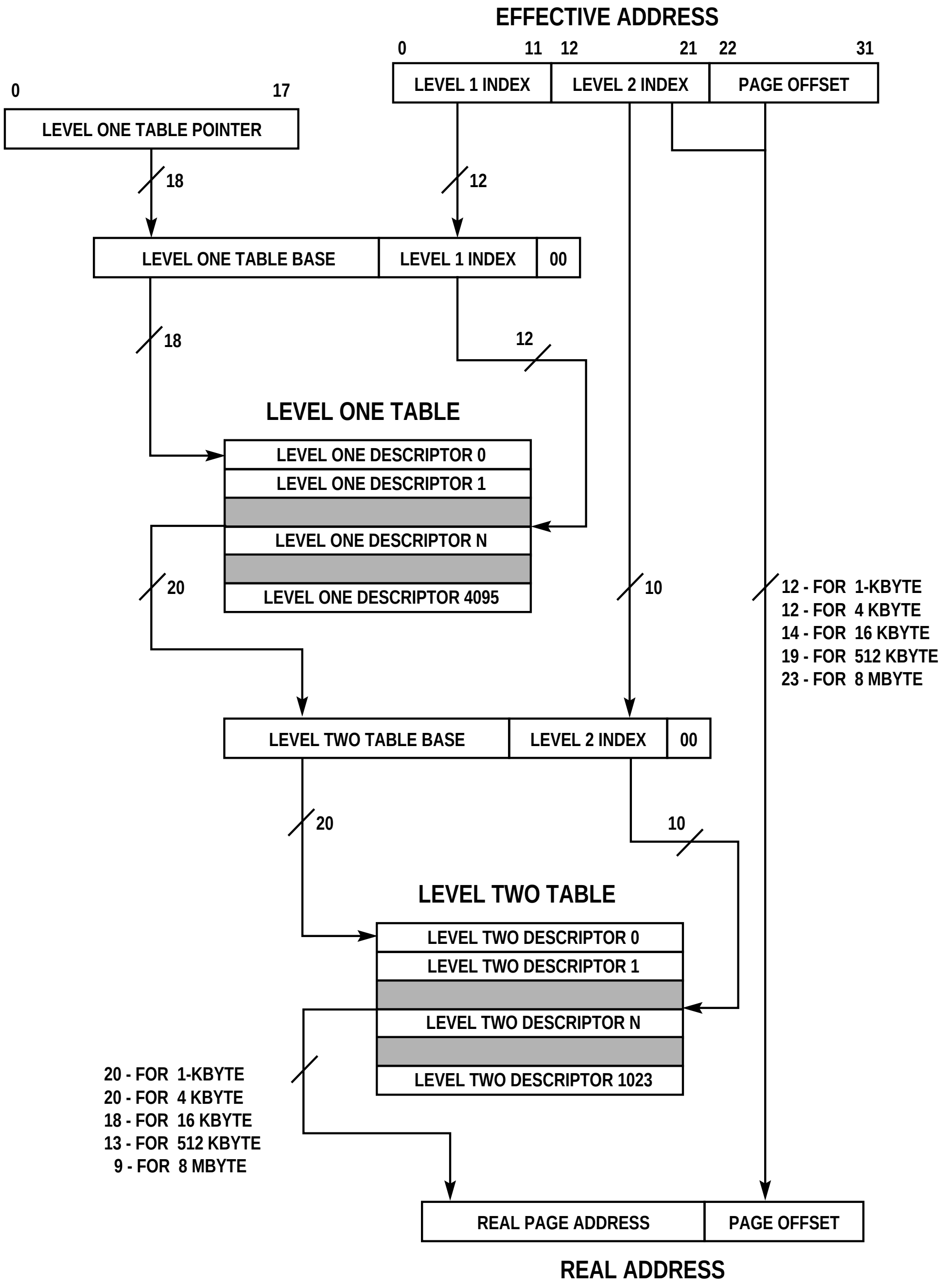


**Figure 11-3. Two Level Translation Table When MD_CTR(TWAM) = 0**

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

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

| PAGE SIZE | NUMBER OF REPLACED EFFECTIVE ADDRESS BITS |
|---|---|
| 1 kbyte | 20 |
| 4 kbyte | 20 |
| 16 kbyte | 18 |
| 512 kbyte | 13 |
| 8 Mbyte | 9 |

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

| PAGE SIZE | MD_CTR$_{TWAM}$ = 0 | MD_CTR$_{TWAM}$ = 1 |
|---|---|---|
| 1 kbyte | 1 | – |
| 4 kbyte | 1 | 1 |
| 16 kbyte | 1 | 1 |
| 512 kbyte | 1 | 1 |
| 8 Mbyte | 8 | 2 |

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

| PAGE SIZE | MD_CTR$_{TWAM}$ = 0 | MD_CTR$_{TWAM}$ = 1 |
|---|---|---|
| 1 kbyte | 1 | – |
| 4 kbyte | 4 | 1 |
| 16 kbyte | 16 | 4 |
| 512 kbyte | 512 | 128 |
| 8 Mbyte | 1,024 | 1,024 |

### 11.6.1 Level One Descriptor

The following table describes the level one descriptor format that is supported by the hardware to minimize the software tablewalk routine.

**Table 11-4. Level One (Segment) Descriptor Format**

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-17 | L2BA | Level 2 Table Base Address | |
| 18-19 | | | These Bits are Used Only When $MD\_CTR_{TWAM}$ = 1, Otherwise They Should Be ‘0’ |
| 20-22 | Reserved | | |
| 23-26 | APG | Access Protection Group | |
| 27 | G | Guarded Storage Attribute For Entry | 0 - Nonguarded Storage<br>1 - Guarded Storage |
| 28-29 | PS | Page Size Level One | 00 - Small (4 kbyte Or 16 kbyte)<br>01 - 512 kbyte<br>11 - 8 kbyte<br>10 - Reserved |
| 30 | WT | Writethrough Attribute For Entry | 0 - Copyback Cache Policy Region (Default)<br>1 - Writethrough Cache Policy Region |
| 31 | V | Segment Valid Bit | 0 - Segment is Not Valid<br>1 - Segment is Valid |

### 11.6.2 Level Two Descriptor

The following table describes the level two descriptor format that is supported by hardware to minimize the software tablewalk routine.

## Table 11-5. Level Two (Page) Descriptor Format

| BITS | MNEMONIC | DESCRIPTION | 4-KBYTE PAGES WITH 1-KBYTE RESOLUTION OF PROTECTION | 4-KBYTE RESOLUTION OF PROTECTION AND PAGES WITH SIZE LARGER THAN 4-KBYTE |
|---|---|---|---|---|
| 0-19 | RPN | Real Page Number | | |
| 20-21 | PP | Protection For the first 1-kbyte subpage in a 4-kbyte Page | For Instruction Pages<br>Privileged Problem<br>00 - No Access No Access<br>01 - Executable No Access<br>10 - Executable Executable<br>11 - Executable Executable<br>For Data Pages<br>Privileged Problem<br>00 - No Access No Access<br>01 - R/W No Access<br>10 - R/W R/O<br>11 - R/W R/W | For Instruction Pages<br>Privileged Problem<br>Extended Encoding:<br>00 - No Access No Access<br>01 - Executable No Access<br>10 - Reserved<br>11 - Reserved<br>PowerPC Encoding:<br>11 - Executable Executable<br>00 - Executable No Access<br>01 - Executable Executable<br>10 - Executable Executable<br>For Data Pages<br>Privileged Problem<br>Extended Encoding:<br>00 - No Access No Access<br>01 - R/O No Access<br>10 - Reserved<br>11 - Reserved<br>PowerPC Encoding:<br>11 - R/O R/O<br>00 - R/W No Access<br>01 - R/W R/O<br>10 - R/W R/W |

## Table 11-5. Level Two (Page) Descriptor Format (Continued)

<table>
<tr><th>BITS</th><th>MNEMONIC</th><th>DESCRIPTION</th><th>4-KBYTE PAGES WITH 1-KBYTE RESOLUTION OF PROTECTION</th><th colspan="2">4-KBYTE RESOLUTION OF PROTECTION AND PAGES WITH SIZE LARGER THAN 4-KBYTE</th></tr>
<tr><td>22</td><td rowspan="4">PP</td><td rowspan="2">Protection For a second 1-kbyte Subpage in a 4-kbyte Page</td><td rowspan="4">For Instruction Pages<br>Privileged Problem<br>00 - No Access No Access<br>01 - Executable No Access<br>10 - Executable Executable<br>11 - Executable Executable<br>For Data Pages<br>Privileged Problem<br>00 - No Access No Access<br>01 - R/W No Access<br>10 - R/W R/O<br>11 - R/W R/W</td><td colspan="2">0 - Bits 20-21 Contains PowerPC Encoding<br>1 - Bits 20-21 Contains Extended Encoding</td></tr>
<tr><td>23</td><td colspan="2">C - Change Bit For Entry<br>0 - Not Changed Region (Write-protected)<br>1 - Changed Region, Write Allowed</td></tr>
<tr><td>24-25</td><td>Protection For a third 1-kbyte subpage in a 4-kbyte Page</td><td rowspan="2">MD_CTR(PPCS)=0<br>0 - Subpage is Not Valid<br>1 - Subpage is Valid<br>If the Page Size is Larger Than 4-kbyte, Then All the 4 Bits Should Have the Same Value.</td><td rowspan="2">MD_CTR(PPCS)=1<br>1000 - Hit Only For Privileged Accesses<br>0100 - Hit Only For Problem Accesses<br>1100 - Hit For Both</td></tr>
<tr><td>26-27</td><td>Protection For a fourth 1-kbyte subpage in a 4-kbyte Page</td></tr>
<tr><td>28</td><td>SPS</td><td>Small Page Size</td><td>Should Be “0”</td><td colspan="2">0 - 4 kbyte<br>1 - 16 kbyte</td></tr>
<tr><td>29</td><td>SH</td><td>Shared Page</td><td colspan="3">0 - This Entry Matches Only if the ASID Filed in the TLB Entry Matches the Value of the M_CASID Register.<br>1 - ASID Comparison is Disabled For the Entry.</td></tr>
<tr><td>30</td><td>CI</td><td>Cache Inhibit</td><td colspan="3">Cache-inhibit Attribute For the Entry.</td></tr>
<tr><td>31</td><td>V</td><td>Valid Bit</td><td colspan="3">Page Valid Bit</td></tr>
</table>

## 11.7 PROGRAMMING MODEL

All programming model registers are special purpose registers accessed via the PowerPC mtspr/mfspr instructions. In addition, the PowerPC tlbie and tlbia architecture instructions are supported. MMU registers should be accessed when both $MSR_{IR}$ =0 and $MSR_{DR}$ =0. No similar restriction exists for the tlbie and tlbia instructions. The following registers control the TLB and the Mx_DCAM, Mx_DRAM0, and Mx_DRAM1 are used to effectively read it:

| | |
|---|---|
| **MI_CTR** | Instruction MMU Control Register |
| **MD_CTR** | Data MMU Control Register |
| **M_CASID** | CASID Register |
| **MI_EPN** | Instruction MMU Effective Number Register |
| **MI_TWC** | Instruction MMU Tablewalk Control Register |
| **MI_RPN** | Instruction MMU Real Page Number Port |
| **MD_EPN** | Data MMU Effective Number Register |
| **M_TWB** | MMU Tablewalk Base Register |
| **MD_TWC** | Data MMU Tablewalk Control Register |
| **MD_RPN** | Data MMU Real Page Number Port |
| **MI_AP** | Instruction MMU Access Protection Register |
| **MD_AP** | Data MMU Access Protection Register |
| **M_TW** | MMU Tablewalk Special Register |
| **MI_DCAM** | Instruction MMU CAM Entry Read Register |
| **MI_DRAM0** | Instruction MMU RAM Entry Read Register 0 |
| **MI_DRAM1** | Instruction MMU RAM Entry Read Register 1 |
| **MD_DCAM** | Data MMU CAM Entry Read Register |
| **MD_DRAM0** | Data MMU RAM Entry Read Register 0 |
| **MD_DRAM1** | Data MMU RAM Entry Read Register 1 |

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

### Table 11-6. MI_CTR Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0 | GPM | Group Protection Mode | 0 - PowerPC Mode<br>1 - Domain Manager Mode |
| 1 | PPM | Page Protection Mode | 0 - Page Resolution of Protection<br>1 - 1-kbyte Resolution of Protection For 4-kbyte Pages |
| 2 | CIDEF | CI Default | Default Value For Instruction Cache-Inhibit Attribute When the I-MMU is disabled ($MSR_{IR}$ = 0) |
| 3 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 4 | RSV4I | Reserve 4 Instruction TLB Entries | 0 - ITLB_INDX Decremented Modulo 32<br>1 - ITLB_INDX Decremented Modulo 28 |
| 5 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 6 | PPCS | Privilege/Problem State Compare Mode | 0 - Ignore Problem/Privileged State During Address Compare<br>1 - Take Into Account Problem/Privileged State According to MI_RPN(24:27) |
| 7-18 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 19-23 | ITLB_INDX | Instruction TLB Index | Pointer to the Instruction TLB Entry to be Loaded. Automatically Decremented Every Instruction TLB Update |
| 24-31 | Reserved | — | Ignore On Write<br>Returns “0” On Read |

NOTE: The reset value of the MI_CTR register is 0x00000000.

## Table 11-7. MD_CTR Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0 | GPM | Group Protection Mode | 0 - PowerPC Mode<br>1 - Domain Manager Mode |
| 1 | PPM | Page Protection Mode | 0 - Page Resolution of Protection<br>1 - 1-kbyte Resolution of Protection For 4-kbyte Pages |
| 2 | CIDEF | CI Default | Default Values For Data Cache Attributes When the D-MMU is Disabled ($MSR_{DR}$ = 0) |
| 3 | WTDEF | WT Default | |
| 4 | RSV4D | Reserve 4 Data TLB Entries | 0 - DTLB_INDX Decremented Modulo 32<br>1 - DTLB_INDX Decremented Modulo 28 |
| 5 | TWAM | Table Walk Assist Mode | 0 - 1-kbyte subpage Hardware Assist<br>1 - 4-kbyte Page Hardware Assist |
| 6 | PPCS | Privilege/Problem State Compare Mode | 0 - Ignore Problem/Privileged State During Address Compare<br>1 - Take Into Account Problem/Privileged State According to MD_RPN(24:27) |
| 7-18 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 19-23 | DTLB_INDX | Data TLB Index | Pointer to the Data TLB Entry to be Loaded<br>Automatically Decremented Every Data TLB Update |
| 24-31 | Reserved | — | Ignore On Write<br>Returns “0” On Read |

NOTE: The reset value of the MD_CTR register is 0x04000000.

## Table 11-8. M_CASID Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-27 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 28-31 | CASID | Current Address Space ID | This Field is Compared With the ASID Field of a TLB Entry to Qualify a Match |

NOTE: The reset value of the M_CASID register is undefined.

## Table 11-9. MI_EPN Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-19 | EPN | Effective Page Number For TLB Entry | Default Value is the Effective Address of the Last Instruction TLB Miss |
| 20-21 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 22 | EV | TLB Entry Valid Bit | 0 - The TLB Entry is Invalid<br>1 - The TLB Entry is Valid<br>This Bit is Set to “1” On Each Instruction TLB Miss |
| 23-27 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 28-31 | ASID | Address Space ID | Address Space ID of the Instruction TLB Entry to be Compared With the CASID Field in the M_CASID Register |

NOTE: The reset value of the MI_EPN register is undefined.

## Table 11-10. MI_TWC Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0 -22 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 23-26 | APG | Access Protection Group | Up to 16 Protection Groups Supported.<br>Default Value On Instruction TLB Miss Is “0’ |
| 27 | G | Guarded Storage Attribute For Entry | 0- Nonguarded Storage<br>1- Guarded Storage<br>Default Value On Instruction TLB Miss Is “0” |
| 28-29 | PS | Page Size Level One | 00 - Small (4 kbyte Or 16 kbyte - see MI_RPN)<br>01 - 512 kbyte<br>11 - 8 Mbyte<br>10 - Reserved<br>Default Value On Instruction TLB Miss Is “00” |
| 30 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 31 | V | Entry Valid Bit | 0 - Entry is Not Valid<br>1 - Entry is Valid<br>Default Value On Instruction TLB Miss Is “1” |

NOTE: The reset value of the MI_TWC register is undefined.

## Table 11-11. Level MI_RPN Register

| BITS | MNEMONIC | DESCRIPTION AND FUNCTION FOR 4-KBYTE PAGES WITH 1-KBYTE RESOLUTION OF PROTECTION | DESCRIPTION AND FUNCTION FOR 4-KBYTE RESOLUTION OF PROTECTION AND PAGES WITH SIZE LARGER THAN 4 KBYTE | |
|---|---|---|---|---|
| 0-19 | RPN | Real Page Number | Real Page Number | |
| 20-21 | PP | Protection Attributes For the 1st, 2nd, 3rd, and 4th Subpages in a 4 kbyte Page.<br>Privileged / Problem<br>00 - No Access / No Access<br>01 - Executable / No Access<br>10 - Executable / Executable<br>11 - Executable / Executable | Privileged / Problem<br>Extended Encoding:<br>00 - No Access / No Access<br>01 - Executable / No Access<br>10 - Reserved / Reserved<br>11 - Reserved / Reserved<br>PowerPC Encoding:<br>11 - Executable / Executable<br>00 - Executable / No Access<br>01 - Executable / Executable<br>10 - Executable / Executable | |
| 22 | | | 0 - Bits 20-21 Contains PowerPC Encoding<br>1 - Bits 20-21 Contains Extended Encoding | |
| 23 | | | Reserved | |
| 24-25 | | | MD_CTR(PPCS)=0 | MD_CTR(PPCS)=1 |
| 26-27 | | | 0 - Subpage is Not Valid<br>1 - Subpage is Valid<br>If the Page Size is Larger Than 4 kbyte, Then All the 4 Bits Should Have the Same Value. | 1000 - Hit Only For Privileged Accesses<br>0100 - Hit Only For Problem Accesses<br>1100 - Hit For Both |
| 28 | LPS | Large Page Size: Should Be “0”. | 0 - 4 kbyte<br>1 - 16 kbyte | |

### Table 11-11. Level MI_RPN Register (Continued)

| BITS | MNEMONIC | DESCRIPTION AND FUNCTION FOR 4-KBYTE PAGES WITH 1-KBYTE RESOLUTION OF PROTECTION | DESCRIPTION AND FUNCTION FOR 4-KBYTE RESOLUTION OF PROTECTION AND PAGES WITH SIZE LARGER THAN 4 KBYTE |
|---|---|---|---|
| 29 | SH | Shared Page:<br>0 - This Entry Matches Only if the ASID Filed in the TLB Entry Matches the Value of the M_CASID Register.<br>1 - ASID Comparison is Disabled For the Entry. | |
| 30 | CI | Cache-inhibit Attribute For the Entry. | |
| 31 | V | Entry Valid Indication. | |

NOTE: The reset value of the MI_RPN register is undefined.

### Table 11-12. MD_EPN Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-21 | EPN | Effective Page Number For Entry | Default Value is the Effective Address of the Last Data TLB Miss |
| 22 | EV | TLB Entry Valid Bit | 0 - The Data TLB Entry is Invalid<br>1 - The Data TLB Entry is Valid<br>This Bit is Set to “1” on the Data TLB Miss |
| 23-27 | Reserved | — | Ignore On Write<br>Returns “0” On Read |
| 28-31 | ASID | Address Space ID | Address Space ID of the Data TLB Entry to be Compared With the CASID Field in the M_CASID Register |

NOTE: The reset value of the MD_EPN register is undefined.

### Table 11-13. M_TWB Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-19 | L1TB | Tablewalk Level One Base Value | Tablewalk Level One Base Value |
| 20-29 | L1INDX | Level One Table Index | Ignore On Write<br>Returns MD_EPN[0:9] On Read When MD_CTR$_{TWAM}$ = 1<br>Returns MD_EPN[2:11] On Read When MD_CTR$_{TWAM}$ = 0 |
| 30-31 | Reserved | — | Ignore On Write<br>Returns “0” On Read |

NOTE: The reset value of the M_TWB register is undefined.

## Table 11-14. MD_TWC Register

| BITS | MNEMONIC | | DESCRIPTION AND FUNCTION ON WRITE | DESCRIPTION AND FUNCTION ON READ |
|---|---|---|---|---|
| | ON WRITE | ON READ | | |
| 0 -19 | L2TB | L2TB | Tablewalk Level Two Table Base Value | Tablewalk Level Two Table Base Value |
| 20 -22 | Reserved | L2INDX | Ignore | Level Two Table Index:<br>Returns MD_EPN[10:19] When $\text{MDCTR}_{\text{TWAM}}$ = 1<br>Returns MD_EPN[12:21] When $\text{MDCTR}_{\text{TWAM}}$ = 0 |
| 23-26 | APG | | Access Protection Group - Up to 16 Protection Groups Are Supported These Bits Are Set to “0000” On the Data TLB Miss | |
| 27 | G | | Guarded Storage Attribute of the Entry:<br>0 - Nonguarded Storage<br>1 - Guarded Storage<br>This Bit is Set to “0” On the Data TLB Miss | |
| 28-29 | PS | | Level One Page Size:<br>00 - Small (4 kbyte Or 16 kbyte - See MD_RPN)<br>01 - 512 kbyte<br>11 - 8 Mbyte<br>10 - Reserved<br>This Bit is Set to “0” On the Data TLB Miss | |
| 30 | WT | “0” | Writethrough Attribute For Entry:<br>0 - Copyback Data Cache Policy Page Entry<br>1 - Writethrough Data Cache Policy Page Entry<br>This Bit is Set to “0” On the Data TLB Miss | Returns “0” On Read |
| 31 | V | “0” | 0 - Entry is not valid<br>1 - Entry is valid<br>This Bit is Set to “1” On the Data TLB Miss | Returns “0” On Read |

NOTE: The reset value of the MD_TWC register is undefined.

### Table 11-15. MD_RPN Register

| BITS | MNEMONIC | DESCRIPTION AND FUNCTION FOR 4 KBYTE PAGES WITH 1-KBYTE RESOLUTION OF PROTECTION | DESCRIPTION AND FUNCTION FOR 4 KBYTE RESOLUTION OF PROTECTION AND PAGES WITH SIZE LARGER THAN 4 KBYTE | |
|---|---|---|---|---|
| 0-19 | RPN | Real Page Number | Real Page Number | |
| 20-21 | PP | Protection Attributes For the 1st, 2nd, 3rd, and 4th Subpages in a 4 kbyte Page.<br>Privileged Problem<br>00 - No Access No Access<br>01 - R/W No Access<br>10 - R/W R/O<br>11 - R/W R/W | Privileged Problem<br>Extended Encoding:<br>00 - No Access No Access<br>01 - R/O No Access<br>10 - Reserved<br>11 - Reserved<br>PowerPC Encoding:<br>11 - R/O R/O<br>00 - R/W No Access<br>01 - R/W R/O<br>10 - R/W R/W | |
| 22 | | | 0 - Bits 20-21 Contains PowerPC Encoding<br>1 - Bits 20-21 Contains Extended Encoding | |
| 23 | | | Change Bit For Data TLB Entry:<br>0 - Unchanged Region. Write Access to This Page Results in the Implementation Specific IMMU Interrupt Invocation. Software Should Take An Appropriate Action Before Setting This Bit to 1.<br>1 - Changed Region. Write Access is Allowed to This Page. | |
| 24-25 | | | MD_CTR(PPCS)=0 | MD_CTR(PPCS)=1 |
| 26-27 | | | 0 - Subpage is Not Valid<br>1 - Subpage is Valid<br>If the Page Size is Larger Than 4 kbyte, Then All the 4 Bits Should Have the Same Value. | 1000 - Hit Only For Privileged Accesses<br>0100 - Hit Only For Problem Accesses<br>1100 - Hit For Both |

**Table 11-15. MD_RPN Register (Continued)**

| BITS | MNEMONIC | DESCRIPTION AND FUNCTION FOR 4 KBYTE PAGES WITH 1-KBYTE RESOLUTION OF PROTECTION | DESCRIPTION AND FUNCTION FOR 4 KBYTE RESOLUTION OF PROTECTION AND PAGES WITH SIZE LARGER THAN 4 KBYTE |
|---|---|---|---|
| 28 | LPS | Large Page Size: Should Be “0”. | 0 - 4 kbyte<br>1 - 16 kbyte |
| 29 | SH | Shared Page:<br>0 - This Entry Matches Only if the ASID Filed in the Data TLB Entry Matches the Value of the M_CASID Register.<br>1 - ASID Comparison is Disabled For the Entry. | |
| 30 | CI | Cache-inhibit Attribute For the Entry. | |
| 31 | V | Entry Valid Indication. | |

NOTE: The reset value of the MD_RPN register is undefined.

## Table 11-16. MI_AP Register

| BITS | MNEMONIC | DOMAIN MANAGER MODE | PowerPC MODE |
|---|---|---|---|
| 0-1 | GP | GP<br>00 - No Access<br>01 - Client–Access Permission Defined By Page Protection Bits<br>10 - Reserved<br>11 - Manager–Free Access | GP = Ks and Kp From the PowerPC Books:<br>00 - All Accesses Are Treated As Privileged<br>01 - Access Permission Defined By Page Protection Bits<br>10 - Problem and Privileged Interpretation is Swapped<br>11 - All Accesses Are Treated As Problem |
| 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 | | | |

NOTE: The reset value of the MI_AP register is undefined.

### Table 11-17. MD_AP Register

| BITS | MNEMONIC | DOMAIN MANAGER MODE | PowerPC MODE |
|---|---|---|---|
| 0-1 | GP | GP<br>00 - No Access<br>01 - Client–Access Permission Defined By Page Protection Bits<br>10 - Reserved<br>11 - Manager–Free Access | GP = Ks and Kp From the PowerPC Books:<br>00 - All Accesses Are Treated As Privileged<br>01 - Access Permission Defined By Page Protection Bits<br>10 - Problem and Privileged Interpretation is Swapped<br>11 - All Accesses Are Treated As Problem |
| 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 | | | |

NOTE: The reset value of the MD_AP register is undefined.

### Table 11-18. M_TW Register

| BITS | MNEMONIC | DESCRIPTION |
|---|---|---|
| 0-31 | M_TW | This Register is Used As a Scratch Register in the Software Tablewalk Interrupt Handlers |

NOTE: The reset value of the M_TW register is undefined.

The MD_DCAM, MD_DRAM0, and MD_DRAM1 registers are interface registers that enable to read data MMU CAM and RAM entries. An attempt to write to the MD_DCAM register using the mtspr instruction will load the CAM and RAM values of the entry pointed by DTLB_INDX to MD_DCAM, MD_DRAM0, and MD_DRAM1. The source register in the mtspr instruction can be any register, since its value is not used. The values of the MD_DCAM, MD_DRAM0, and MD_DRAM1 registers can be read using the mtspr instruction. Any try to write to the MD_DRAM0 and MD_DRAM1 registers using the mtspr instruction will be considered as an NOP.

### Table 11-19. MD_DBCAM Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-19 | EPN | Effective Page Number | |
| 20 | SPVF | Subpage Validity Flags | 0 - Subpage 0 (Address[20:21]=00) is Not Valid<br>1 - Subpage 0 (Address[20:21]=00) is Valid |
| 21 | | | 0 - Subpage 1 (Address[20:21]=01) is Not Valid<br>1 - Subpage 1 (Address[20:21]=01) is Valid |
| 22 | | | 0 - Subpage 2 (Address[20:21]=10) is Not Valid<br>1 - Subpage 2 (Address[20:21]=10) is Valid |
| 23 | | | 0 - Subpage 3 (Address[20:21]=11) is Not Valid<br>1 - Subpage 3 (Address[20:21]=11) is Valid |
| 24-26 | PS | Page Size | 000 - 4 kbyte<br>001 - 16 kbyte<br>011 - 512 kbyte<br>111 - 8 Mbyte<br>010 - Reserved<br>100 - Reserved<br>101 - Reserved<br>110 - Reserved |
| 27 | SH | Shared Page | 0 - This Entry Matches Only if the ASID Field in the Data TLB Entry Matches the Value of the M_CASID Register<br>1 - ASID Comparison is Disabled For the Entry |
| 28-31 | ASID | Address Space ID | Address Space ID of the Data TLB Entry to Be Compared With the CASID Field in the M_CASID Register |

NOTE: The reset value of the MD_DBCAM register is undefined.

### Table 11-20. MD_DBRAM0 Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-19 | RPN | Real Page Number | |
| 20-22 | PS | Page Size | 000 - 4 kbyte<br>001 - 16 kbyte<br>011 - 512 kbyte<br>111 - 8 Mbyte<br>010 - Reserved<br>100 - Reserved<br>101 - Reserved<br>110 - Reserved |
| 23-26 | APGI | Access Protection Group Inverted | Access Protection Group Number Represented in One's Complement Format |
| 27 | G | Guarded Storage Attribute For the Entry | 0 - Nonguarded Storage<br>1 - Guarded Storage |
| 28 | WT | Writethrough Attribute For the Entry | 0 - Copyback Data Cache Policy Page Entry<br>1 - WriteThrough Data Cache Policy Page Entry |
| 29 | CI | Cache-Iinhibit Attribute For the Entry | |
| 30-31 | | Reserved | |

NOTE: The reset value of the MD_DBRAM0 register is undefined.

### Table 11-21. MD_DBRAM1 Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-16 | | Reserved | |
| 17 | C | Change Bit For Data TLB Entry | 0 - Unchanged Region. Write Access to This Page Results in the Implementation Specific IMMU Interrupt Invocation. Software Should Take An Appropriate Action Before Setting This Bit to 1.<br>1 - Changed Region. Write Access is Allowed to This Page. |
| 18 | EVF | Entry Valid Flag | 0 - Entry is Not Valid<br>1 - Entry is Valid |
| 19 | SA | Privileged (Supervisor) Access | 0 - Subpage 0 (Address[20:21]=00) Privileged Access is Not Permitted<br>1 - Subpage 0 (Address[20:21]=00) Privileged Access is Permitted |
| 20 | | | 0 - Subpage 1 (Address[20:21]=01) Privileged Access is Not Permitted<br>1 - Subpage 1 (Address[20:21]=01) Privileged Access is Permitted |

### Table 11-21. MD_DBRAM1 Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 21 | SA | Privileged (Supervisor) Access | 0 - Subpage 2 (Address[20:21]=10) Privileged Access is Not Permitted<br>1 - Subpage 2 (Address[20:21]=10) Privileged Access is Permitted |
| 22 | | | 0 - Subpage 3 (Address[20:21]=11) Privileged Access is Not Permitted<br>1 - Subpage 3 (Address[20:21]=11) Privileged Access is Permitted |
| 23 | SAT | Privileged (Supervisor) Access Type | 0 - Privileged Access Type is Read Only<br>1 - Privileged Access Type is Read/Write |
| 24 | URP0 | Problem (User) Read Permission Page Zero | 0 - Subpage 0 (Address[20:21]=00) Problem Read Access is Not Permitted<br>1 - Subpage 0 (Address[20:21]=00) Problem Read Access is Permitted |
| 25 | UWP0 | Problem (User) Write Permission Page Zero | 0 - Subpage 0 (Address[20:21]=00) Problem Write Access is Not Permitted<br>1 - Subpage 0 (Address[20:21]=00) Problem Write Access is Permitted |
| 26 | URP1 | Problem (User) Read Permission Page One | 0 - Subpage 1 (Address[20:21]=01) Problem Read Access is Not Permitted<br>1 - Subpage 1 (Address[20:21]=01) Problem Read Access is Permitted |
| 27 | UWP1 | Problem (User) Write Permission Page One | 0 - Subpage 1 (Address[20:21]=01) Problem Write Access is Not Permitted<br>1 - Subpage 1 (Address[20:21]=01) Problem Write Access is Permitted |
| 28 | URP2 | Problem (User) Read Permission Page Two | 0 - Subpage 2 (Address[20:21]=10) Problem Read Access is Not Permitted<br>1 - Subpage 2 (Address[20:21]=10) Problem Read Access is Permitted |
| 29 | UWP2 | Problem (User) Write Permission Page Two | 0 - Subpage 2 (Address[20:21]=10) Problem Write Access is Not Permitted<br>1 - Subpage 2 (Address[20:21]=10) Problem Write Access is Permitted |

### Table 11-21. MD_DBRAM1 Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 30 | URP3 | Problem (User) Read Permission Page Three | 0 - Subpage 3 (Address[20:21]=11) Problem Read Access is Not Permitted<br>1 - Subpage 3 (Address[20:21]=11) Problem Read Access is Permitted |
| 31 | UWP3 | Problem (User) Write Permission Page Three | 0 - Subpage 3 (Address[20:21]=11) Problem Write Access is Not Permitted<br>1 - Subpage 3 (Address[20:21]=11) Problem Write Access is Permitted |

NOTE: The reset value of the MD_DBRAM1 register is undefined.

### Table 11-22. MI_DBCAM Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-19 | EPN | Effective Page Number | |
| 20-22 | PS | Page Size | 000 - 4 kbyte<br>001 - 16 kbyte<br>011 - 512 kbyte<br>111 - 8 Mbyte<br>010 - Reserved<br>100 - Reserved<br>101 - Reserved<br>110 - Reserved |
| 23-26 | ASID | Address Space ID | Address Space ID of the Data TLB Entry to Be Compared With the CASID Field in the M_CASID Register |
| 27 | SH | Shared Page | 0 - This Entry Matches Only if the ASID Field in the Data TLB Entry Matches the Value of the M_CASID Register<br>1 - ASID Comparison is Disabled For the Entry |
| 28 | SPV | Subpage Validity | 0 - Subpage 0 (Address[20:21]=00) is Not Valid<br>1 - Subpage 0 (Address[20:21]=00) is Valid |
| 29 | | | 0 - Subpage 1 (Address[20:21]=01) is Not Valid<br>1 - Subpage 1 (Address[20:21]=01) is Valid |
| 30 | | | 0 - Subpage 2 (Address[20:21]=10) is Not Valid<br>1 - Subpage 2 (Address[20:21]=10) is Valid |
| 31 | | | 0 - Subpage 3 (Address[20:21]=11) is Not Valid<br>1 - Subpage 3 (Address[20:21]=11) is Valid |

NOTE: The reset value of the MI_DBCAM register is undefined.

Table 11-23. MI_DBRAM0 Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-19 | RPN | Real Page Number | |
| 20-22 | PS_B | Page Size | 000 - 4 kbyte<br>001 - 16 kbyte<br>011 - 512 kbyte<br>111 - 8 Mbyte<br>010 - Reserved<br>100 - Reserved<br>101 - Reserved<br>110 - Reserved |
| 23 | CI | Cache-Inhibit Attribute For the Entry | |
| 24-27 | APG | Access Protection Group | Up to 16 Protection Groups Are Supported (Represented in One's Compliment Format) |
| 28 | SFP | Privileged (Supervisor) Fetch Permission | 0 - Subpage 0 (Address[20:21]=00) Privileged Fetch is Not Permitted<br>1 - Subpage 0 (Address[20:21]=00) Privileged Fetch is Permitted |
| 29 | | | 0 - Subpage 1 (Address[20:21]=01) Privileged Fetch is Not Permitted<br>1 - Subpage 1 (Address[20:21]=01) Privileged Fetch is Permitted |
| 30 | | | 0 - Subpage 2 (Address[20:21]=10) Privileged Fetch is Not Permitted<br>1 - Subpage 2 (Address[20:21]=10) Privileged Fetch is Permitted |
| 31 | | | 0 - Subpage 3 (Address[20:21]=11) Privileged Fetch is Not Permitted<br>1 - Subpage 3 (Address[20:21]=11) Privileged Fetch is Permitted |

NOTE: The reset value of the MI_DBRAM0 register is undefined.

The MI_DCAM, MI_DRAM0, and MI_DRAM1 registers are interface registers that enable to read data MMU CAM and RAM entries. An attempt to write to the MI_DCAM register using the mtspr instruction will load the CAM and RAM values of the entry pointed by DTLB_INDX to MI_DCAM, MI_DRAM0, and MI_DRAM1. The source register in the mtspr instruction can be any register, since its value is not used. The values of the MI_DCAM, MI_DRAM0, and MI_DRAM1 registers can be read using the mtspr instruction. Any try to write to the MI_DRAM0 and MI_DRAM1 registers using the mtspr instruction will be considered as an NOP.

Table 11-24. MI_DBRAM1 Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-25 | | Reserved | |
| 26 | UFP | Problem (User) Fetch Permission | 0 - Subpage 0 (Address[20:21]=00)<br>Problem Fetch is Not Permitted<br>1 - Subpage 0 (Address[20:21]=00)<br>Problem Fetch is Permitted |
| 27 | | | 0 - Subpage 1 (Address[20:21]=01)<br>Problem Fetch is Not Permitted<br>1 - Subpage 1 (Address[20:21]=01)<br>Problem Fetch is Permitted |
| 28 | UFP | Problem (User) Fetch Permission | 0 - Subpage 2 (Address[20:21]=10)<br>Problem Fetch is Not Permitted<br>1 - Subpage 2 (Address[20:21]=10)<br>Problem Fetch is Permitted |
| 29 | | | 0 - Subpage 3 (Address[20:21]=11)<br>Problem Fetch is Not Permitted<br>1 - Subpage 3 (Address[20:21]=11)<br>Problem Fetch is Permitted |
| 30 | PV | Page Validity | 0 - Page is Not Valid<br>1 - Page is Valid |
| 31 | G | Guarded Storage Attribute For Entry | 0 - Nonguarded Storage<br>1 - Guarded Storage |

NOTE: The reset value of the MI_DRAM1 register is undefined.

## 11.8 MEMORY MANAGEMENT UNIT INTERRUPTS

### 11.8.1 Implementation Specific Instruction TLB Miss Interrupt

The implementation specific instruction TLB miss interrupt occurs when $MSR_{IR}$ =1 and there is an attempt to fetch an instruction from a page whose effective page number can not be translated by the instruction TLB. The software tablewalk code is responsible for loading the translation information of the missed page from the translation table that resides in the memory. Refer to **Section 11.9.1.1 Translation Reload Examples** for more information.

### 11.8.2 Implementation Specific Data TLB Miss Interrupt

The implementation specific data TLB miss interrupt occurs when $MSR_{DR}$ =1 and there is an attempt to access a page whose effective page number cannot be translated by the data TLB. The software tablewalk code is responsible for loading the translation information of the missed page from the translation table that resides in the memory. Refer to **Section 11.9.1.1 Translation Reload Examples** for more information.

### 11.8.3 Implementation Specific Instruction TLB Error Interrupt

The implementation specific instruction TLB error interrupt occurs in the following cases:

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

The exact reason for invocation of the instruction TLB error interrupt handler can be found in the SRR1 register. For bit assignments refer to **Section 7.3.7.3.12 Implementation Specific Instruction TLB Error Interrupt**. If needed, it is the software's responsibility to invoke the instruction storage interrupt handler.

### 11.8.4 Implementation Specific Data TLB Error Interrupt

The implementation specific data TLB error interrupt occurs in the following cases:

- The effective address of a load, store, icbi, dcbz, dcbst, dcbf, or dcbi instruction cannot be translated. Either the segment or page valid bit of this page are cleared in the translation table.
- The access violates the storage protection.
- An attempt is made to write to a page with the negated change bit.

The DSISR explains invocation of the data TLB error interrupt handler. For bit assignments refer to **Section 7.3.7.3.14 Implementation Specific Data TLB Error Interrupt**. If needed, it is the software's responsibility to invoke the data storage interrupt handler.

## 11.9 TLB MANIPULATION

### 11.9.1 TLB Reload

The TLB reload (tablewalk) function is performed in the software with some hardware assistance that consists of:

- Automatic storage of the missed effective data or instruction address and default attributes in the MI_EPN or MD_EPN registers, respectively. This value is loaded into the selected entry on a write to MI_RPN or MD_RPN for the instruction TLB and data TLB.
- Automatic updating of the replacement location counter to point to the entry to be replaced. This value is placed in the index field in the MI_CTR and MD_CTR registers.
- The level one pointer is generated when a mfspr Rx, M_TWB is performed by the concatenation of the level one table base with the level one index. Refer to Figure 11-2 and Figure 11-3 for details.
- The level two pointer is generated when a mfspr Rx, MD_TWC is performed by the concatenation of the level two table base (extracted from the level one table) with the level two index.

- A write to the TLB entry is performed by loading the tablewalk level two entry value to the MI_RPN or MD_RPN register.
- A special register, M_TW, is available for the software tablewalk routine, in addition to the architecture's four OS special registers—SPRG0- SPRG3. Thus, allowing the miss code to save enough general-purpose registers so it can execute without corrupting the state of any of the existing general-purpose registers.

**11.9.1.1 TRANSLATION RELOAD EXAMPLES.** The following are code examples for generating the real page number using a two-level tree page table structure. The first example is of the data TLB reload and the second one is of the instruction TLB reload. Notice that the following assumptions are made:

1. M_TWB holds the base pointer to the first level table.
2. Both instruction and data address translation is turned off (MSR$_{IR}$ =0 and MSR$_{DR}$ =0).

```
dtlb_swtw       mtspr    M_TW, R1         # save R1
                mfspr    R1, M_TWB        # load R1 with level one pointer
                lwz      R1, (R1)         # Load level one page entry
                mtspr    MD_TWC,R1        # save level two base pointer and
                                          # level one attributes
                mfspr    R1, MD_TWC       # load R1 with level two pointer
                                          # while taking into account the
                                          # page size
                lwz      R1, (R1)         # Load level two page entry
                mtspr    MD_RPN, R1       # Write TLB entry
                mfspr    R1, M_TW         # restore R1
                rfi



itlb_swtw       mtspr    M_TW, R1         # save R1
                mfspr    R1, SRR0         # load R1 with instruction miss
                                          # effective address (the same data
                                          # may be taken from the MI_EPN
                                          # register)
                mtspr    MD_EPN, R1       # save instruction miss effective
                                          # address in MD_EPN
                mfspr    R1, M_TWB        # load R1 with level one pointer
                lwz      R1, (R1)         # Load level one page entry
                mtspr    MI_TWC,R1        # save level one attributes
                mtspr    MD_TWC,R1        # save level two base pointer
                mfspr    R1, MD_TWC       # load R1 with level two pointer
                                          # while taking into account the
                                          # page size
```

```
lwz       R1, (R1)         # Load level two page entry
mtspr     MI_RPN, R1       # Write TLB entry
mfspr     R1, M_TW         # restore R1
rfi
```

### 11.9.2 TLB Replacement Counter

The TLB replacement counter can be selectively controlled to only select among the first 28 entries in each TLB by setting the $\overline{\text{RSV4I}}$ bit in the MI_CTR or $\overline{\text{RSV4D}}$ bit in the MD_CTR. Those control bits also affect the tlbia instruction as described later. Replacement counters are cleared to zero after the execution of the tlbia instruction and the counters decrement after an appropriate TLB reload (ITLB_INDX decrements after the ITLB reload and DTLB_INDX decrements after the DTLB reload).

### 11.9.3 TLB Invalidation

The MPC821 implements the tlbie instruction for the invalidation of TLB entries. This instruction invalidates TLB entries in the TLB that hit, including the reserved entries. Notice that the 22 most-significant bits of the effective address are used in the comparison since no segment registers are implemented. Although for entries with page sizes greater than 4 kbytes, some of the lower bits of the effective page number are ignored. The ASID value in the entry is ignored for the purpose of matching an invalidate address, thus multiple entries may be invalidated if they have the same effective address and different ASID values.

The MPC821 supports the tlbia instruction to invalidate all entries in both TLBs. If the $\overline{\text{RSV4D}}$ or $\overline{\text{RSV4I}}$ bit is set for a TLB, the four reserved entries will not be invalidated on execution of tlbia. However, the software can explicitly invalidate one or more of these entries by setting the index field in MD_CTR (DTLB_INDX) or MI_CTR (ITLB_INDX), negating the entry valid bit (EV) in MD_EPN or MI_EPN, and performing a write to the appropriate MD_RPN or MI_RPN. The TLBs are not automatically invalidated on reset although they are disabled. However, they must be invalidated under program control.

### 11.9.4 Loading the Reserved TLB Entries

The process of loading a single reserved entry in the TLB is as follows:

- Disable the TLB by clearing $MSR_{IR}$ or $MSR_{DR}$ as needed.
- Clear the $\overline{\text{RSV4I}}$ ($\overline{\text{RSV4D}}$) bit in the MI_CTR (MD_CTR).
- Invalidate the effective address of the reserved page by using the tlbia or tlbie instruction.
- Set the register ITLB_INDX (DTLB_INDX) fields of the MI_CTR (MD_CTR) to the appropriate value (between 27 and 31).
- Load the MI_EPN (MD_EPN) register with the effective page number, the ASID of the reserved page, and “1” as the EV bit.

- Run software tablewalk code to load the appropriate entry in the TLB. Refer to **Section 11.9.1.1 Translation Reload Examples** for examples of this code.
- If needed, repeat the three previous steps to load other TLB entries.
- Set the RSV4I ($\overline{\text{RSV4D}}$) bit in the MI_CTR (MD_CTR).

## 11.10 REQUIREMENTS FOR ACCESSING THE MMU CONTROL REGISTERS

All instruction and data MMU control registers should be accessed when both instruction and data address translation is turned off ($MSR_{IR}$ =0 and $MSR_{DR}$ =0). Prior to an mtspr MD_DBCAM, Rx instruction, an eieio instruction should be placed.

# SECTION 12
# SYSTEM INTERFACE UNIT

## 12.1 INTRODUCTION

The system interface unit (SIU) of the MPC821 consists of several functions that control system startup, initialization and operation, protection, and the external system bus. The following is a list of the system interface unit's important features:

- System configuration and protection
- System reset monitoring and generation
- Clock synthesizer
- Power management
- External bus interface (EBI) control
- Eight memory banks supported by the memory controller
- Debug support
- IEEE 1149.1 test access port

The system configuration and protection function provides various monitors and timers, including the bus monitor, software watchdog timer, periodic interrupt timer, PowerPC (PPC) decrementer, timebase, and real-time clock. The clock synthesizer generates the clock signals used by the SIU, as well as the other modules and external devices. This circuitry generates the system clock from an inexpensive 32-KHz/4-MHz crystal. The SIU supports various low-power modes and each one supplies a different range of power consumption, functionality and wake-up time. The clock scheme supports low power modes for applications that use the baud rate generators and/or serial ports during the standby mode. The main system clock can be changed dynamically while the baud rate generators and serial ports work with a fixed frequency. For more information, refer to **Section 5 Clocks and Power Control**.

The external bus interface (EBI) handles the transfer of information between the internal busses and the memory or peripherals in the external address space. The MPC821 is designed to allow external bus masters to request and obtain mastership of the system bus. **Section 13 Bus Interface** describes the bus operation, but the configuration control of the EBI is explained in this section. The memory controller module provides a glueless interface to many types of memory devices and peripherals. It supports up to eight memory banks, each with its own device and timing attributes. The MPC821 implementation supports circuit board test strategies through a user-accessible test logic that is fully compliant with the **Section 20 IEEE 1149.1 Test Access Port**.

## 12.2 SYSTEM CONFIGURATION AND PROTECTION

The MPC821 incorporates many system functions that normally must be provided in external circuits. In addition, it is designed to provide maximum system safeguards against hardware and/or software faults. The following features are provided in the system configuration and protection submodule.

- **System Configuration**—The SIU allows the user to configure the system according to the particular requirements. The functions include control of parity checking, show cycle operation, and part and mask number constants.
- **Bus Monitor**—Monitors the transfer acknowledge ($\overline{\text{TA}}$) response time for all bus accesses initiated by internal masters. A transfer error acknowledge ($\overline{\text{TEA}}$) is asserted if the $\overline{\text{TA}}$ response limit is exceeded. This function can be disabled if needed.
- **Software Watchdog Timer**—Asserts a reset or NMI interrupt, selected by the system protection control register (SYPCR) if the software fails to service the software watchdog timer (SWT) for a certain period of time (for example, because the software is trapped in a loop or lost). After a system reset, this function is enabled, selects a maximum timeout period, and asserts a system reset if the timeout is reached. The SWT may be disabled or its timeout period may be changed in the SYPCR. Once the SYPCR is written, it cannot be written again until a system reset.
- **Periodic Interrupt Timer**—Generates periodic interrupts for use with a real-time operating system or the application software. The periodic interrupt timer (PIT) is clocked by the pitrtclk clock, providing a period from 122 microseconds to 8,000 milliseconds (assuming a 32.768-KHz crystal). The PIT function can be disabled if needed.
- **PowerPC Timebase Counter**—A 64-bit counter defined by the PowerPC™ architecture to provide a timebase reference for the operating system or application software. The timebase counter (TB) has two independent reference registers that generates a maskable interrupt when the timebase counter reaches the value programmed in one of the two reference registers. The associated bit in the TB status register is set for the reference register that generated the interrupt. The TB is clocked by the tmbclk clock.
- **PowerPC Decrementer**—A 32-bit decrementing counter defined by the PowerPC architecture to provide a decrementer interrupt. This binary counter is clocked by the same frequency as the timebase (also defined by the PowerPC architecture). The decrementer (DEC) is clocked by the tmbclk clock and the period for the DEC when it is driven by a 4-MHz oscillator is 4,295 seconds, which is approximately 71.6 minutes.
- **Real-Time Clock**—Provides a time-of-day information to the operating system/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 real-time clock (RTC) is clocked by the pitrtclk clock.
- **Freeze Support**—The SIU allows control of whether the SWT, PIT, TB, DEC and RTC should continue to run during the freeze mode.

Figure 12-1 illustrates the block diagram of the system configuration and protection logic.

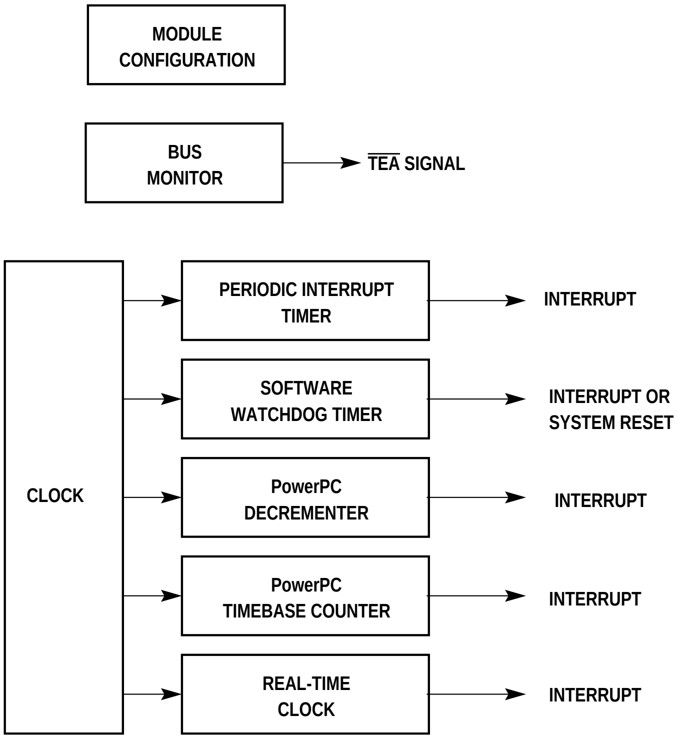


Figure 12-1. System Configuration and Protection Logic

## 12.2.1 System Configuration

Many aspects of the system configuration are controlled by the SIU module configuration register (SIUMCR). The SIUMCR programming model is described in more detail in **Section 12.4 Programming Model**.

**12.2.1.1 SIU INTERRUPT CONFIGURATION.** An overview of the MPC821 interrupt structure is illustrated in Figure 12-2. The SIU receives interrupts from internal sources, such as the PIT or RTC, from the communication processor module (CPM) (with it’s own interrupt controller) and from external pins $\overline{\text{IRQ}}$[0:7].

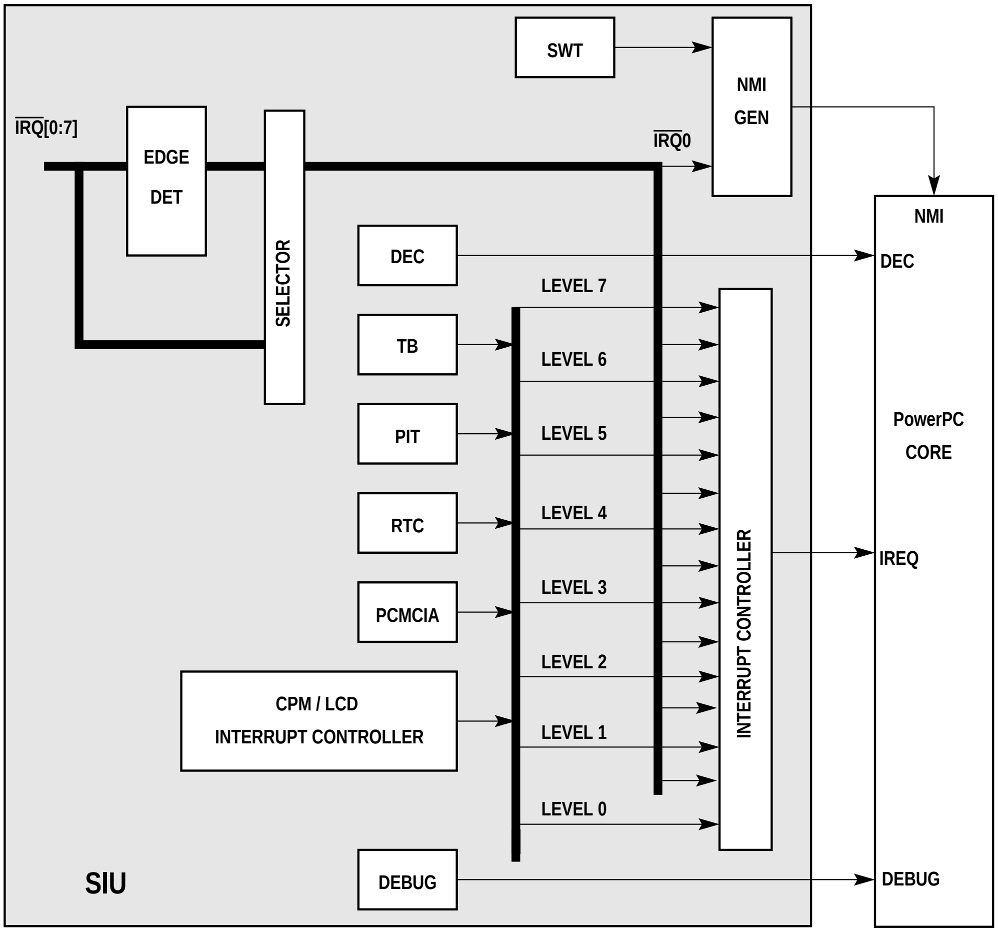


Figure 12-2. MPC821 Interrupt Structure

If it is programmed to generate an interrupt, the SWT always generates a nonmaskable interrupt (NMI) to the core. The external pin IRQ0 can generate NMI as well. Notice that the core takes the system reset interrupt when an NMI is asserted and the external interrupt for any other interrupt asserted by the interrupt controller. Each one of the external pins IRQ[1:7] has its own dedicated assigned priority level and there are eight additional interrupt priority levels. Each one of the SIU internal interrupt sources, the interrupt request which is generated by the CPM interrupt controller, can be assigned by the software to any one of those eight interrupt priority levels. Thus, a very flexible interrupt scheme is realized.

**12.2.1.2 SIU INTERRUPT SOURCES PRIORITY.** The SIU has 15 interrupt sources that assert just one interrupt request to the PowerPC core. There are seven external IRQ pins (the eighth one generates a NMI) and eight interrupt levels and the priority between all interrupt sources is shown in Table 12-1.

**Table 12-1. Priority of SIU Interrupt Sources**

| NUMBER | PRIORITY LEVEL | INTERRUPT SOURCE DESCRIPTION | INTERRUPT CODE |
|---|---|---|---|
| 0 | Highest | $\overline{\text{IRQ0}}$ | 00000000 |
| 1 | | Level 0 | 00000100 |
| 2 | | $\overline{\text{IRQ1}}$ | 00001000 |
| 3 | | Level 1 | 00001100 |
| 4 | | $\overline{\text{IRQ2}}$ | 00010000 |
| 5 | | Level 2 | 00010100 |
| 6 | | $\overline{\text{IRQ3}}$ | 00011000 |
| 7 | | Level 3 | 00011100 |
| 8 | | $\overline{\text{IRQ4}}$ | 00100000 |
| 9 | | Level 4 | 00100100 |
| 10 | | $\overline{\text{IRQ5}}$ | 00101000 |
| 11 | | Level 5 | 00101100 |
| 12 | | $\overline{\text{IRQ6}}$ | 00110000 |
| 13 | | Level 6 | 00110100 |
| 14 | | $\overline{\text{IRQ7}}$ | 00111000 |
| 15 | Lowest | Level 7 | 00111100 |
| 16-31 | | Reserved | |

**12.2.1.3 SIU INTERRUPT CONTROLLER PROGRAMMING MODEL.** The SIU interrupt controller contains the SIPEND, SIMASK, SIEL, and SIVEC registers.

**12.2.1.3.1 SIPEND Register.** The SIPEND is a 32-bit register where each bit corresponds to an interrupt request. The bits associated with internal exceptions indicate, if set, that an interrupt service is requested (if not masked by the corresponding bit in the SIMASK register). These bits reflect the status of the internal requestor device and is cleared when the appropriate actions are initiated by the software in the device itself. Writing to these bits has no effect.

The bits associated with the $\overline{\text{IRQ}}$ pins have a different behavior depending on the sensitivity defined for them in the SIEL register. When the $\overline{\text{IRQ}}$ is defined as a “level” interrupt the corresponding bit behaves similar to the bits associated with internal interrupt sources. When the $\overline{\text{IRQ}}$ is defined as an “edge” interrupt and if the corresponding bit is set, it indicates that a falling edge was detected on the line and the bit can be reset by software by writing a 1 to it.

SIPEND

| BIT | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | IRQ0 | LVL0 | IRQ1 | LVL1 | IRQ2 | LVL2 | IRQ3 | LVL3 | IRQ4 | LVL4 | IRQ5 | LVL5 | IRQ6 | LVL6 | IRQ7 | LV7 |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| ADDR | 010 | | | | | | | | | | | | | | | |
| BIT | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | RESERVED | | | | | | | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| ADDR | 012 | | | | | | | | | | | | | | | |

**12.2.1.3.2 SIMASK Register.** The SIMASK is a 32-bit read/write register where each bit corresponds to an interrupt request bit in the SIPEND register. When the bit is set, it enables the generation of an interrupt request to the CPU core. SIMASK is updated by the software and cleared at reset and it is the responsibility of the software to determine which of the interrupt sources are enabled at a given time.

SIMASK

| BIT | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | IRM0 | LVM0 | IRM1 | LVM1 | IRM2 | LVM2 | IRM3 | LVM3 | IRM4 | LVM4 | IRM5 | LVM5 | IRM6 | LVM6 | IRM7 | LVM7 |
| RESET | 0 | | | | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| ADDR | 014 | | | | | | | | | | | | | | | |
| BIT | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | RESERVED | | | | | | | | | | | | | | | |
| RESET | 0 | | | | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| ADDR | 016 | | | | | | | | | | | | | | | |

**12.2.1.3.3 SIEL Register.** The SIEL is a 32-bit read/write register where each pair of bits correspond to an external interrupt request. The EDx bit, if set, specifies that a falling edge in the corresponding $\overline{\text{IRQ}}$ line is detected as an interrupt request. When the EDx bit is “0”, a low logical level in the $\overline{\text{IRQ}}$ line is detected as an interrupt request. The WMx (wake-up mask) bit, if set, indicates that a low level detection in the corresponding interrupt request line causes the MPC821 to exit low-power mode.

SIEL

| BIT | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | ED0 | WM0 | ED1 | WM1 | ED2 | WM2 | ED3 | WM3 | ED4 | WM4 | ED5 | WM5 | ED6 | WM6 | ED7 | WM7 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 018 | | | | | | | | | | | | | | | |
| BIT | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | RESERVED | | | | | | | | | | | | | | | |
| 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 | 018 | | | | | | | | | | | | | | | |

**12.2.1.3.4 SIVEC Register.** The SIVEC is a 32-bit read-only register that contains an 8-bit code representing the unmasked interrupt source of the highest priority level. The SIVEC can be read as either a byte, half, or a word. When read as a byte, a branch table can be used in which each entry contains one instruction (branch). When read as a half-word, each entry can contain a full routine of up to 256 instructions. The interrupt code is defined such that it’s two least-significant bits are zero, thus allowing indexing into the table. Refer to Figure 12-3 for details.

SIVEC

| BIT | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | INTERRUPT CODE | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| ADDR | 01C | | | | | | | | | | | | | | | |
| BIT | 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 | | | | | | | | | | | | | | | | |
| R/W | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| ADDR | 01E | | | | | | | | | | | | | | | |

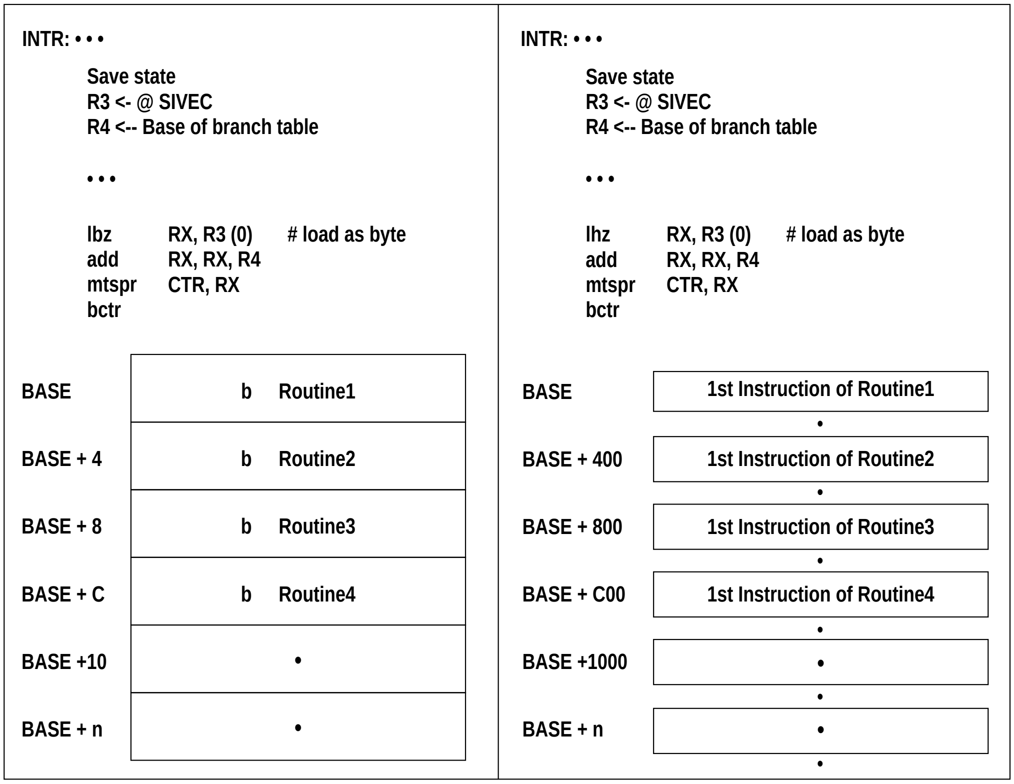


**Figure 12-3. Interrupt Table Handling Example**

### 12.2.2 Bus Monitor

The bus monitor ensures that each bus cycle is terminated within a reasonable period of time. The MPC821 SIU provides a bus monitor option to monitor internal to external bus accesses on the external bus. The monitor counts from transfer start to transfer acknowledge and from transfer acknowledge to transfer acknowledge within bursts. If the monitor times out, transfer error acknowledge ($\overline{\text{TEA}}$) is internally asserted. The programmability of the timeout allows for a variation in system peripheral response time. The timing mechanism is clocked by the system clock divided by eight. The maximum value can be 2,040 system clocks. The bus monitor will always be active when freeze is asserted or when a debug mode request is pending, regardless of the state of the BMT enable bit.

### 12.2.3 PowerPC Decrementer

The PowerPC decrementer (DEC) is a 32-bit decrementing counter defined by the PowerPC architecture to provide a decrementer interrupt. This binary counter is clocked by the same frequency as the timebase (also defined by the PowerPC architecture). In the MPC821, the DEC is clocked by the tmbclk clock.

$$T_{dec} = \frac{2^{32}}{\left(F_{tmbclk}\right)}$$

The state of the DEC is not affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ and, therefore, should be initialized by the software. The DEC runs continuously after power-up. The decrementer continues counting while $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ are asserted and implemented with the following requirements in mind. The decrementer interrupt is also sent to the power-down wake-up logic, allowing a pin to be toggled while the rest of the MPC821 is not powered.

1. The operation of the timebase and the decrementer are coherent, which means the counters are driven by the same fundamental timebase.
2. Loading from the decrementer has no effect on the decrementer.
3. Storing to the decrementer replaces the value in the decrementer with the value in the GPR.
4. Whenever Bit 0 (MSB) of the decrementer changes from 0 to 1, an interrupt request is signaled. If multiple decrementer interrupt requests are received before the first one is reported, only one interrupt is reported.
5. If the decrementer is altered by the software and the content of Bit 0 is changed from 0 to 1, an interrupt request is signaled.

A decrementer exception causes a decrementer interrupt request to be pending in the CPU core. When the decrementer interrupt is taken, the decrementer interrupt request is automatically cleared. The following chart shows some of the periods available for the DEC, assuming a 4-MHz crystal.

| COUNT VALUE | TIMEOUT |
|---:|---:|
| 0 | 1 microsecond |
| 9 | 10. microseconds |
| 99 | 100. microseconds |
| 999 | 1.0 millisecond |
| 9999 | 10.0 milliseconds |

| COUNT VALUE | TIMEOUT |
|---:|---:|
| 999999 | 1.0 second |
| 9999999 | 10.0 seconds |
| 99999999 | 100.0 seconds |
| 999999999 | 1000. seconds |
| (hex) FFFFFFFF | 4295 seconds |

## 12.2.4 PowerPC TIMEBASE

The timebase (TB) is a PowerPC architecture defined timer facility. It is a 64-bit free-running binary counter which is incremented at a frequency determined by each implementation of the timebase. There is no interrupt or other indication generated when the count rolls over. The period of the TB depends on the driving frequency. For the MPC821, the TB is clocked by the tmbclk clock and the period for the TB is:

$$T_{TB} = \frac{2^{64}}{F_{tmbclk}}$$

The state of the TB is not affected by any resets and should be initialized by the software. Reads and writes of the TB are restricted to special instructions. For the MPC821 implementation, it is not possible to read or write the entire TB in a single instruction. Therefore, the mttb and mftb instructions are used to move the lower half of the timebase (TBL) while the mttbu and mftbu instructions are used to move the upper half of the timebase (TBU). The TB has two reference registers associated with it. A maskable interrupt is generated when the TB count reaches to the value programmed in one of the two reference registers and two status bits indicate which one of the two reference registers generated the interrupt.

## 12.2.5 REAL-TIME CLOCK

The real-time clock (RTC) is a 45-bit counter which is clocked by the pitrtclk clock. It is used to provide a time-of-day indication to the operating system and application software. The counter is not affected by reset and operates in all low-power modes. It is initialized by the software. The RTC can be programmed to generate a maskable interrupt when the time value matches the value programmed in its associated alarm register. It can also be programmed to generate an interrupt once every second. A control and status register is used to enable or disable the different functions and report the interrupt source. The RTC-related registers (RTCSC, RTC, RTSEC, and RTCAL) are “locked” after $\overline{\text{PORESET}}$. To enable a write action to any of these registers, a previously open operation should be taken. For more information refer to **Section 5.11.2 Keep Alive Power Registers Lock Mechanism**.

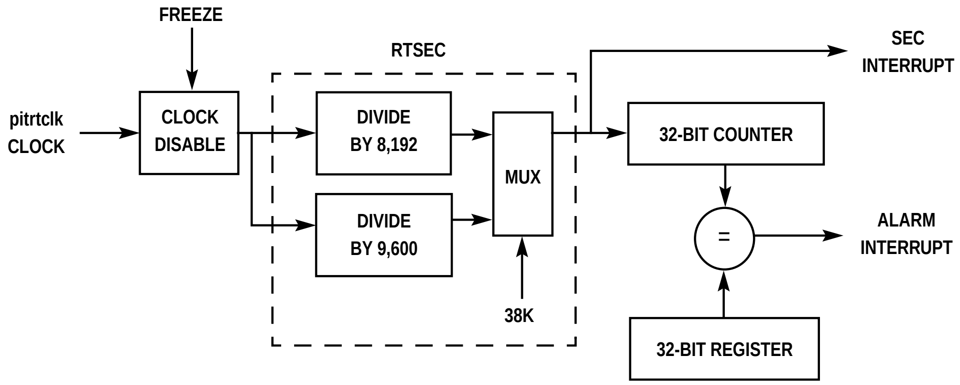


**Figure 12-4. RTC Block Diagram**

### 12.2.6 Periodic Interrupt Timer

The periodic interrupt timer (PIT) consists of a 16-bit counter clocked by the pitrtclk clock that the clock module supplies. The 16-bit counter decrements to zero when loaded with a value from the PITC and after the timer reaches zero, the PS bit is set and an interrupt is generated if the PIE bit is a logic 1. At the next input clock edge, the value in the PITC is loaded into the counter and the process starts over again. When a new value is loaded into the PITC, the PIT is updated, the divider is reset, and the counter begins counting. If the PS bit is not cleared, it makes an interrupt pending at the interrupt controller and remains pending until it is cleared. If the PS bit is set again, prior to being cleared, the interrupt remains pending until the PS bit is cleared. Any write to the PITC stops the current countdown and the count resumes with the new value in PITC. If the PTE bit is not set, the PIT will be unable to count and retains the old count value. Reads of the PIT have no effect on the PIT.

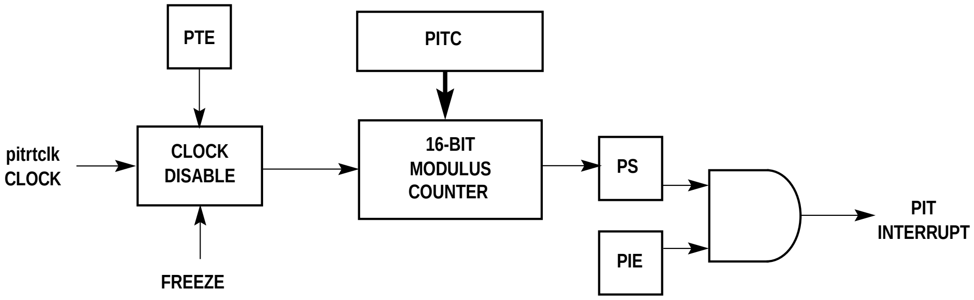


**Figure 12-5. PIT Block Diagram**

The timeout period is calculated as:

$$\mathrm{PIT}_{\mathrm{period}} = \frac{\mathrm{PITC}+1}{F_{\mathrm{pitrtclk}}} = \frac{\mathrm{PITC}+1}{\left(\frac{\mathrm{ExternalClock}}{1\mathrm{or}128}\right) \div 4}$$

Solving this equation using a 32.768 KHz external clock gives:

$$\mathrm{PITperiod} = \frac{\mathrm{PITC}+1}{8192}$$

This gives a range from 122 microseconds, with a PITC of $0000, to 8 seconds, with a PITC of $FFFF.

### 12.2.7 Software Watchdog Timer

The SIU provides the software watchdog timer (SWT) option to prevent system lockout in case the software becomes trapped in loops with no controlled exit. The SWT is enabled after system reset to cause a system reset if it times out. If the SWT is not needed, the user must clear the software watchdog enable (SWE) bit in the system protection control register (SYPCR) to disable it. If used, the SWT requires a special service sequence to be executed on a periodic basis. If this periodic servicing action does not occur, the SWT times out and issues a reset or a nonmaskable interrupt (as programmed by the software watchdog reset interrupt select (SWRI) bit in the SYPCR). Once the SYPCR bit is written by the software, the state of the SWE bit cannot be changed. Refer to the system configuration and protection registers for more information. The SWT service sequence consists of the following two steps:

1. Write $556C to the software service register (SWSR).
2. Write $AA39 to the SWSR.

The service sequence clears the watchdog timer and the timing process begins again. If any value other than $556C or $AA39 is written to the SWSR, the entire sequence must start over. Although the writes must occur in the correct order prior to a timeout, any number of instructions may be executed between the writes. This allows interrupts and exceptions to occur between the two writes when necessary. Refer to Figure 12-6 for more information.

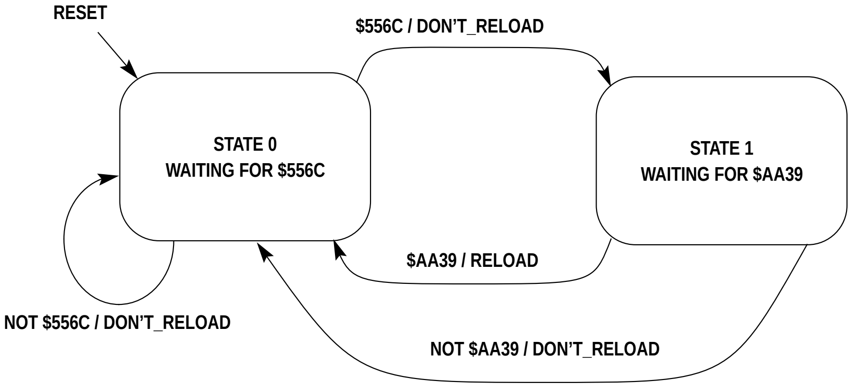


**Figure 12-6. SWT Service State Diagram**

Although most software disciplines permit or even encourage the watchdog concept, some system needs require a selection of timeout periods. For this reason, the software watchdog timer must provide a selectable range for the timeout period. Figure 12-7 illustrates the present method for handling this need. In Figure 12-7, the range is determined by the value software watchdog timing count (SWTC) field. The value held in the SWTC field is then loaded into a 16-bit decrementer clocked by the system clock. An additional divide by 2,048 prescaler is used when necessary.

The decrementer begins counting when loaded with a value from the SWTC field. After the timer reaches $0, a software watchdog expiration request is issued to the reset or NMI control logic. Upon reset, the value in the SWTC is set to the maximum value and is again loaded into the software watchdog register (SWR), starting the process over. When a new value is loaded into the SWTC, the software watchdog timer will not be updated until the servicing sequence is written to the SWSR. If the SWE is loaded with the value 0, the modulus counter will not count.

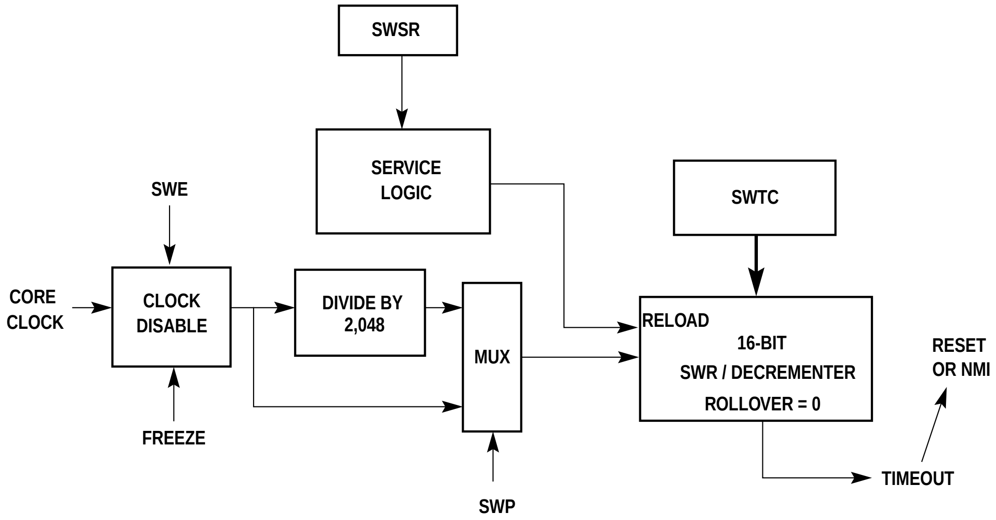


Figure 12-7. SWT Block Diagram

### 12.2.8 Freeze Operation

When the freeze line is asserted, the clocks to the software watchdog, the periodic interrupt timer, the real-time clock, the timebase counter, and the decrementer can be disabled. this is controlled by the associated bits in the control register of each timer. if programmed to stop while freeze, the counters maintains their values while freeze is asserted, unless changed by the software. the bus monitor, however, will be enabled regardless of this signal.

### 12.2.9 Low Power Stop Operation

When the PowerPC core is set in a low power mode (doze, sleep, deep sleep), the SWT is frozen. It remains frozen (and maintains its count value) until the core exits this state and continues executing instructions. Neither the periodic interrupt timer nor the decrementer and timebase is influenced by these low power modes and continues to run at their respective frequencies. These timers are capable of generating an interrupt to bring the MPC821 out of these low power modes.

## 12.3 SIU PINS MULTIPLEXING

Due to the limited number of pins available in the MPC821 package, some of the functionalities defined in the various sections (bus operation, memory controller, and PCMCIA) share pins. The actual pin-out of the MPC821 is defined in **Section 2 External Signals**. The control of the actual functionality used on a specific pin is shown in the following table.

**Table 12-2. SIU Pins Multiplexing Control**

| PIN NAME | PIN CONFIGURATION CONTROL |
|---|---|
| TSIZ0/$\overline{REG}$ | Dynamically active depending if the transaction addresses a slave controlled by the PCMCIA interface. |
| $\overline{BDIP}$/$\overline{GPL\_B}$(5)<br>$\overline{RSV}$/$\overline{IRQ2}$<br>CR/$\overline{IRQ3}$<br>$\overline{KR}$/$\overline{IRQ4}$/SPKROUT<br>DP(0:3)/$\overline{IRQ}$(3:6)<br>FRZ/$\overline{IRQ6}$ | Programmed in SIUMCR. |
| CS(6)/CE(1)_B<br>CS(7)/CE(2)_B | Address matching and bank valid bits.<br>When there is a transfer such that there is a match in either memory controller bank 6 or any PCMCIA bank mapped to slot B, the $\overline{CS}$(6)/$\overline{CE(1)\_B}$ will be asserted.<br>When there is a transfer such that there is a match in either memory controller bank 7 or any PCMCIA bank mapped to slot B, the $\overline{CS}$(7)/$\overline{CE(2)\_B}$ will be asserted. |
| $\overline{WE0}$/$\overline{BS\_B0}$/$\overline{IORD}$<br>$\overline{WE1}$/$\overline{BS\_B1}$/$\overline{IOWR}$<br>$\overline{WE2}$/$\overline{BS\_B2}$/$\overline{PCOE}$<br>$\overline{WE3}$/$\overline{BS\_B3}$/$\overline{PCWE}$ | Dynamically active depending on the machine (GPCM, UPMB, PCMCIA I/F) assigned to control the required slave. |
| $\overline{GPL\_A0}$/$\overline{GPL\_B0}$ | Dynamically active depending on the machine (UPMA or UPMB) assigned to control the required slave. |
| $\overline{OE}$/$\overline{GPL\_A1}$/$\overline{GPL\_B1}$ | Dynamically active depending on the machine (GPCM, UPMA, or UPMB) assigned to control the required slave. |
| $\overline{GPL\_A}$(2:3)/$\overline{GPL\_B}$(2:3)/$\overline{CS}$(2:3) | $\overline{GPL\_A}$(2:3)/$\overline{GPL\_B}$(2:3): Dynamically active depending on the machine (UPMA or UPMB) assigned to control the required slave.<br>$\overline{GPL\_A}$(2:3)/$\overline{CS}$(2:3): Programmed in SIUMCR. |

**Table 12-2. SIU Pins Multiplexing Control (Continued)**

| PIN NAME | PIN CONFIGURATION CONTROL | |
|---|---|---|
| ALE_B/DSCK/AT1<br>IP_B(0:1)/IWP(0:1)/VFLS(0:1)<br>IP_B2/IOIS16_B/AT2<br>IP_B3/IWP2/VF2<br>IP_B4/LWP0/VF0<br>IP_B5/LWP1/VF1<br>IP_B6/DSDI/AT0<br>IP_B7/PTR/AT3<br>TDI/DSDI<br>TCK/DSCK<br>TDO/DSDO | Programmed in SIUMCR and hard reset configuration. | |
| OP2/MODCK0STS<br>OP3/MODCK1/DSDO | At power-on reset:<br>Otherwise: | MODCK(1:2)<br>Programmed in SIUMCR and hard reset configuration. |

## 12.4 PROGRAMMING MODEL

### 12.4.1 System Configuration and Protection Registers

**12.4.1.1 SIU MODULE CONFIGURATION REGISTER.** The SIU module configuration register (SIUMCR) contains bits that configure various features in the SIU module.

SISUMCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | EARB | EARP | | | RESERVED | | | | DS HW | DBGC | | DBPC | | RES | FRC | DLK |
| RESET | 0 | 000 | | | 0 | 0 | 0 | 0 | 0 | 00 | | 00 | | 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 | 000 | | | | | | | | | | | | | | | |

| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | PNCS | OPAR | DPC | MP RE | MLRC | | AEME | SEME | BSC | GB5E | B2DD | B3DD | RESERVED | | | |
| 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 | 002 | | | | | | | | | | | | | | | |

EARB—External Arbitration

If the EARB bit is set (1), then external arbitration is assumed. If it is cleared (0), then internal arbitration is performed. For more information refer to **Section 13.5.6 Arbitration Phase**.

EARP—External Arbitration Request Priority

This field defines the priority of the external master’s arbitration request. This field is valid when EARB is cleared 000 is the lowest priority level and 111 represents the highest. For more information refer to Figure 13-21 in **Section 13.5.6 Arbitration Phase.**

DSHW—Data Show Cycles

This bit selects the show cycle mode to be applied to data cycles. Instruction show cycles are programmed in the ICTRL register. This field is locked by the DLK bit. The following chart shows the meaning of the field. Refer to **Section 19.5.3 Development Support Registers Description** for more information.

| DSHW | MEANING |
|---|---|
| 0 | Disable show cycles for all internal data cycles. |
| 1 | Show address and data of all internal data cycles. |

DBGC—Debug Pins Configuration

This field configures the functionality of the following pins:

| DBGC | PIN USAGE | |
|---|---|---|
| 00 | IP_B(0:1)/IWP(0:1)/VFLS(0:1) functions as IP_B(0:1) | ALE_B/DSCK/AT1 functions as ALE_B |
| | IP_B3/IWP2/VF2 functions as IP_B3 | IP_B2/AT2 functions as IP_B2 |
| | IP_B4/LWP0/VF0 functions as IP_B4 | IP_B6/DSDI/AT0 functions as IP_B6 |
| | IP_B5/LWP1/VF1 functions as P_B5 | IP_B7/PTR/AT3 functions as IP_B7 |
| | OP2/MODCK1/$\overline{\text{STS}}$ functions as OP2 | OP3/MODCK2/DSDO functions as OP3 |
| 01 | IP_B(0:1)/IWP(0:1)/VFLS(0:1) functions as WP(0:1) | ALE_B/DSCK/AT1 functions as AT1 |
| | IP_B3/IWP2/VF2 functions as IWP2 | IP_B2/AT2 functions as AT2 |
| | IP_B4/LWP0/VF0 functions as LWP0 | IP_B6/DSDI/AT0 functions as AT0 |
| | IP_B5/LWP1/VF1 functions as LWP1 | IP_B7/PTR/AT3 functions as AT3 |
| | OP2/MODCK1/$\overline{\text{STS}}$ functions as $\overline{\text{STS}}$ | OP3/MODCK2/DSDO functions as OP3 |
| 10 | Reserved | |
| 11 | IP_B(0:1)/IWP(0:1)/VFLS(0:1) functions as VFLS(0:1) | ALE_B/DSCK/AT1 functions as AT1 |
| | IP_B3/IWP2/VF2 functions as VF2 | IP_B2/AT2 functions as AT2 |
| | IP_B4/LWP0/VF0 functions as VF0 | IP_B6/DSDI/AT0 functions as AT0 |
| | IP_B5/LWP1/VF1 functions as VF1 | IP_B7/PTR/AT3 functions as AT3 |
| | OP2/MODCK1/$\overline{\text{STS}}$ functions as $\overline{\text{STS}}$ | OP3/MODCK2/DSDO functions as OP3 |

DBPC—Debug Port Pins Configuration

This field configures the pins on which the development port is active.

| DBPC | PIN USAGE | |
|---|---|---|
| 00 | ALE_B/DSCK/AT1 functions as defined by DBGC<br>IP_B6/DSDI/AT0 functions as defined by DBGC<br>OP3/MODCK2/DSDO functions as defined by DBGC<br>IP_B7/PTR/AT3 functions as defined by DBGC | TCK/DSCK functions as DSCK<br>TDI/DSDI functions as DSDI<br>TDO/DSDO functions as DSDO |
| 01 | ALE_B/DSCK/AT1 functions as defined by DBGC<br>IP_B6/DSDI/AT0 functions as defined by DBGC<br>OP3/MODCK2/DSDO functions as defined by DBGC<br>IP_B7/PTR/AT3 functions as defined by DBGC | TCK/DSCK functions as TCK<br>TDI/DSDI functions as TDI<br>TDO/DSDO functions as TDO |
| 10 | Reserved | |
| 11 | ALE_B/DSCK/AT1 functions as DSCK<br>IP_B6/DSDI/AT0 functions as DSDI<br>OP3/MODCK2/DSDO functions as DSDO<br>IP_B7/PTR/AT3 functions as PTR | TCK/DSCK functions as TCK<br>TDI/DSDI functions as TDI<br>TDO/DSDO functions as TDO |

FRC—FRZ pin Configuration

This bit configures the functionality of the FRZ/$\overline{\text{IRQ6}}$ pin.

| FRC | PIN USAGE |
|---|---|
| 0 | FRZ/$\overline{\text{IRQ6}}$ functions as FRZ |
| 1 | FRZ/$\overline{\text{IRQ6}}$ functions as $\overline{\text{IRQ6}}$ |

DLK—Debug Register Lock

If the DLK bit is set (1), then bits 8:15 are locked. That is, writes to those bits are no longer performed. These bits (8:15) are writable in test mode and when the internal freeze is asserted, regardless of the DLK bit state. DLK is cleared by reset.

PNCS—Parity Enable For Nonmemory Controller Regions

The bit enables parity generation/checking for memory regions not under memory controller control.

OPAR—Odd Parity

This attribute is used to program odd or even parity. It can also be used to generate parity errors for testing purposes by writing the memory with OPAR = 1 and reading the memory with OPAR = 0.

DPC—Data Parity Pins Configuration

This bit configures the functionality of the DP(0:3)/$\overline{\text{IRQ}}$(3:6) pins.

| DPC | PIN USAGE |
|---|---|
| 0 | DP(0:3)/$\overline{\text{IRQ}}$(3:6) functions as $\overline{\text{IRQ}}$(3:6) |
| 1 | DP(0:3)/$\overline{\text{IRQ}}$(3:6) functions as DP(0:3) |

MPRE—Multiprocessors Reservation Enable

If the MPRE bit is set (1), then the interprocessor reservation protocol is enabled. The two pins $\overline{\text{RSV}}$ and CR are functional as defined in **Section 13.5.9 Storage Reservation**. If it is cleared (0), then the interprocessor reservation protocol is disabled. The two pins serve as $\overline{\text{IRQ2}}$ and $\overline{\text{IRQ3}}$.

MLRC—Multi-Level Reservation Control

This field configures the functionality of the $\overline{\text{KR}}$/$\overline{\text{IRQ4}}$/SPKROUT pin.

| MLRC | PIN USAGE |
|---|---|
| 00 | $\overline{\text{KR}}$/$\overline{\text{IRQ4}}$/SPKROUT functions as $\overline{\text{IRQ4}}$ |
| 01 | $\overline{\text{KR}}$/$\overline{\text{IRQ4}}$/SPKROUT is three-stated |
| 10 | $\overline{\text{KR}}$/$\overline{\text{IRQ4}}$/SPKROUT functions as $\overline{\text{KR}}$ |
| 11 | $\overline{\text{KR}}$/$\overline{\text{IRQ4}}$/SPKROUT functions as SPKROUT |

AEME—Asynchronous External Master Enable

This bit configures how the memory controller refers to external asynchronous masters initiating a transaction. If the bit is set, any assertion on the $\overline{\text{AS}}$ pin is interpreted by the memory controller as an external asynchronous master initiating a transaction. If reset, the memory controller ignores the value of the $\overline{\text{AS}}$ pin.

SEME—Synchronous External Master Enable

This bit configures how the memory controller refers to external synchronous masters initiating a transaction. If the bit is set, any $\overline{\text{TS}}$ assertion that the external bus does not own is interpreted by the memory controller as an external synchronous master initiating a transaction. If reset, the memory controller ignores the value of the $\overline{\text{TS}}$ pin if it does not own the external bus.

BSC—Byte Select Configuration

This bit configure how the memory controller byte select and strobes, as well as the PCMCIA interface strobes, are configured.

| BSC | PIN USAGE |
|---|---|
| 0 | BS_A(0:3) are driven just on their dedicated pins.<br>WE0/BS_B0/IORD is driven on its dedicated pin.<br>WE1/BS_B1/IOWR is driven on its dedicated pin.<br>WE2/BS_B2/PCOE is driven on its dedicated pin.<br>WE3/BS_B3/PCWE is driven on its dedicated pin. |
| 1 | Assertion of either BS_A0, WE0, BS_B0 or IORD is driven on both BS_A0 pin AND on WE0/BS_B0/IORD.<br>Assertion of either BS_A1, WE1, BS_B1 or IOWR is driven on both BS_A1 pin AND on WE1/BS_B1/IOWR.<br>Assertion of either BS_A2, WE2, BS_B2 or PCOE is driven on both BS_A2 pin AND on WE2/BS_B2/PCOE.<br>Assertion of either BS_A3, WE3, BS_B3 or PCWE is driven on both BS_A3 pin AND on WE3/BS_B3/PCWE. |

GB5E—GPL_B(5) Enable

If the GB5E bit is set (1), then the GPL_B(5) of the memory controller functionality is active. If it is cleared (0), the BDIP functionality is active.

B2DD—Bank 2 Double Drive

If the B2DD bit is set (1), then the CS2 line is reflected on GPL_A2/GPL_B2.

B3DD—Bank 3 Double Drive

If the B3DD bit is set (1), then the CS3 line is reflected on GPL_A3/GPL_B3.

**12.4.1.2 INTERNAL MEMORY MAP REGISTER.** The internal memory map register (IMMR) is a special register located within the PowerPC special register space. It contains identification of a specific device as well as the base for the internal memory map. Based on the value read from this register, the software can deduce availability and location of any on-chip system resources. Contents of this register can be read by the mfspr instruction and the ISB field can be written by the mtspr instruction. The PARTNUM and MASKNUM fields are mask programmed and cannot be changed for any particular device.

**IMMR**

| **BITS** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **FIELD** | ISB | | | | | | | | | | | | | | | |
| **RESET** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 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 |
| **BITS** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **FIELD** | PARTNUM | | | | | | | | MASKNUM | | | | | | | |
| **RESET** | | | | | | | | | | | | | | | | |
| **R/W** | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |

ISB—Internal Space Base

This read/write field defines the base address of the internal memory space. The initial value of this field can be configured at reset to one of four addresses and then can be changed to any value by the software. The number of programmable bits in this field, and hence the resolution of the location of internal space, depends on the internal memory space of a specific implementation. In the MPC821, all 16 bits can be programmed. Refer to **Section 3 Memory Map** for details on the device's internal memory map and **Section 4.3.1.1 Hard Reset Configuration Word** for the available and default initial values.

PARTNUM—Part Number

This read-only field is mask programmed with a code corresponding to the part number of the part on which the SIU is located. It is intended to help factory test and user code which is sensitive to part changes. This changes when the part number changes. For example, it would change if any new module is added or if the size of any memory module is changed. It would not change if the part is changed to fix a bug in an existing module. The MPC821 has an ID of $00.

MASKNUM—Mask Number

This read-only field is mask programmed with a code corresponding to the mask number of the part on which the SIU is located. It is intended to help factory test and user code which is sensitive to part changes. It is programmed in a commonly changed layer and should be changed for all mask set changes. The MPC821 (Rev 0) has an ID of $00.

**12.4.1.3 SYSTEM PROTECTION CONTROL REGISTER.** The system protection control register (SYPCR) controls the system monitors, software watchdog period, and bus monitor timing. This register can be read at any time, but can only be written once after system reset.

SYPCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | SWTC | | | | | | | | | | | | | | | |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| R/W | | | | | | | | | | | | | | | | |
| ADDR | 004 | | | | | | | | | | | | | | | |

| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | BMT | | | | | | | | BME | RESERVED | | | SWF | SWE | SWRI | SWP |
| RESET | 11111111 | | | | | | | | 0 | 000 | | | 0 | 1 | 1 | 1 |
| R/W | | | | | | | | | | | | | | | | |
| ADDR | 006 | | | | | | | | | | | | | | | |

BME—Bus Monitor Enable

This bit controls the operation of the bus monitor when an internal to external bus cycle is executed.

| BME | MEANING |
|---|---|
| 0 | Disable Bus Monitor |
| 1 | Enable Bus Monitor |

BMT—Bus Monitor Timing
This field defines the timeout period, in 8 system clock resolution, for the bus monitor.

SWF—Software Watchdog Freeze
If this bit is asserted (1), the software watchdog timer stops when freeze is asserted.

SWE—Software Watchdog Enable
This bit enables the operation of the software watchdog timer. It should be cleared by software after a system reset to disable the SWT.

SWRI—Software Watchdog Reset/Interrupt Select
When this bit is cleared SWT causes a nonmaskable interrupt to the PowerPC core. When this bit is set SWT causes a system reset (this is the default value after system reset).

SWP—Software Watchdog Prescale
This bit controls the divide-by-2,048 SWT prescaler. If it is cleared, the SWT is not prescaled and if it is set, the SWT clock is prescaled.

SWTC—Software Watchdog Timer Count
These bits contain the count value for the SWT.

**12.4.1.4 SOFTWARE SERVICE REGISTER.** The software service register (SWSR) is the location to which the SWT servicing sequence is written. To prevent SWT timeout, the user should write a $556C followed by $AA93 to this register. The SWSR can be written at any time, but returns all zeros when read.

**12.4.1.5 TRANSFER ERROR STATUS REGISTER.** The transfer error status register (TESR) contains a bit for each exception source generated by a transfer error. A bit set to logic 1 indicates what type of transfer error exception occurred since the last time the bits were cleared by reset or by the normal software status bit clearing mechanism. These bits may be set because of canceled speculative accesses that do not cause an interrupt. The register has two identical sets of bit fields–one is associated with instruction transfers and the other with data transfers.

TESR

| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | IEXT | IBM | IPB0 | IPB1 | IPB2 | IPB3 | RESERVED | | DEXT | DBM | DPB0 | DPB1 | DPB2 | DPB3 |
| RESET | | | 0 | 0 | 0 | 0 | 0 | 0 | | | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | | | | | | | | | | | | | | | | |
| ADDR | 020 | | | | | | | | | | | | | | | |

IPB[0:3]—Instruction Parity Error on Byte

There are four parity error status bits, one per 8-bit lane. A bit is set for the byte that had a parity error when an instruction was fetch. Parity check for a memory region that is not under memory controller control is enabled by the PNCS bit in SIUMCR.

IBM—Instruction transfer Monitor Timeout

This bit is set if the cycle is terminated by a bus monitor timeout when an instruction fetch is initiated.

IEXT—Instruction External Transfer Error Acknowledge

This bit is set if the cycle is terminated by an externally generated $\overline{\text{TEA}}$ signal when an instruction fetch is initiated.

DPB[0:3]—Data Parity Error On Byte

There are four parity error status bits, one per 8 bit lane. A bit is set for the byte that has a parity error when a data load is requested by an internal master. Parity check for a memory region which is not under memory controller control is enabled by the PNCS bit in SIUMCR.

DBM—Data Transfer Monitor Timeout

This bit is set if the cycle is terminated by a bus monitor timeout when a data load or store is requested by an internal master.

DEXT—Data External Transfer Error Acknowledge

This bit is set if the cycle is terminated by an externally generated $\overline{\text{TEA}}$ signal when a data load or store is requested by an internal master.

### 12.4.2 SYSTEM TIMER REGISTERS

The following sections describe registers associated with the system timers. These facilities are powered by the KAPWR and, as such, preserves their value when the main power supply is off. Refer to **Section 5.11.2 Keep Alive Power Registers Lock Mechanism** for details on the required actions needed to guarantee data retention.

### 12.4.3 DECREMENTER REGISTER

The decrementer (DEC) register is defined by PowerPC architecture as a 32-bit register. The values stored in this register are used by a down counter to cause decrementer interrupts. The decrementer causes an interrupt whenever Bit 0 changes from a logic 0 to a logic 1. A read of this register always returns the current count value from the down counter. Contents of this register may be read or written to by the mfspr or the mtspr instruction. This register is not affected by reset. The decrementer is powered by standby power and continues to count when standby power is applied.

| 0 | 31 |
|---|---|
| DEC | |

### 12.4.4 TIMEBASE REGISTERS

**12.4.4.1 TIMEBASE REGISTER.** The timebase (TB) register is a 64-bit register containing a 64-bit integer that is incremented periodically. There is no automatic initialization of the TB register. The system software must perform this initialization. The contents of the register can be written by the mttbl or the mttbu instructions.

| 0 | 31 |
|---|---|
| TBU | |

| 0 | 31 |
|---|---|
| TBL | |

**12.4.4.2 TIMEBASE REFERENCE REGISTERS.** There are two timebase reference registers—TBREFF0 and TBREFF1—associated with the lower part of the timebase. Each register is 32-bits, read/write. When there is a match between the contents of timebase and the reference register, a maskable interrupt is generated.

| 0 | 31 |
|---|---|
| TBREF | |

**12.4.4.3 TIMEBASE CONTROL AND STATUS REGISTER.** The timebase control and status register (TBSCR) is 16-bit read/write register that controls the timebase count enable and interrupt generation. It is also is used for reporting the source of the interrupts and can be read at any time. A status bit is cleared by writing a 1 (writing a zero does not affect a status bit's value) and more than one bit can be cleared at a time.

TBSCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | TBIRQ | | | | | | | | REFA | REFB | RESERVED | | REF<br>AE | REF<br>BE | TBF | TBE |
| RESET | 00000000 | | | | | | | | 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 | 200 | | | | | | | | | | | | | | | |

TBIRQ—Timebase Interrupt Request

These bits determine the interrupt priority level of the timebase. To specify a certain level, the appropriate bit should be set. Refer to **Section 12.2.1.1 SIU Interrupt Configuration** for more information.

REFA, REFB—Reference Interrupt Status

If set, the bit indicates that a match is detected between the corresponding reference register (TBREFF0 for REFA and TBREFF1 for REFB) and the timebase low register. The bit should be cleared by writing a 1.

REFAE, REFBE—Reference Interrupt Enable

If REFAE (REFBE) is asserted (1), the timebase generates an interrupt on assertion of the REFA (REFB) bit.

TBF—Timebase Freeze

If this bit is asserted (1), the timebase and the decrementer stops while freeze is asserted.

TBE—Timebase Enable

This bit, when set to 1, enables the operation of the timebase and decrementer.

## 12.4.5 REAL-TIME CLOCK REGISTERS

**12.4.5.1 REAL-TIME CLOCK STATUS AND CONTROL REGISTER.** The real-time clock status and control register (RTCSC) is used to enable the different RTC functions and for reporting the source of the interrupts. The register can be read at any time. A status bit is cleared by writing a 1 (writing a zero does not affect a status bit’s value) and more than one status bit can be cleared at a time.

RTCSC

| BITS | 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 | 00000000 | | | | | | | | 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 | 220 | | | | | | | | | | | | | | | |

RTCIRQ—RTC Interrupt Request

These bits control the RTC interrupt priority level. Refer to **Section 12.2.1.1 SIU Interrupt Configuration** for details.

SEC—Once Per Second Interrupt

This status bit is set every second and should be cleared by the software.

ALR—Alarm Interrupt

This status bit is set when the value of the RTC is equal to the value programmed in the alarm register.

38K—Real-Time Clock Source Select

If this bit is negated (0), the real-time clock assumes that it is driven by 32.768 KHz to generate the second pulse and it is asserted, the real-time clock assumes 38.4 KHz. This bit is not affected by reset.

SIE—Second Interrupt Enable

If this bit is asserted (1), the real-time clock generates an interrupt when the SEC bit is asserted.

ALE—ALarm Interrupt Enable

If this bit is asserted (1), the real-time clock generates an interrupt when the ALR bit is asserted.

RTF—Real-Time Clock Freeze

If this bit is asserted (1), the real-time clock stops while freeze is asserted.

RTE—Real-Time Clock Enable

When set, the real-time clock timers are enabled. This bit is not affected by reset.

**12.4.5.2 REAL-TIME CLOCK REGISTER.** The real-time clock (RTC) register is a 32-bit read/write register. It contains the current value of the real-time clock.

RTC

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RTC | | | | | | | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 224 | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | RTC | | | | | | | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 226 | | | | | | | | | | | | | | | |

**12.4.5.3 REAL-TIME CLOCK ALARM REGISTER.** The real-time clock alarm (RTCAL) register is a 32-bit read/write register. When the value of the RTC is equal to the value programmed in the alarm register, a maskable interrupt is generated.

RTCAL

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | ALARM | | | | | | | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 22C | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | ALARM | | | | | | | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 22E | | | | | | | | | | | | | | | |

ALARM

The alarm interrupt will be generated as soon as there is a match between the ALARM field and the corresponding bits in the RTC. The resolution of the alarm is 1 second.

## 12.4.6 PERIODIC INTERRUPT REGISTERS

**12.4.6.1 PERIODIC INTERRUPT STATUS AND CONTROL REGISTER.** The periodic interrupt status and control register (PISCR) contains the interrupt request level and the interrupt status bit. It also contains the controls for the 16 bits to be loaded in a modulus counter. This register is always read/write.

PISCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | PIRQ | | | | | | | | PS | RESERVED | | | | PIE | PITF | PTE |
| RESET | 00000000 | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 240 | | | | | | | | | | | | | | | |

PIRQ—Periodic Interrupt Request Level

These bits determine which interrupt request level is asserted when a periodic interrupt is generated.

PS—Periodic Interrupt Status

This bit is asserted if the PIT issues an interrupt. The PIT issues an interrupt after the modulus counter counts to zero. The PS bit can be negated by writing a one to PS. A write of zero has no effect on this bit.

PIE—Periodic Interrupt Enable

If this bit is asserted (1), the periodic interrupt timer generates an interrupt when the PS bit is asserted.

PITF—Periodic Interrupt Timer Freeze

If this bit is asserted (1), the periodic interrupt timer stops while freeze is asserted.

PTE—Periodic Timer Enable

This bit controls the counting of the periodic interrupt timer. When the timer is disabled, it maintains its old value. When the counter is enabled, it continues counting using the previous value.

| PTE | MEANING |
|---|---|
| 0 | Disable Counter |
| 1 | Enable Counter |

**12.4.6.2 PERIODIC INTERRUPT TIMER COUNT.** The periodic interrupt timer count (PITC) register contains the 16 bits to be loaded in a modulus counter. This register is always read/write.

PITC

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | PITC | | | | | | | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 244 | | | | | | | | | | | | | | | |
| **BITS** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| FIELD | RESERVED | | | | | | | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 246 | | | | | | | | | | | | | | | |

PITC—Periodic Interrupt Timing Count

Bits 0-15 are defined as the PITC and it contains the count for the periodic timer. If this field is loaded with the value $FFFF, the maximum count period will be selected.

**12.4.6.3 PERIODIC INTERRUPT TIMER REGISTER.** The periodic interrupt timer register (PITR) is a read-only register that shows the current value in the periodic interrupt down counter. Writes to this register do not affect this register and reads of this register do not have any affect on the counter.

PITR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | PIT | | | | | | | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| ADDR | 248 | | | | | | | | | | | | | | | |
| **BITS** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| FIELD | RESERVED | | | | | | | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| ADDR | 24A | | | | | | | | | | | | | | | |

PIT—Periodic Interrupt Timing Count

Bits 0-15 are defined as the PIT. It contains the current count remaining for the periodic timer. Writes have no effect on this field.

# SECTION 13
# EXTERNAL BUS INTERFACE

## 13.1 OVERVIEW

The MPC821 bus is a synchronous, burstable bus. Signals driven on this bus are required to make the setup and hold time relative to the bus clock's rising edge. The bus has the ability to support multiple masters. The MPC821 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). The access to 16- and 8-bit ports is done for slaves controlled by the memory controller.

## 13.2 FEATURES

The MPC821 bus interface features are listed below.

- 32-bit address bus with transfer size indication
- 32-bit data bus
- TTL-compatible interface
- Bus arbitration logic on-chip supports an external master
- Chip-select and wait state generation internally to support peripheral or static memory devices
- Supports different memory (SRAM, EEPROM) types: asynchronous, burstable memory
- Asynchronous DRAM interface supports
- Flash ROM programming support
- Compatible with PowerPC architecture
- Easy to interface to slave devices
- Bus is synchronous (all signals are referenced to rising edge of bus clock)
- Contains supports for data parity

## 13.3 BUS TRANSFER SIGNALS

The bus transfers information between the MPC821 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 the port width.

The MPC821 contains an address bus that specifies the address for the transfer and a data bus that transfers the data. Control signals indicate the beginning and type of the cycle, as well as the address space and size of the transfer. The selected device then controls the length of the cycle with the signal(s) used to terminate the cycle. A strobe signal for the address bus indicates the validity of the address and provides timing information for the data. The MPC821 bus is synchronous, but the bus and control input signals must be timed to setup and hold times relative to the rising edge of the clock. In this situation, bus cycles can be completed in two clock cycles.

Furthermore, for all inputs, the MPC821 latches the level of the input during a sample window around the rising edge of the clock signal. This window is illustrated in Figure 13-1, where tsu and tho are the input setup and hold times, respectively. To ensure that an input signal is recognized on a specific falling edge of the clock, that input must be stable during the sample window. If an input makes a transition during the window time period, the level recognized by the MPC821 is not predictable; however, the MPC821 always resolves the latched level to either a logic high or low before using it. In addition to meeting input setup and hold times for deterministic operation, all input signals must obey the protocols described in this section.

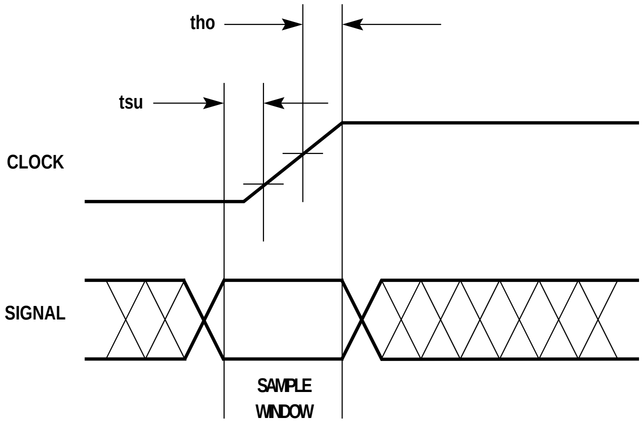


**Figure 13-1. Input Sample Window**

## 13.3.1 Bus Control Signals

The MPC821 initiates a bus cycle by driving the address, size, address type, cycle type, and read/write outputs. At the beginning of a bus cycle, TSIZ0 and TSIZ1 are driven with the address type signals. TSIZ0 and TSIZ1 indicate the number of bytes remaining to be transferred during an operand cycle (consisting of one or more bus cycles). These signals are valid at the rising edge of the clock in which the transfer start ($\overline{\text{TS}}$) signal is asserted. The read/write (RD/$\overline{\text{WR}}$) signal determines the direction of the transfer during a bus cycle. Driven at the beginning of a bus cycle, RD/$\overline{\text{WR}}$ is valid at the rising edge of the clock in which the transfer start signal is asserted. RD/$\overline{\text{WR}}$ only transitions when a write cycle is preceded by a read cycle or vice versa. The signal may remain low for consecutive write cycles.

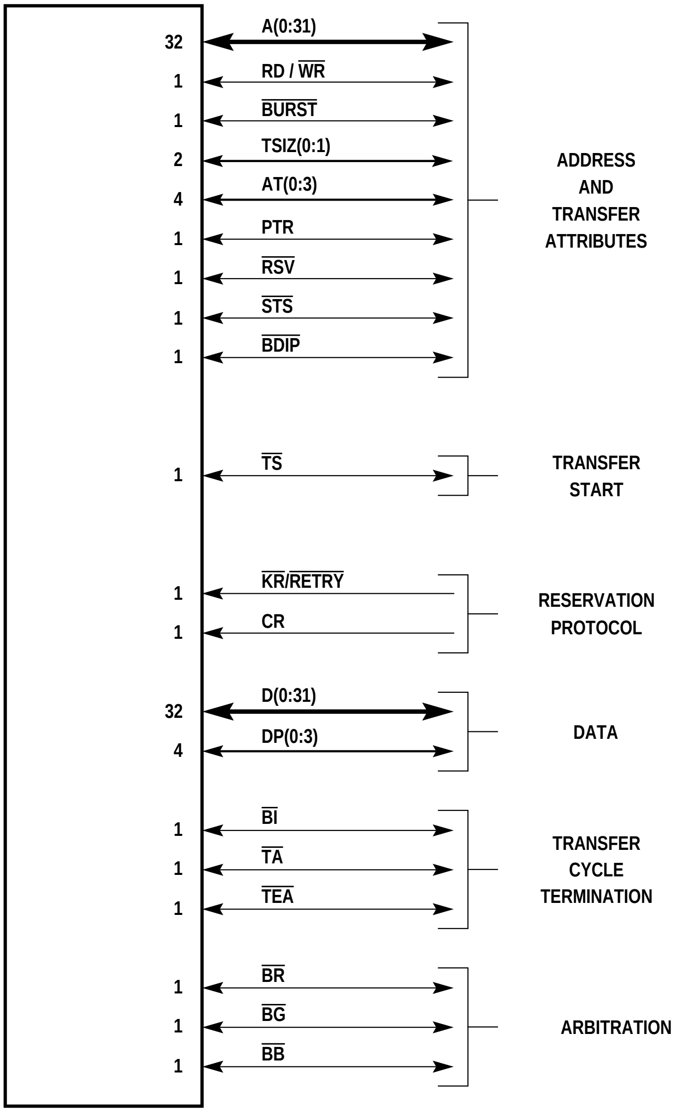


Figure 13-2. MPC821 Bus Signals

## 13.4 BUS INTERFACE SIGNAL DESCRIPTIONS

The following table decribes each signal and more detailed descriptions can be found in subsequent sections.

**Table 13-1. MPC821 SIU Signals**

| SIGNAL NAME | PINS | ACTIVE | I/O | DESCRIPTION |
|---|---|---|---|---|
| ADDRESS AND TRANSFER ATTRIBUTES | | | | |
| A(0:31)<br>ADDRESS BUS | 32 | HIGH | O | Driven by the MPC821 when it “owns” the external bus. Specifies the physical address of the bus transaction. These lines can change during a transaction when controlled by the memory controller. |
| | | | I | Only for testing purposes. |
| RD/$\overline{\text{WR}}$<br>READ/WRITE | 1 | HIGH | O | Driven by the MPC821 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 | Only for testing purposes. |
| $\overline{\text{BURST}}$<br>BURST TRANSFER | 1 | LOW | O | Driven by the MPC821 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 | Only for testing purposes. |
| TSIZ(0:1)<br>TRANSFER SIZE | 2 | HIGH | O | Driven by the MPC821 along with the address when it “owns” the external bus. Specifies the data transfer size for the transaction. |
| | | | I | Only for testing purposes. |
| AT(0:3)<br>ADDRESS TYPE | 3 | HIGH | O | Driven by the MPC821 along with the address when it “owns” the external bus. Indicates additional information about the address on the current transaction. |
| | | | I | Only for testing purposes. |
| $\overline{\text{RSV}}$<br>RESERVATION TRANSFER | 1 | HIGH | O | Driven by the MPC821 along with the address when it “owns” the external bus. Indicates additional information about the address on the current transaction. |
| | | | I | Only for testing purposes. |
| PTR<br>PROGRAM TRACE | 1 | HIGH | O | Driven by the MPC821 along with the address when it “owns” the external bus. Indicates additional information about the address on the current transaction. |
| | | | I | Only for testing purposes. |

## Table 13-1. MPC821 SIU Signals (Continued)

| SIGNAL NAME | PINS | ACTIVE | I/O | DESCRIPTION |
|---|---|---|---|---|
| $\overline{\text{BDIP}}$<br>BURST DATA IN PROGRESS | 1 | LOW | O | Driven by the MPC821 when it “owns” the external bus. It is part of the burst protocol. Asserted indicates that the second beat in front of the current one is requested by the master. This signal is negated prior to the end of a burst to early terminate the burst data phase. |
| | | | I | Only for testing purposes. |
| TRANSFER START | | | | |
| $\overline{\text{TS}}$<br>TRANSFER START | 1 | LOW | O | Driven by the MPC821 when it “owns” the external bus.Indicates the start of a transaction on the external bus. |
| | | | I | $\overline{\text{TS}}$ is input also for testing purposes. |
| $\overline{\text{STS}}$<br>SPECIAL TRANSFER START | 1 | LOW | O | Driven by the MPC821 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 | | | | |
| CR<br>CANCEL RESERVATION | 1 | LOW | I | Each PowerPC CPU has its own signal. Asserted: instructs the bus master to clear its reservation, some other master has touched its reserved space. This is a pulsed signal. |
| $\overline{\text{KR/RETRY}}$<br>KILL RESERVATION/ RETRY | 1 | LOW | I | In case of a Bus cycle, initiated by a STWCX instruction issued by the CPU core, to a nonlocal bus on which the storage reservation has been lost, this signal is used by the nonlocal bus interface to back-off the cycle. Refer to **Section 13.5.9 Storage Reservation** for details.<br>In the case of regular transaction, this signal is driven by the slave device to indicate that the MPC821 has to relinquish the ownership of the bus and retry the cycle. |

Table 13-1. MPC821 SIU Signals (Continued)

| SIGNAL NAME | PINS | ACTIVE | I/O | DESCRIPTION |
|---|---|---|---|---|
| DATA | | | | |
| D(0:31)<br>DATA BUS | 32 | HIGH | | The data bus has the following byte lane assignments:<br>Data Byte Byte Lane<br>D(0:7) 0<br>D(8:15) 1<br>D(16:23) 2<br>D(24:31) 3 |
| | | | O | Driven by the MPC821 when it “owns” the external bus 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 MPC821 |
| DP(0:3)<br>PARITY BUS | 4 | HIGH | | Each parity line corresponds to each one of the data bus lanes:<br>Data Bus Byte Parity Line<br>D(0:7) DP0<br>D(8:15) DP1<br>D(16:23) DP2<br>D(24:31) DP3 |
| | | | O | Driven by the MPC821 when it “owns” the external bus and it initiated a write transaction to a slave device.<br>Each parity line has the parity value (even or odd) of its corresponding data bus byte. For single beat transactions, the byte lanes not selected for the transfer by the 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 will be sampled by the MPC821 and checked (if enabled) against the expected value parity value (even or odd) of its corresponding data bus byte. 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 MPC821 and its parity lines will not be checked. |

### Table 13-1. MPC821 SIU Signals (Continued)

| SIGNAL NAME | PINS | ACTIVE | I/O | DESCRIPTION |
|---|---|---|---|---|
| TRANSFER CYCLE TERMINATION | | | | |
| $\overline{\text{TA}}$<br>TRANSFER ACKNOWLEDGE | 1 | LOW | I | Driven by the slave device to which the current transaction was addressed. Indicates that the slave has 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 one of the transaction beats. |
| | | | O | Driven by the MPC821 when the slave device is controlled by the on-chip memory controller or PCMCIA interface. |
| $\overline{\text{TEA}}$<br>TRANSFER ERROR ACKNOWLEDGE | 1 | LOW | I | Driven by the slave device to which the current transaction was addressed. Indicates that an error condition has occurred during the bus cycle. |
| | | | O | Driven by the MPC821 when the internal bus monitor detected an erroneous bus condition. |
| $\overline{\text{BI}}$<br>BURST INHIBIT | 1 | LOW | 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 MPC821 when the slave device is controlled by the on-chip memory controller. |
| ARBITRATION | | | | |
| $\overline{\text{BR}}$<br>BUS REQUEST | 1 | LOW | I | When the internal arbiter is enabled asserted indicates that an external master is requesting the bus. |
| | | | O | Driven by the MPC821 when the internal arbiter is disabled and the chip is not parked. |
| $\overline{\text{BG}}$<br>BUS GRANT | 1 | LOW | O | When the internal arbiter is enabled, the MPC821 asserts this signal to indicate that an external master may assume ownership of the bus and begin a bus transaction. The $\overline{\text{BG}}$ signal should be qualified by the master requesting the bus in order to ensure it is the bus owner:<br>Qualified $\overline{\text{BG}}$ = $\overline{\text{BG}}$ & ~ $\overline{\text{BB}}$ |
| | | | I | When the internal arbiter is disabled, the $\overline{\text{BG}}$ is sampled, and properly qualified, by the MPC821 when an external bus transaction is to be executed by the chip. |

**Table 13-1. MPC821 SIU Signals (Continued)**

| SIGNAL NAME | PINS | ACTIVE | I/O | DESCRIPTION |
|---|---|---|---|---|
| $\overline{\text{BB}}$<br>BUS BUSY | 1 | LOW | O | When the internal arbiter is enabled, the MPC821 asserts this signal to indicate that it is the current owner of the bus. When the internal arbiter is disabled, it will assert this signal after the external arbiter granted to the chip the ownership of the bus and it is ready to start the transaction. |
| | | | I | When the internal arbiter is enabled, the MPC821 samples this signal to get indication of when the external master ended its bus tenure ($\overline{\text{BB}}$ negated).<br>When the internal arbiter is disabled, the $\overline{\text{BB}}$ is sampled, to properly qualify the $\overline{\text{BG}}$ line, when an external bus transaction is to be executed by the chip. |

NOTES: 1. O = Output from the MPC821
2. I = Input to the MPC821
3. A = Central bus arbiter when the internal arbiter of the part is not used
4. T = Bus watchdog timer
5. X = Any device on the bus

## 13.5 BUS OPERATIONS

This section provides a functional description of the system bus, the signals that control it, and the bus cycles provided for data transfer operations. It also describes the error conditions, bus arbitration, and reset operation. The MPC821 generates a system clock output (CLKOUT). This output sets the frequency of operation for the bus interface directly. Internally, the MPC821 uses a phase-lock loop (PLL) circuit to generate a master clock for all of the CPU circuitry (including the bus interface) which is phase-locked to the CLKOUT output signal.

All signals for the MPC821 bus interface are specified with respect to the rising-edge of the external CLKOUT and are guaranteed to be sampled as inputs or changed as outputs with respect to that edge. Since the same clock edge is referenced for driving or sampling the bus signals, the possibility of clock skew could exist between various modules in a system due to routing or the use of multiple clock lines. It is the responsibility of the system to handle any clock skew problems that could occur as a result of layout, lead-length, and physical routing.

### 13.5.1 Basic Transfer Protocol

The basic transfer protocol defines the sequence of actions that must occur on the MPC821 bus to perform a complete bus transaction. A simplified scheme of the basic transfer protocol is illustrated in Figure 13-3.

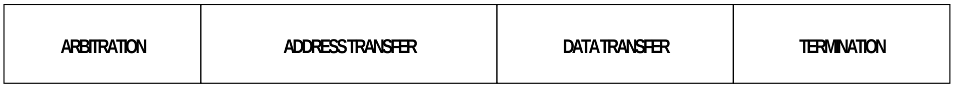


**Figure 13-3. Basic Transfer Protocol**

The basic transfer protocol provides for an arbitration phase and an address and data transfer phase. The address phase specifies the address for the transaction and the transfer attributes that describe the transaction. The data phase performs the transfer of data (if any is to be transferred). The data phase may transfer a single beat of data (4 bytes or less) for nonburst operations, a 4 beat burst of data ($4 \times 4$ bytes), an 8 beat burst of data ($8 \times 2$ bytes), or a 16 beat burst of data ($16 \times 1$ bytes).

### 13.5.2 Single Beat Transfer

During the data transfer phase, the data is transferred from master to slave (in write cycles) or from slave to master (on read cycles). On a write cycle, the master drives the data as soon as it can, but never earlier than the cycle following the address transfer phase. The master has to take into consideration the “one dead clock cycle” switching between drivers to avoid electrical contentions. The master can stop driving the data bus as soon as it samples the $\overline{\text{TA}}$ line asserted on the rising edge of the CLKOUT. On a read cycle the master accepts the data bus contents as valid at the rising edge of the CLKOUT in which the $\overline{\text{TA}}$ signal is sampled asserted.

**13.5.2.1 SINGLE BEAT READ FLOW.** The basic read cycle begins with a bus arbitration, followed by the address transfer, then the data transfer. The handshakes are illustrated in the following flow and timing diagrams as applicable to the fixed transaction protocol.

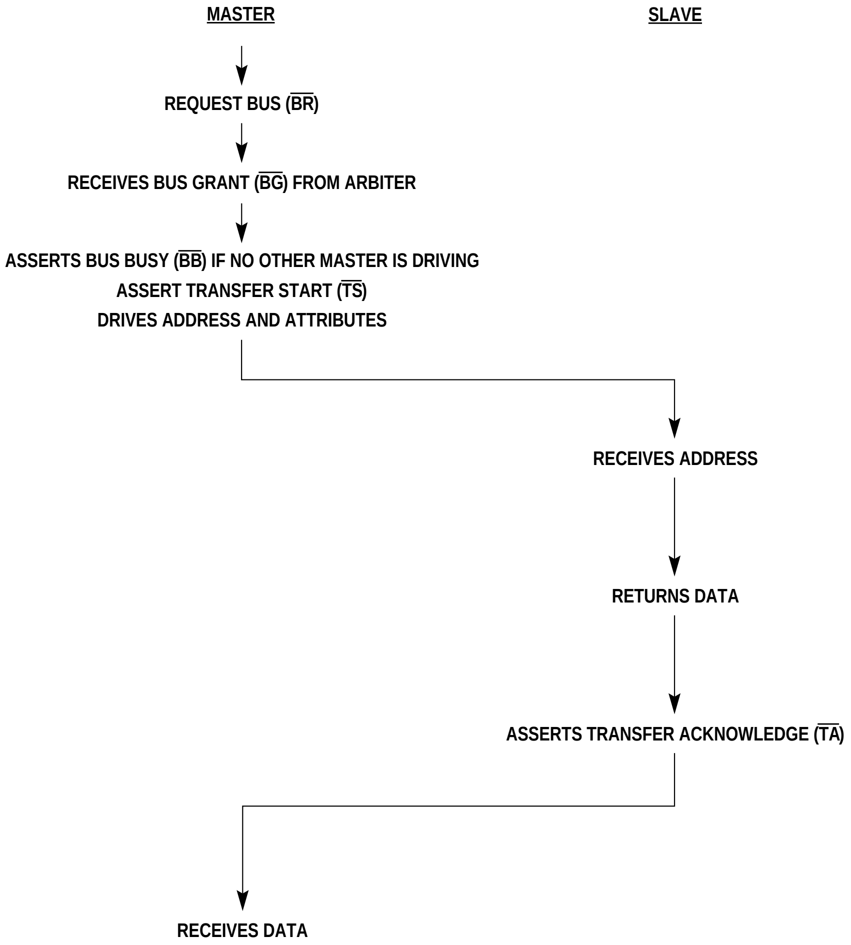

MASTER
SLAVE
REQUEST BUS ($\overline{\text{BR}}$)
RECEIVES BUS GRANT ($\overline{\text{BG}}$) FROM ARBITER
ASSERTS BUS BUSY ($\overline{\text{BB}}$) IF NO OTHER MASTER IS DRIVING
ASSERT TRANSFER START ($\overline{\text{TS}}$)
DRIVES ADDRESS AND ATTRIBUTES
RECEIVES ADDRESS
RETURNS DATA
ASSERTS TRANSFER ACKNOWLEDGE ($\overline{\text{TA}}$)
RECEIVES DATA

**Figure 13-4. Basic Flow Diagram of a Single Beat Read Cycle**

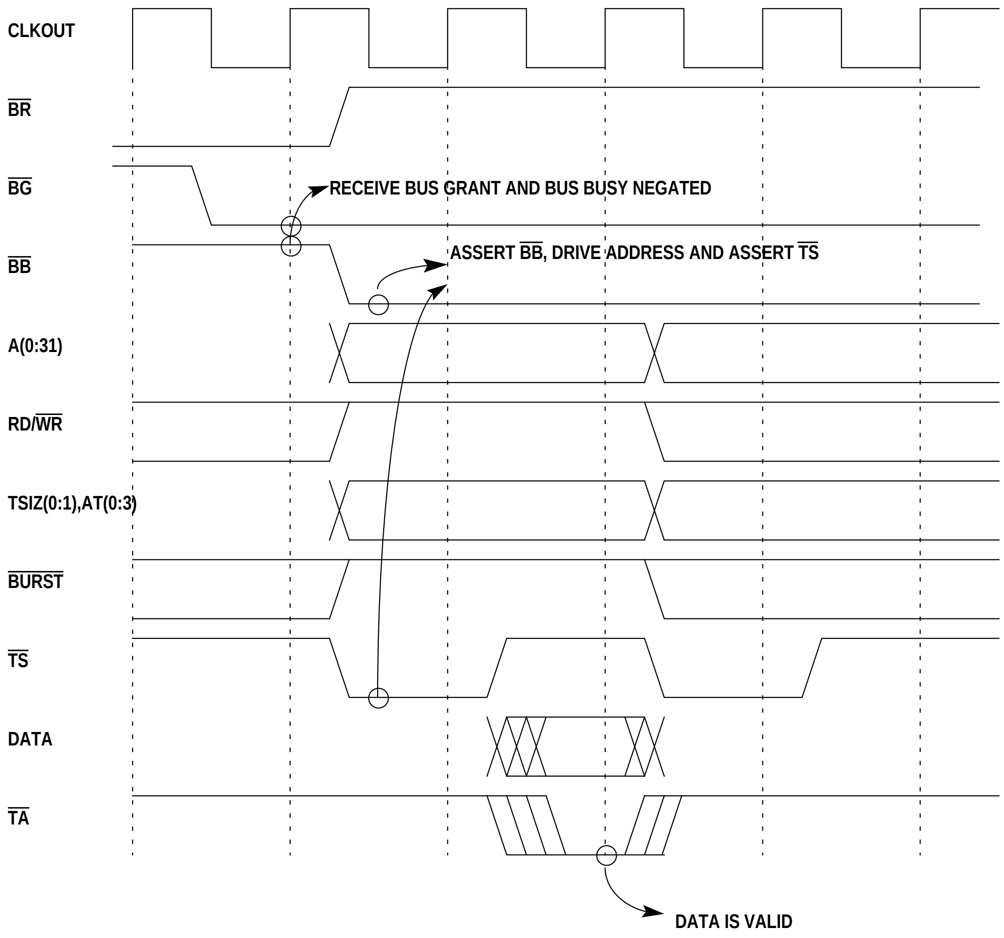

CLKOUT
BR
BG
BB
A(0:31)
RD/WR
TSIZ(0:1),AT(0:3)
BURST
TS
DATA
TA
RECEIVE BUS GRANT AND BUS BUSY NEGATED
ASSERT BB, DRIVE ADDRESS AND ASSERT TS
DATA IS VALID

**Figure 13-5. Single Beat Read Cycle–Basic Timing–Zero Wait States**

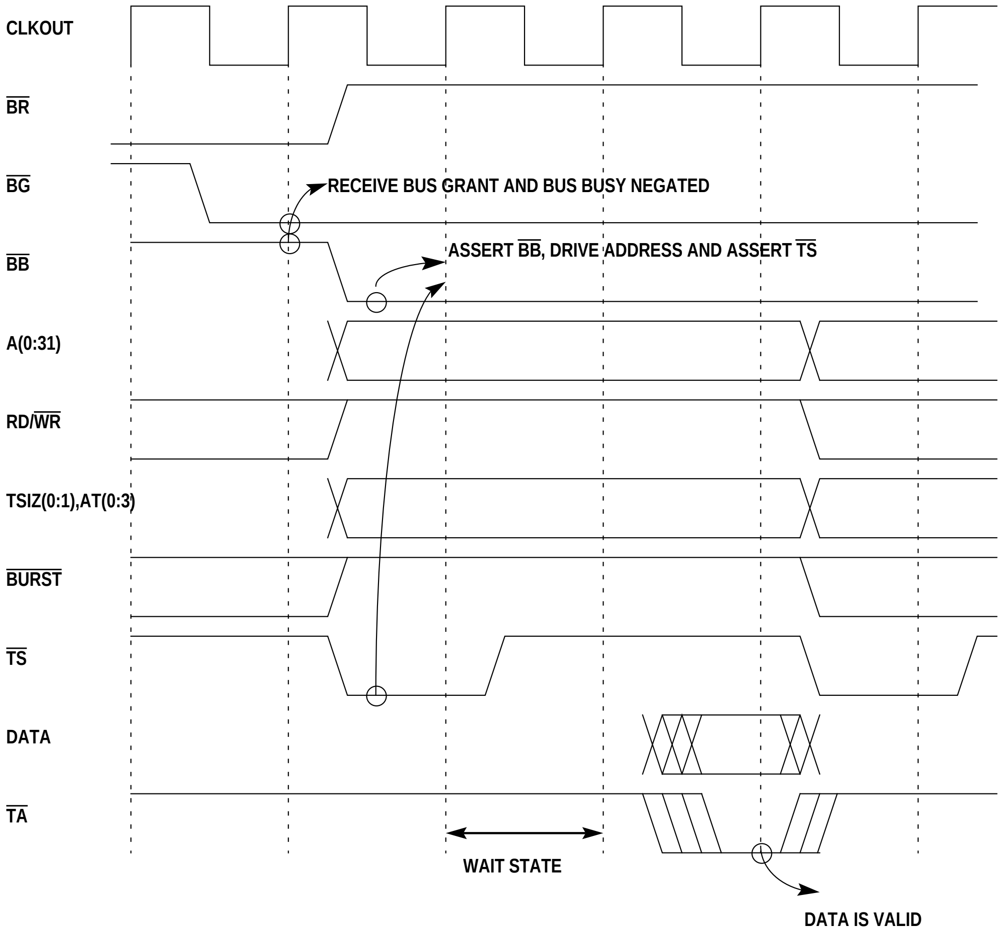


**Figure 13-6. Single Beat Read Cycle–Basic Timing–One Wait State**

**13.5.2.2 SINGLE BEAT WRITE FLOW.** The basic write cycle begins with a bus arbitration, followed by the address transfer, then the data transfer. The handshakes are illustrated in the following flow and timing diagrams as applicable to the fixed transaction protocol.

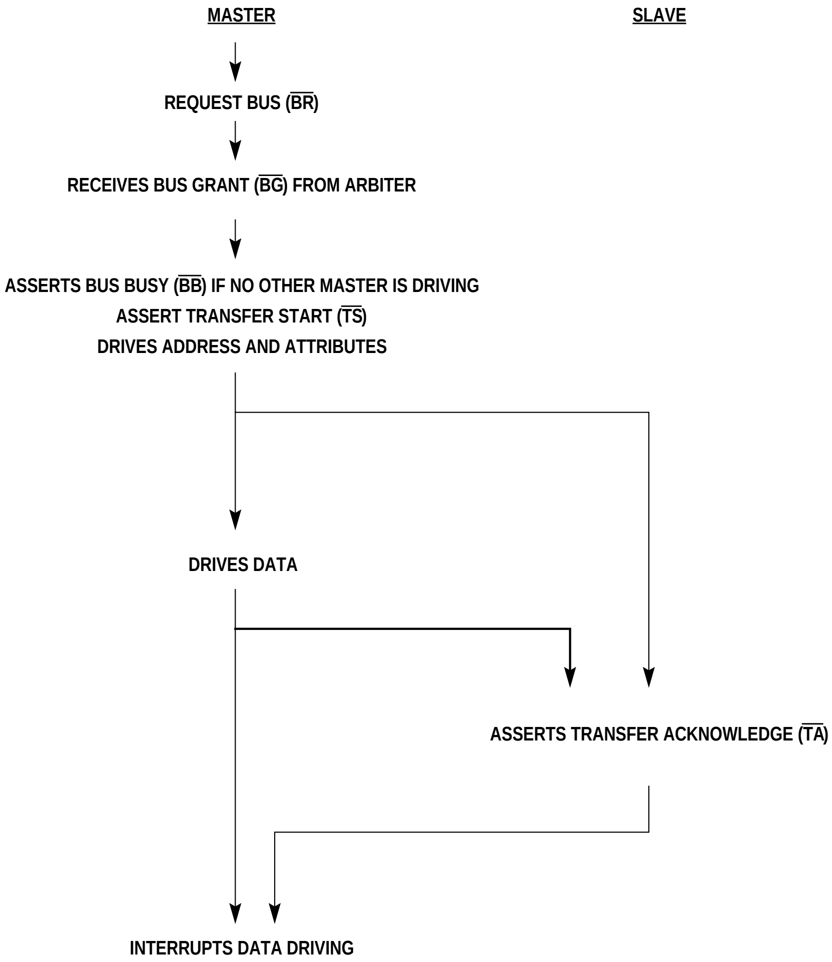

MASTER
SLAVE
REQUEST BUS ($\overline{BR}$)
RECEIVES BUS GRANT ($\overline{BG}$) FROM ARBITER
ASSERTS BUS BUSY ($\overline{BB}$) IF NO OTHER MASTER IS DRIVING
ASSERT TRANSFER START ($\overline{TS}$)
DRIVES ADDRESS AND ATTRIBUTES
DRIVES DATA
ASSERTS TRANSFER ACKNOWLEDGE ($\overline{TA}$)
INTERRUPTS DATA DRIVING

**Figure 13-7. Basic Flow Diagram of a Single Beat Write Cycle**

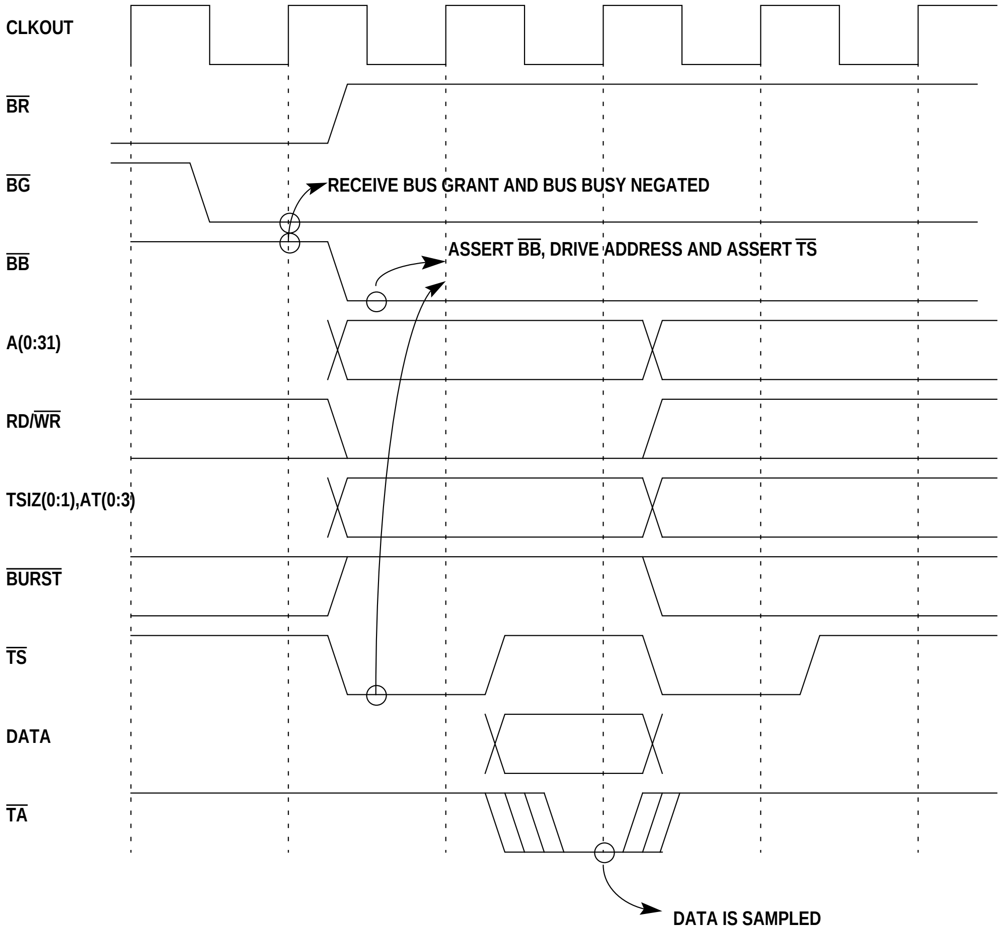

CLKOUT
BR
BG
BB
A(0:31)
RD/WR
TSIZ(0:1),AT(0:3)
BURST
TS
DATA
TA
RECEIVE BUS GRANT AND BUS BUSY NEGATED
ASSERT BB, DRIVE ADDRESS AND ASSERT TS
DATA IS SAMPLED

**Figure 13-8. Single Beat Write Cycle–Basic Timing–Zero Wait States**

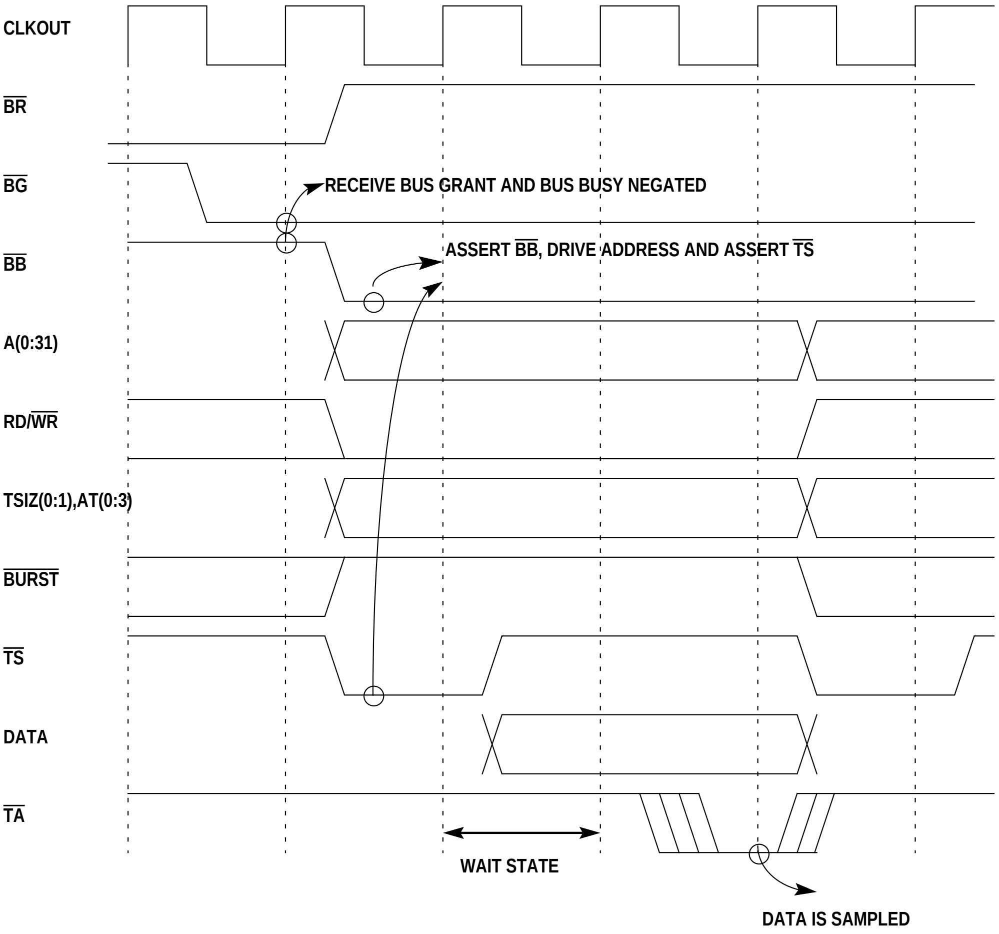


**Figure 13-9. Single Beat Write Cycle–Basic Timing–One Wait State**

The general case of single beat transfers assumes that the external memory has 32-bit port size. The MPC821 provides an effective mechanism for interfacing with 16-bit port size memories and 8-bits port size memories allowing transfers to these devices when they are controlled by the internal memory controller.

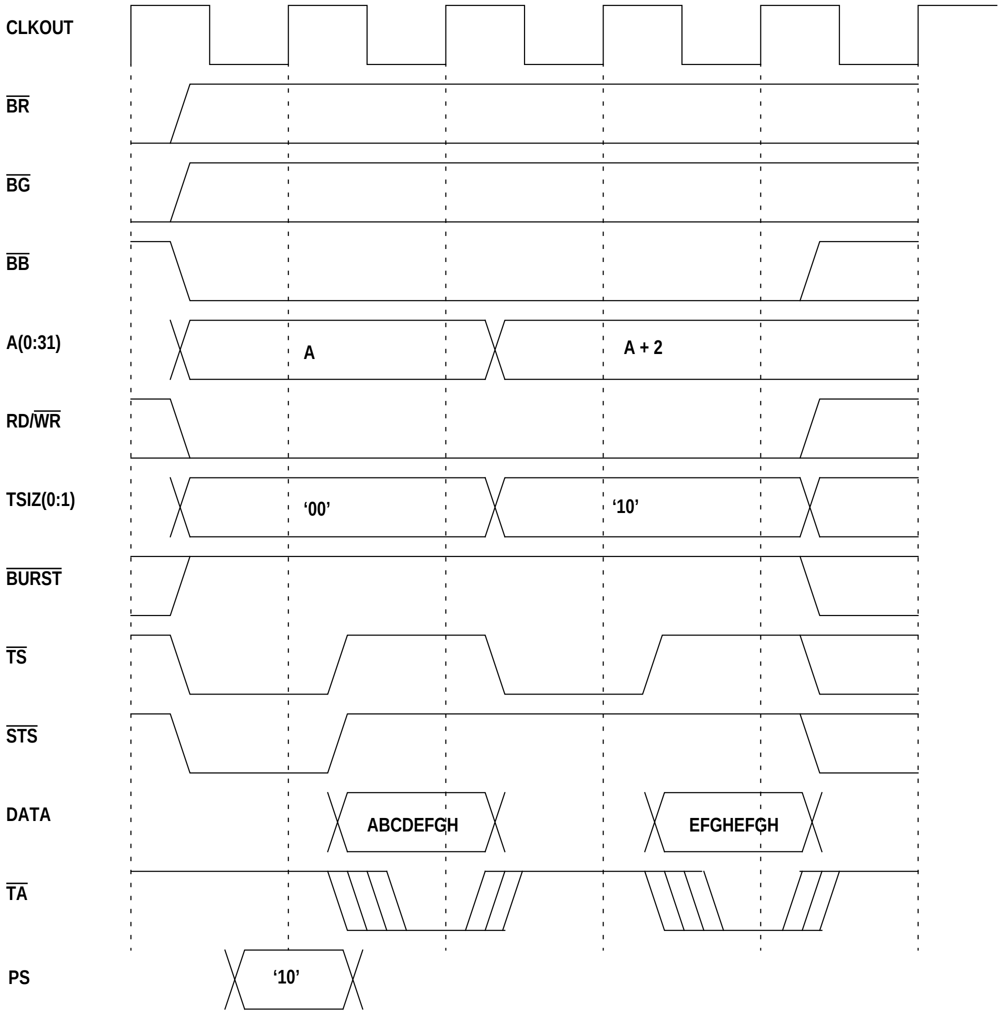


**Figure 13-10. Single Beat–32-Bit Data–Write Cycle–16 Bits Port Size Basic Timing**

## 13.5.3 Burst Transfer

The MPC821 uses burst transfers to access 16-byte operands. A burst accesses a block of 16 bytes that is 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 words block. The address and transfer attributes supplied by the MPC821 remain stable during the transfers and the selected device terminates each transfer by driving/sampling the word on the data bus and asserting $\overline{\text{TA}}$.

The MPC821 also supports burst-inhibited transfers for slave devices that are unable to support bursting. For this type of bus cycle, the selected slave device supplies/samples the first word the MPC821 points to and asserts the burst-inhibit signal with $\overline{\text{TA}}$ for the first transfer of the burst access. The MPC821 responds by terminating the burst and accessing the remainder of the 16-byte block, using three read/write cycles bus (each one for a word) in the case of a 32-bit port width slave, seven read/write cycles bus in the case of a 16-bit port width slave, or fifteen read/write cycles bus in the case of a 8-bit port width slave.

The general case of burst transfers assumes that the external memory has a 32-bit port size. The MPC821 provides an effective mechanism for interfacing with 16-bit port size memories and 8-bit port size memories allowing bursts transfers to these devices when they are controlled by the internal memory controller. In this case, the MPC821 attempts to initiate a burst transfer as in the normal case. If the slave device responds a cycle prior to the transfer acknowledge to the first beat, that it's port size is 16-/8-bits and that the burst is accepted, the MPC821 completes a burst of 8/16 beats. Each of the data beats of the burst transfers effectively only 2/1 bytes. It should be noted that this 8/16-beats burst is considered an atomic transaction, so the MPC821 will not allow other unrelated master accesses or bus arbitration to intervene between the transfers.

### 13.5.4 Burst Mechanism

The MPC821 burst mechanism consists of a signal indicating that the cycle is a burst cycle ($\overline{\text{BURST}}$), another indicating the duration of the burst data (burst data in progress or $\overline{\text{BDIP}}$), and a signal indicating whether the slave is burstable (burst inhibit). These signals are in addition to the basic signals of the bus. At the start of the burst transfer, the master drives the address, the address attributes, and the $\overline{\text{BURST}}$ signal to indicate that a burst transfer is being initiated, along with the assertion of the transfer start signal. If the slave is burstable, it negates the burst-inhibit ($\overline{\text{BI}}$) signal. If the slave cannot burst, it asserts the burstiinhibit signal. During the data phase of a burst write cycle the master drives the data. It also asserts the signal $\overline{\text{BDIP}}$ if it intends to drive the data beat following the current data beat.

When the slave has received the data, it asserts the signal transfer acknowledge to indicate to the master that it is ready for the next data transfer. The master again drives the next data and asserts or negates the $\overline{\text{BDIP}}$ signal. If the master does not intend to drive another data beat following the current one, it negates the $\overline{\text{BDIP}}$ to indicate to the slave that the next subsequent data beat transfer is the last data of the burst write transfer. During the data phase of a burst read cycle, the master receives data from the addressed slave. If the master needs more than one data, it asserts the signal $\overline{\text{BDIP}}$. When the data is received prior to the last data, the master deasserts the signal $\overline{\text{BDIP}}$. Thus, the slave stops driving new data after it received the negation of the $\overline{\text{BDIP}}$ signal at the rising edge of the clock.

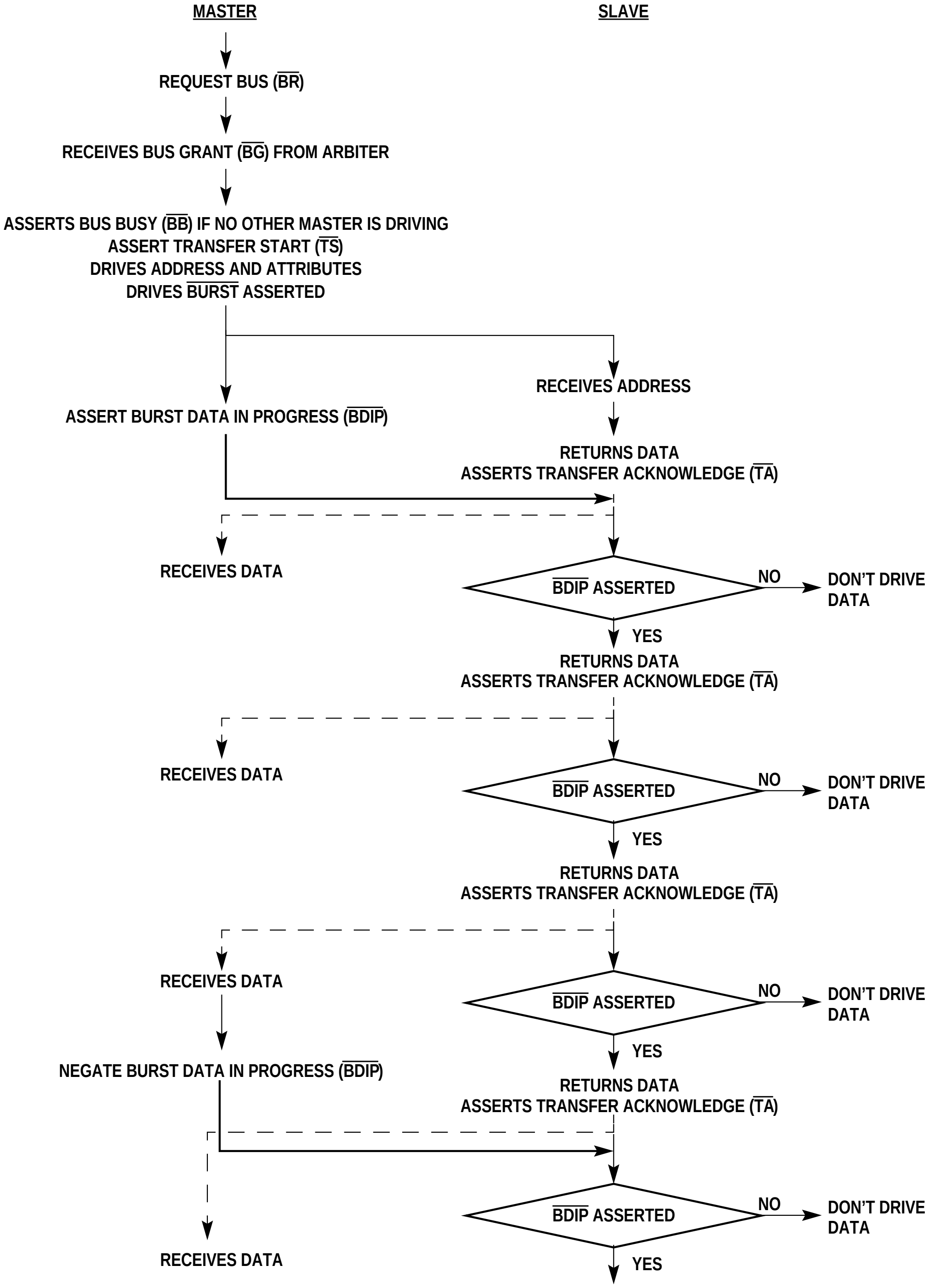

MASTER
SLAVE
REQUEST BUS (BR)
RECEIVES BUS GRANT (BG) FROM ARBITER
ASSERTS BUS BUSY (BB) IF NO OTHER MASTER IS DRIVING
ASSERT TRANSFER START (TS)
DRIVES ADDRESS AND ATTRIBUTES
DRIVES BURST ASSERTED
RECEIVES ADDRESS
ASSERT BURST DATA IN PROGRESS (BDIP)
RETURNS DATA
ASSERTS TRANSFER ACKNOWLEDGE (TA)
RECEIVES DATA
BDIP ASSERTED
NO
DON'T DRIVE DATA
YES
RETURNS DATA
ASSERTS TRANSFER ACKNOWLEDGE (TA)
RECEIVES DATA
BDIP ASSERTED
NO
DON'T DRIVE DATA
YES
RETURNS DATA
ASSERTS TRANSFER ACKNOWLEDGE (TA)
RECEIVES DATA
BDIP ASSERTED
NO
DON'T DRIVE DATA
NEGATE BURST DATA IN PROGRESS (BDIP)
YES
RETURNS DATA
ASSERTS TRANSFER ACKNOWLEDGE (TA)
BDIP ASSERTED
NO
DON'T DRIVE DATA
RECEIVES DATA
YES

**Figure 13-11. Basic Flow Diagram Of A Burst Read Cycle**

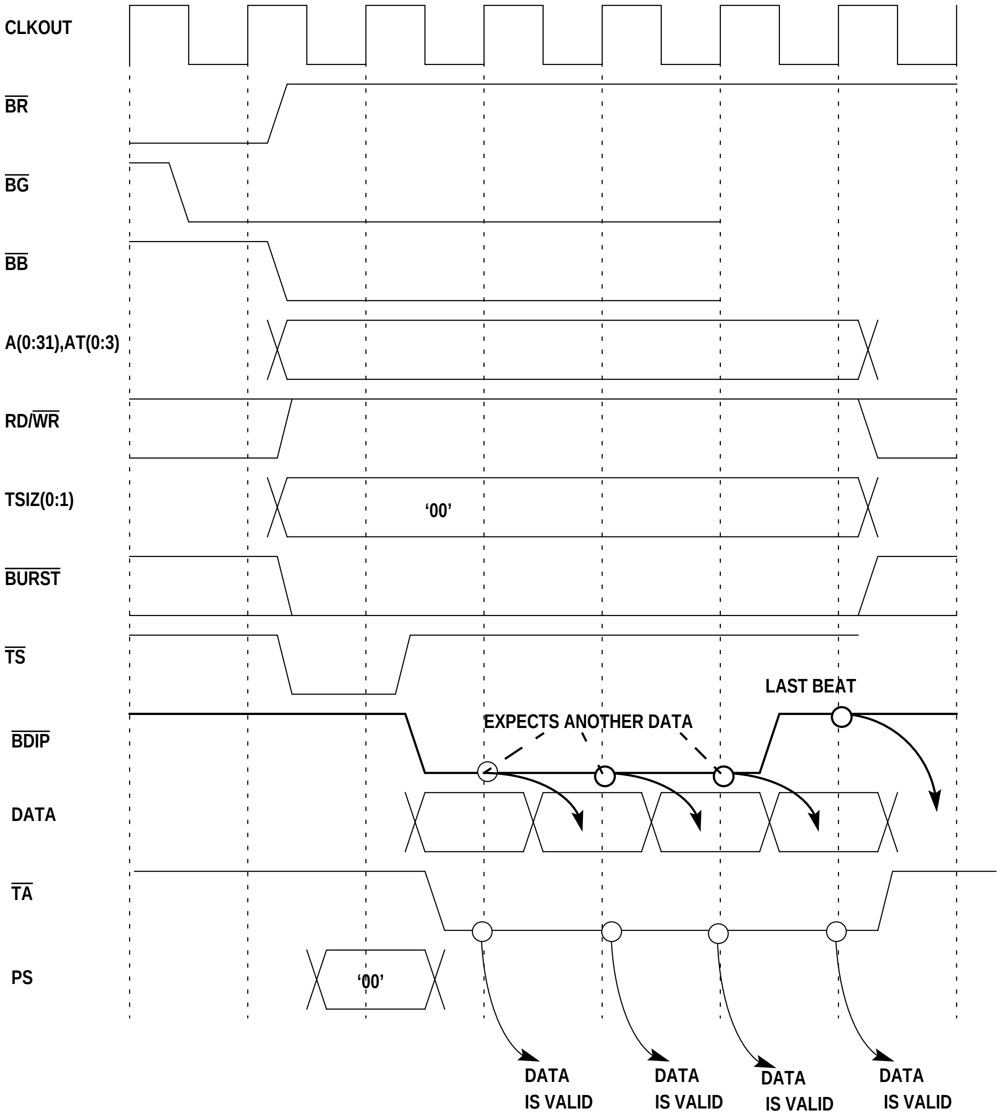


**Figure 13-12. Burst-Read Cycle–32-Bit Port Size–Zero Wait State**

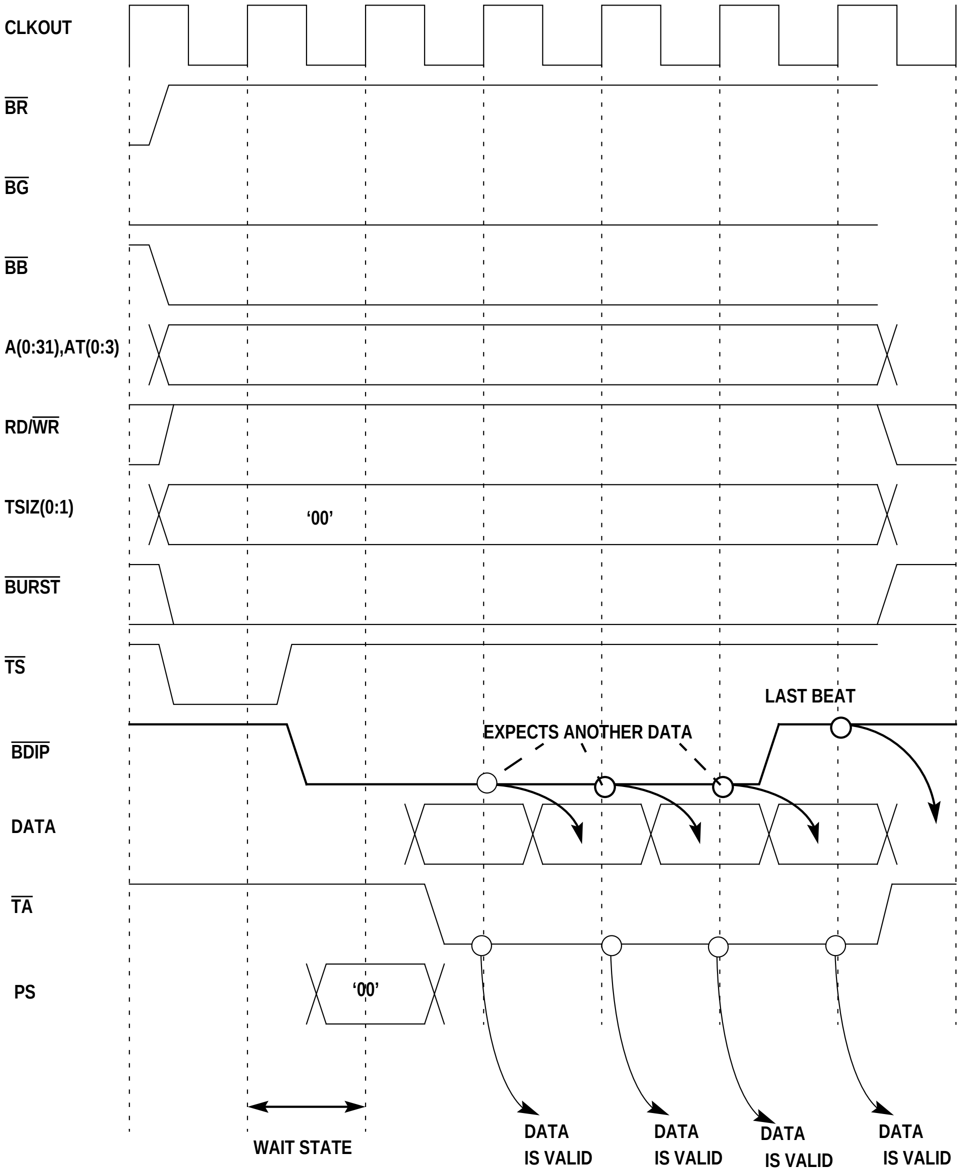


**Figure 13-13. Burst-Read Cycle–32-Bit Port Size–One Wait State**

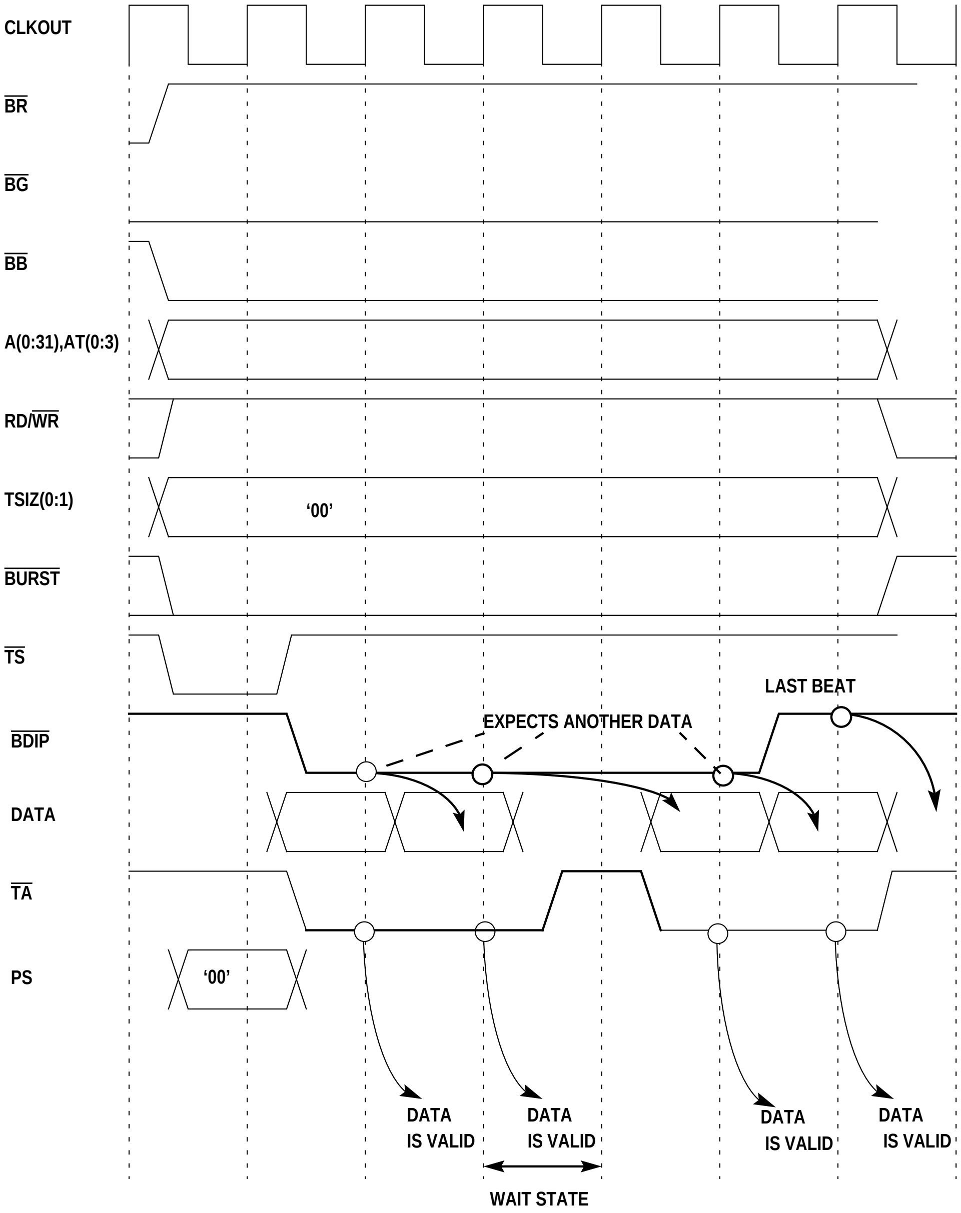


**Figure 13-14. Burst-Read Cycle–32-Bit Port Size–Wait States Between Beats**

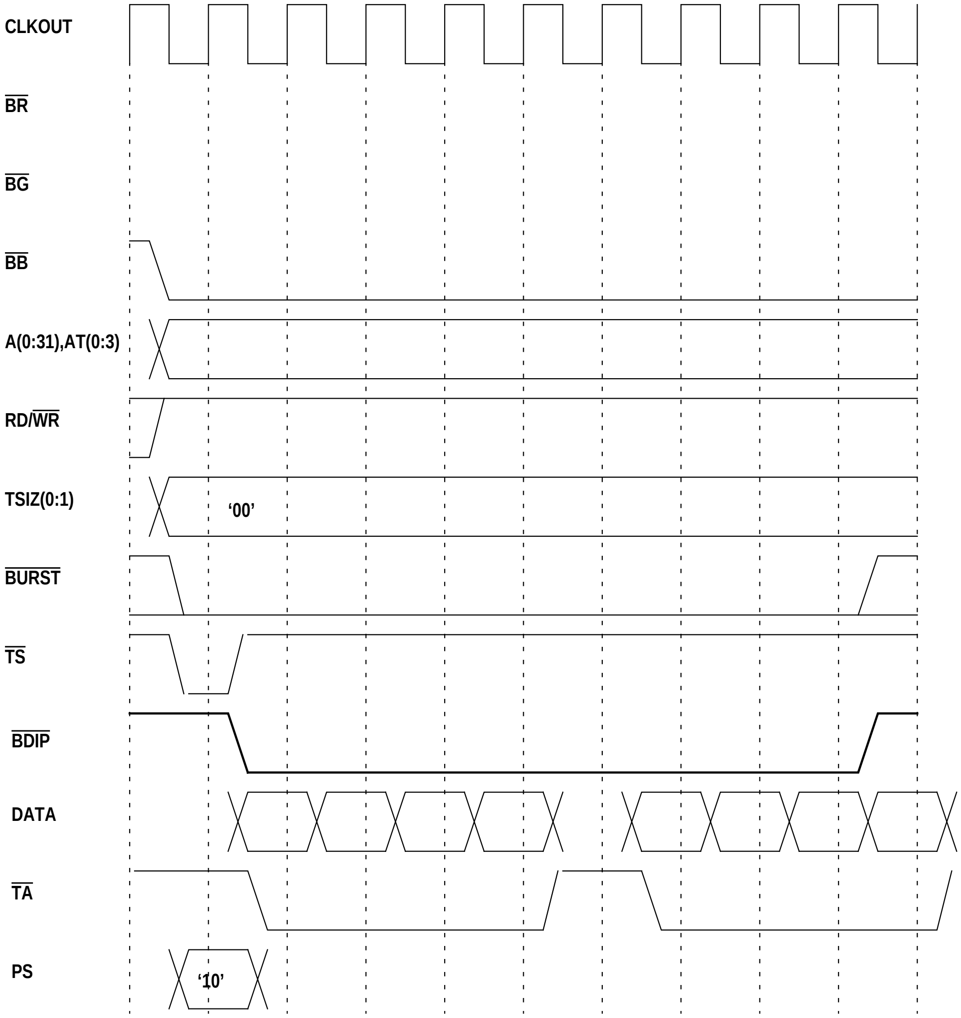


**Figure 13-15. Burst-Read Cycle–16-Bit Port Size–One Wait State Between Beats**

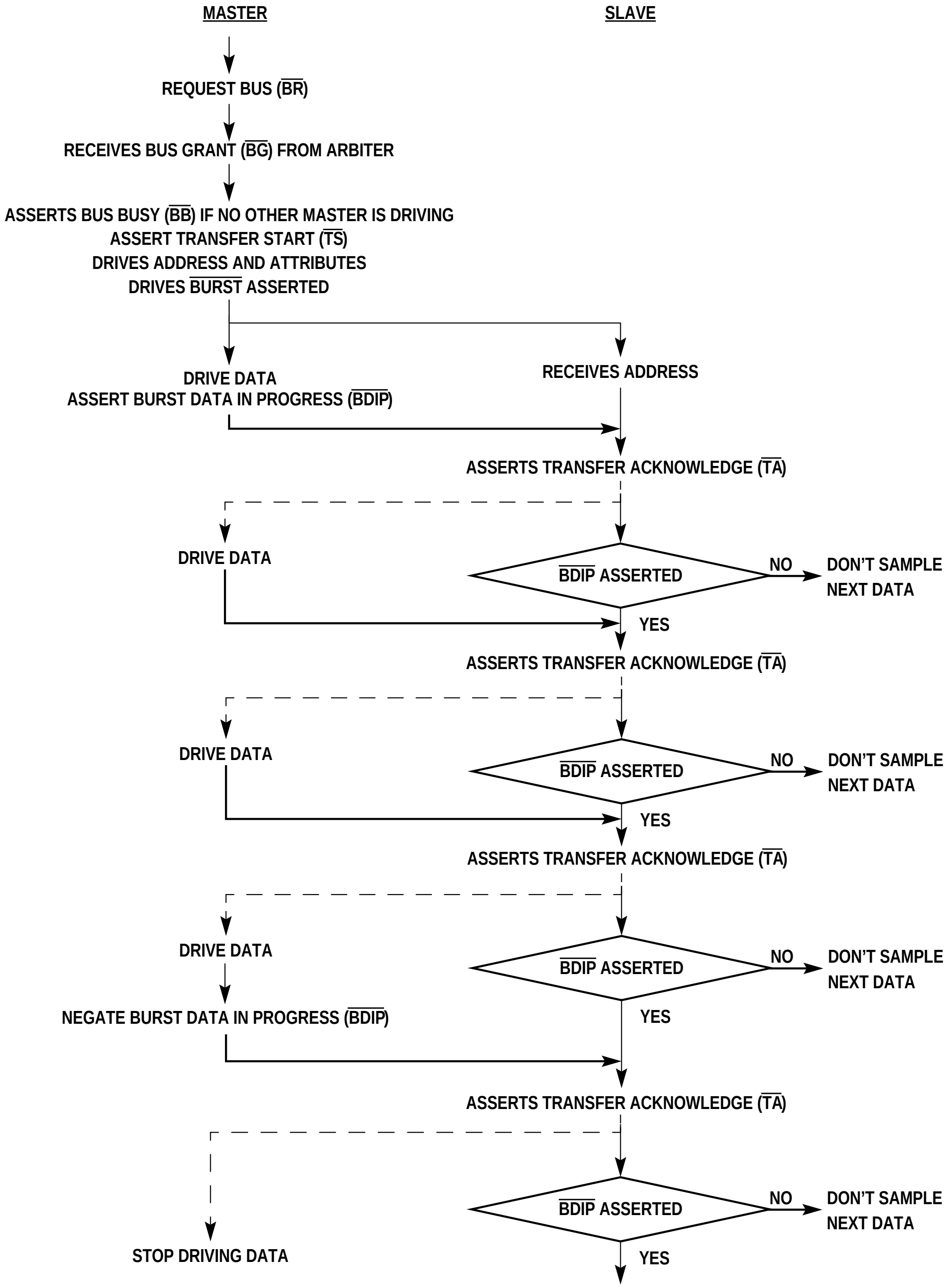

MASTER
SLAVE
REQUEST BUS (BR)
RECEIVES BUS GRANT (BG) FROM ARBITER
ASSERTS BUS BUSY (BB) IF NO OTHER MASTER IS DRIVING
ASSERT TRANSFER START (TS)
DRIVES ADDRESS AND ATTRIBUTES
DRIVES BURST ASSERTED
DRIVE DATA
ASSERT BURST DATA IN PROGRESS (BDIP)
RECEIVES ADDRESS
ASSERTS TRANSFER ACKNOWLEDGE (TA)
DRIVE DATA
BDIP ASSERTED
NO
DON'T SAMPLE
NEXT DATA
YES
ASSERTS TRANSFER ACKNOWLEDGE (TA)
DRIVE DATA
BDIP ASSERTED
NO
DON'T SAMPLE
NEXT DATA
YES
ASSERTS TRANSFER ACKNOWLEDGE (TA)
DRIVE DATA
BDIP ASSERTED
NO
DON'T SAMPLE
NEXT DATA
NEGATE BURST DATA IN PROGRESS (BDIP)
YES
ASSERTS TRANSFER ACKNOWLEDGE (TA)
BDIP ASSERTED
NO
DON'T SAMPLE
NEXT DATA
STOP DRIVING DATA
YES

**Figure 13-16. Basic Flow Diagram of a Burst Write Cycle**

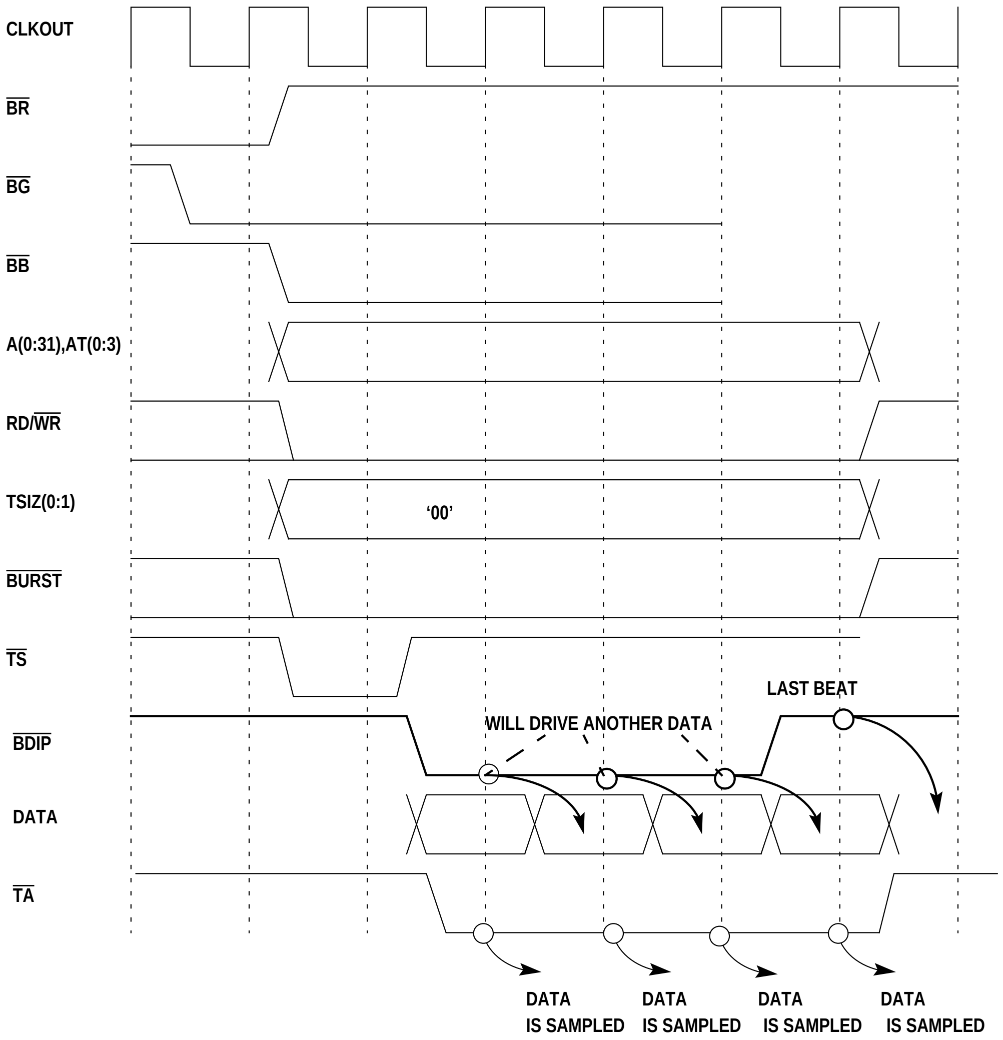


**Figure 13-17. Burst-Write Cycle–32-Bit Port Size–Zero Wait States**

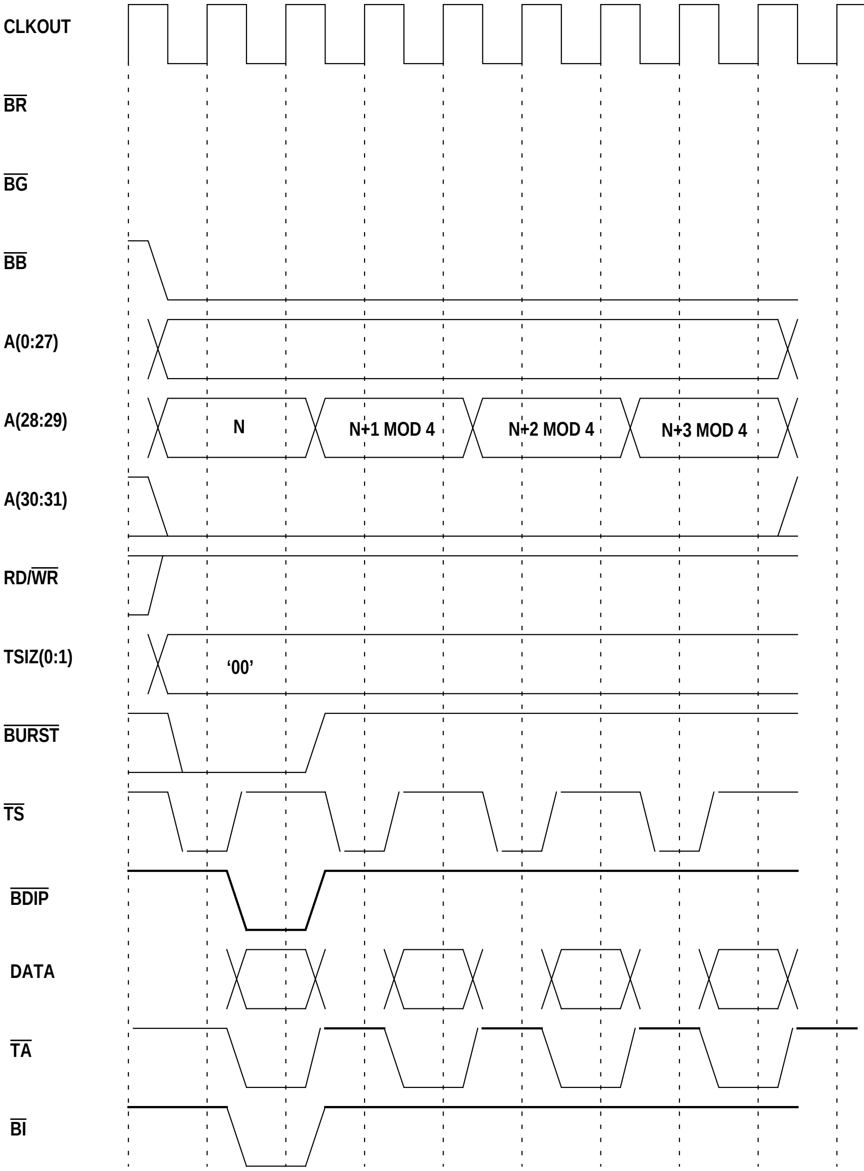


Figure 13-18. Burst-Inhibit Cycle–32-Bit Port Size

### 13.5.5 Alignment and Packaging on Transfers

The MPC821 external bus only supports natural address alignment:

- Byte access can have any address alignment.
- Half-word access must have address bit 31equal to 0.
- Word access must have address 30-31 equa to 0.
- For burst access must have address 30-31 equal to 0.

The MPC821 is able to perform operand transfers through its 32-bit data port. If the transfer is controlled by the internal memory controller, the MPC821 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 data bus bits 0-31, a 16-bit port must reside on bits 0-15, and an 8-bit port must reside on bits 0-7. The MPC821 always tries to transfer the maximum amount of data on all bus cycles and for a word operation, it always assumes that the port is 32 bits wide when beginning the bus cycle. In Figure 13-19 and Figure 13-20 and Table 13-2 and Table 13-3, the following conventions are adopted:

- OP0 is the most-significant byte of a word operand and OP3 is the least-significant byte.
- The two bytes of a half-word operand are OP0 (most-significant) and OP1 or OP2 (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.

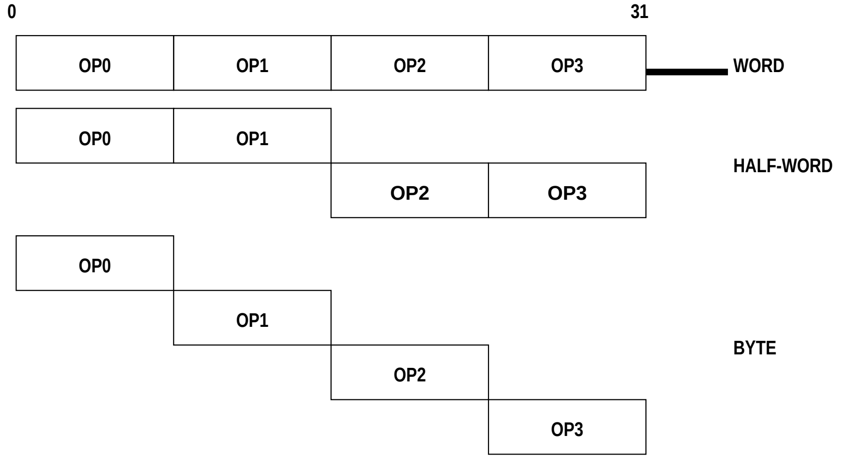


**Figure 13-19. Internal Operand Representation**

Figure 13-20 illustrates the device connections on the data bus.

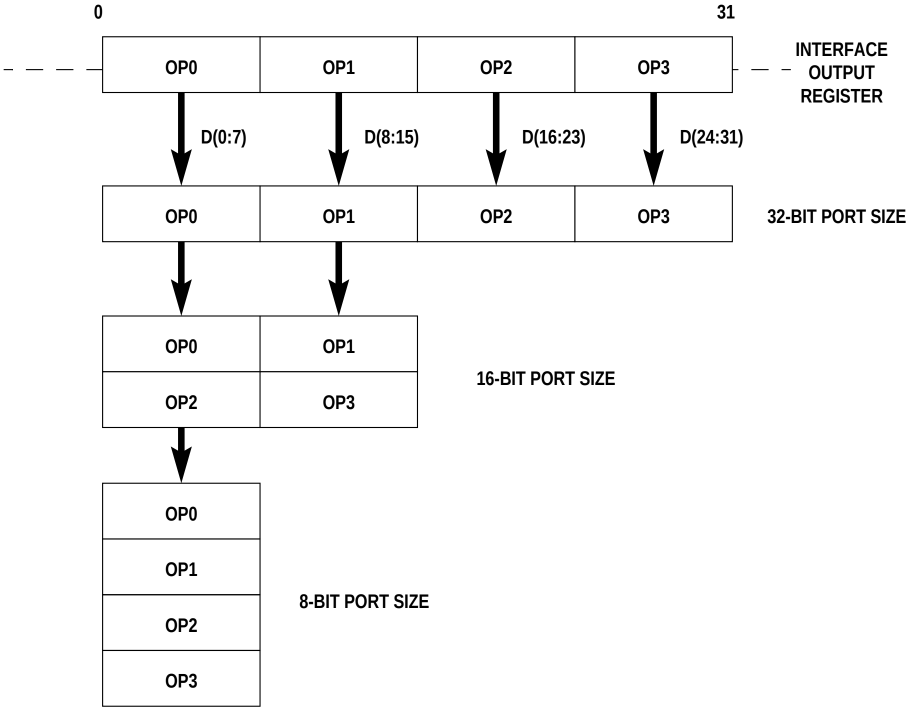


**Figure 13-20. Interface To Different Port Size Devices**

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 | TSIZE | | ADDRESS | | 32-BIT PORT SIZE | | | | 16-BIT PORT SIZE | | 8-BIT PORT SIZE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | A30 | A31 | D0:D7 | D8:D15 | D16:D23 | D24:D31 | D0:D7 | D8:D15 | D0:D7 |
| Byte | 0 | 1 | 0 | 0 | OP0 | — | — | — | OP0 | — | OP0 |
| | 0 | 1 | 0 | 1 | — | OP1 | — | — | — | OP1 | OP1 |
| | 0 | 1 | 1 | 0 | — | — | OP2 | — | OP2 | — | OP2 |
| | 0 | 1 | 1 | 1 | — | — | — | OP3 | — | OP3 | OP3 |
| Half-Word | 1 | 0 | 0 | 0 | OP0 | OP1 | — | — | OP0 | OP1 | OP0 |
| | 1 | 0 | 1 | 0 | — | — | OP2 | OP3 | OP2 | OP3 | OP2 |
| Word | 0 | 0 | 0 | 0 | OP0 | OP1 | OP2 | OP3 | OP0 | OP1 | OP0 |

NOTE: — denotes a byte not required during that read cycle.

Table 13-3 lists the patterns of the data transfer for write cycles when accesses are initiated by the MPC821.

**Table 13-3. Data Bus Contents for Write Cycles**

| TRANSFER SIZE | TSIZE | | ADDRESS | | EXTERNAL DATA BUS PATTERN | | | |
|---|---|---|---|---|---|---|---|---|
| | | | A30 | A31 | D0:D7 | D8:D15 | D16:D23 | D24:D31 |
| Byte | 0 | 1 | 0 | 0 | OP0 | — | — | — |
| | 0 | 1 | 0 | 1 | OP1 | OP1 | — | — |
| | 0 | 1 | 1 | 0 | OP2 | — | OP2 | — |
| | 0 | 1 | 1 | 1 | OP3 | OP3 | — | OP3 |
| Half-Word | 1 | 0 | 0 | 0 | OP0 | OP1 | — | — |
| | 1 | 0 | 1 | 0 | OP2 | OP3 | OP2 | OP3 |
| Word | 0 | 0 | 0 | 0 | OP0 | OP1 | OP2 | OP3 |

NOTE: — denotes a byte not required during that read cycle.

### 13.5.6 Arbitration Phase

The external bus design provides for a single bus master at any one time, either the MPC821 or an external device. One or more of the external devices on the bus can have the capability of becoming bus master for the external bus. Bus arbitration may be handled either by an external central bus arbiter or by the internal on-chip arbiter. In the latter case, the system is optimized for one external bus master besides the MPC821. The arbitration configuration (external or internal) is set at system reset. See **Section 15.5 Memory Controller External Master Support** for more information.

Each bus master must have bus request, bus grant, and bus busy signals. The device that needs the bus asserts the bus request ($\overline{\text{BR}}$) signal. The device then waits for the arbiter to assert bus grant ($\overline{\text{BG}}$) signal. In addition, the new master must look at the bus busy ($\overline{\text{BB}}$) signal to ensure that no other master is driving the bus before it can assert bus busy to assume ownership of the bus. At any time the arbiter has taken the bus grant away from the master and the master wants to execute a new cycle, the master must rearbitrate before a new cycle can be accomplished. The MPC821, however, guarantees data coherency for access to a small port size and for decomposed bursts. This means that the MPC821 will not release the bus before the completion of the transactions that are considered atomic. Figure 13-21 describes the basic protocol for bus arbitration. See **Section 12.4.1.1 SIU Module Configuration Register** for details.

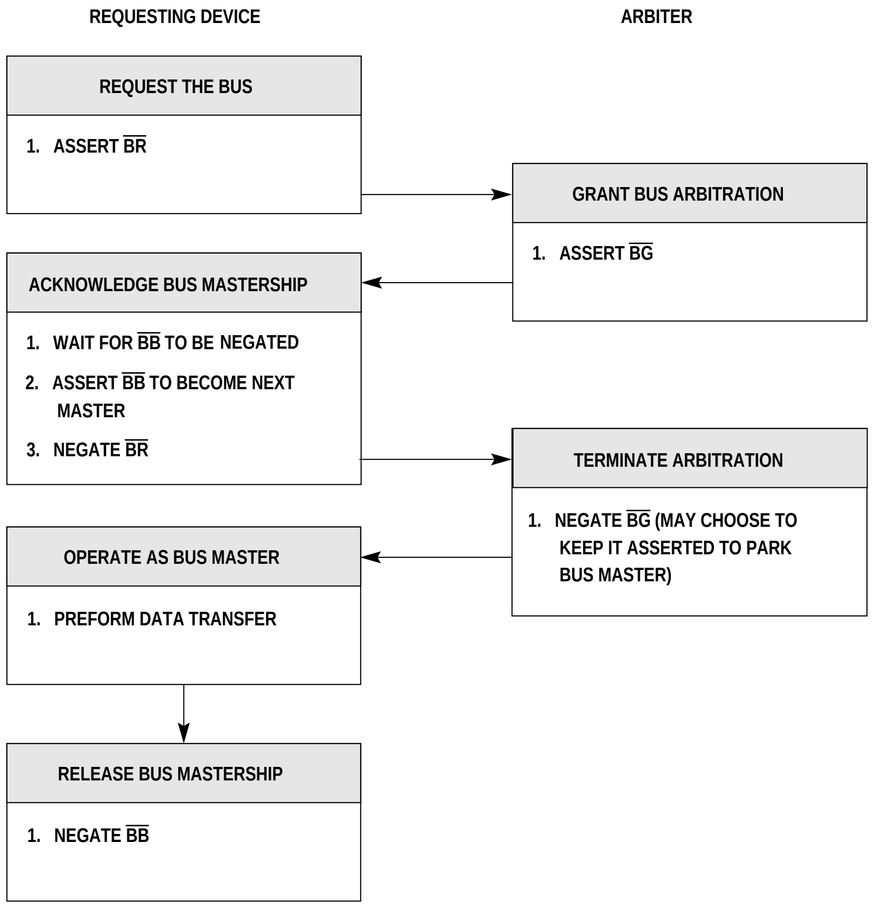

REQUESTING DEVICE
ARBITER
REQUEST THE BUS
1. ASSERT $\overline{\text{BR}}$
GRANT BUS ARBITRATION
1. ASSERT $\overline{\text{BG}}$
ACKNOWLEDGE BUS MASTERSHIP
1. WAIT FOR $\overline{\text{BB}}$ TO BE NEGATED
2. ASSERT $\overline{\text{BB}}$ TO BECOME NEXT MASTER
3. NEGATE $\overline{\text{BR}}$
TERMINATE ARBITRATION
1. NEGATE $\overline{\text{BG}}$ (MAY CHOOSE TO KEEP IT ASSERTED TO PARK BUS MASTER)
OPERATE AS BUS MASTER
1. PREFORM DATA TRANSFER
RELEASE BUS MASTERSHIP
1. NEGATE $\overline{\text{BB}}$

**Figure 13-21. Bus Arbitration Flowchart**

**13.5.6.1 BUS REQUEST.** The potential bus master asserts $\overline{\text{BR}}$ to request bus mastership. $\overline{\text{BR}}$ should be negated as soon as the bus is granted, the bus is not busy, and the new master can drive the bus. If more request is pending, the master can keep asserting it's bus request as long as needed. When configured for external central arbitration, the MPC821 drives this signal when it requires bus mastership. When the internal on-chip arbiter is used, this signal is an input to the internal arbiter and should be driven by the external bus master.

**13.5.6.2 BUS GRANT.** $\overline{\text{BG}}$ is asserted by the arbiter to indicate that the bus is granted to the requesting device. This signal can be negated following the negation of $\overline{\text{BR}}$ or kept asserted for the current master to park the bus. When configured for external central arbitration, this is an input signal to the MPC821 from the external arbiter. When the internal on-chip arbiter is used, this signal is an output from the internal arbiter to the external bus master.

**13.5.6.3 BUS BUSY.** $\overline{\text{BB}}$ indicates that the current bus master is using the bus. New masters should not begin transfer until this signal is deasserted. The bus owner should not relinquish or negate this signal until it's transfer is complete. To avoid contention on the $\overline{\text{BB}}$ line, masters should three-state this signal when it gets a logical '1' value. This situation implies the connection of an external pull-up resistor is needed to ensure that a master that acquires the bus is able to recognize the $\overline{\text{BB}}$ line negated, regardless of how many cycles have passed since the previous master relinquished the bus. Refer to Figure 13-22 for more information.

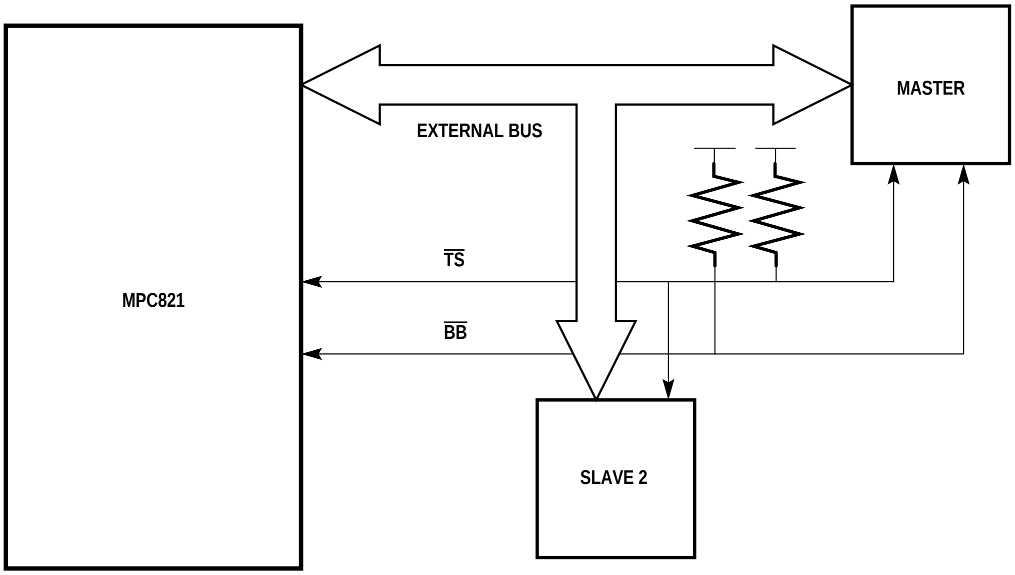


**Figure 13-22. Masters Signals Basic Connection**

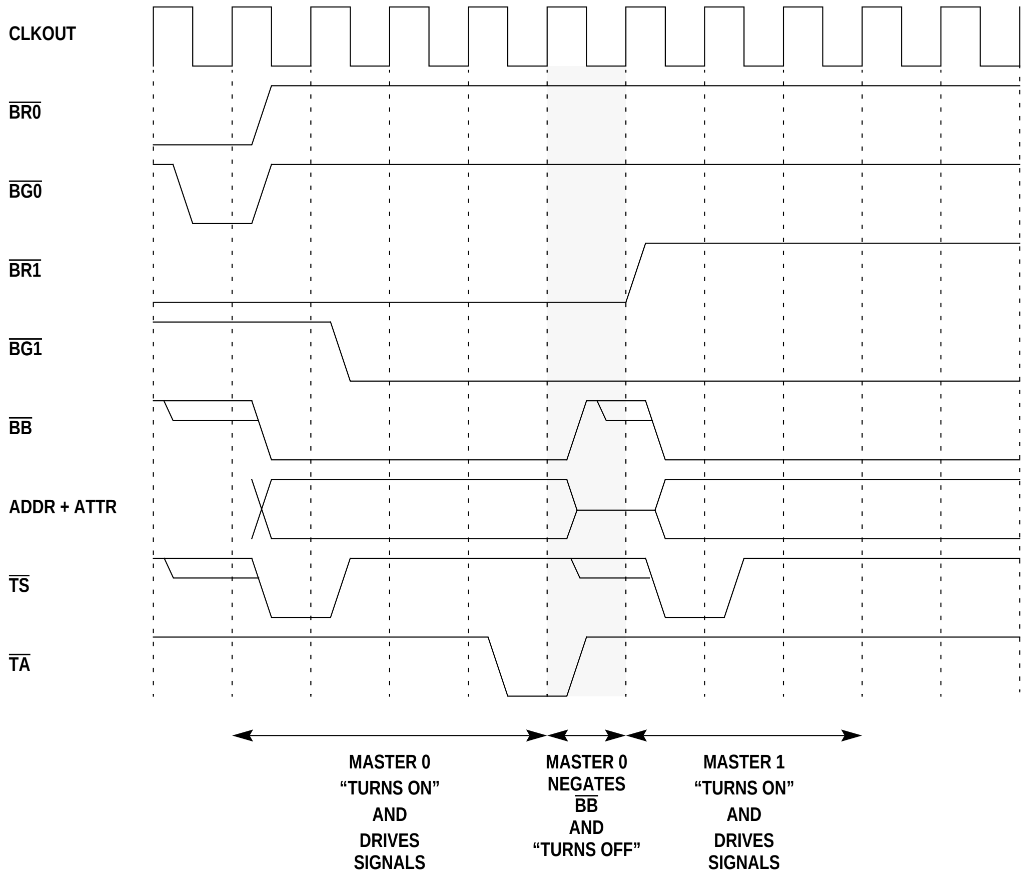

CLKOUT
BR0
BG0
BR1
BG1
BB
ADDR + ATTR
TS
TA
MASTER 0 "TURNS ON" AND DRIVES SIGNALS
MASTER 0 NEGATES BB AND "TURNS OFF"
MASTER 1 "TURNS ON" AND DRIVES SIGNALS

**Figure 13-23. Bus Arbitration Timing Diagram**

The MPC821 can be configured at system reset to use the internal bus arbiter. In this case, the MPC821 will be parked on the bus. The priority of the external device relative to the internal MPC821 bus masters is programmed in the SIU module configuration register. See **Section 12.4.1.1 SIU Module Configuration Register**. If the external device requests the bus and the MPC821 does not require it, or the external device has higher priority than the current internal bus master, the MPC821 grants the bus to the external device.
Figure 13-24 illustrates the internal finite state machine that implements the arbiter protocol.

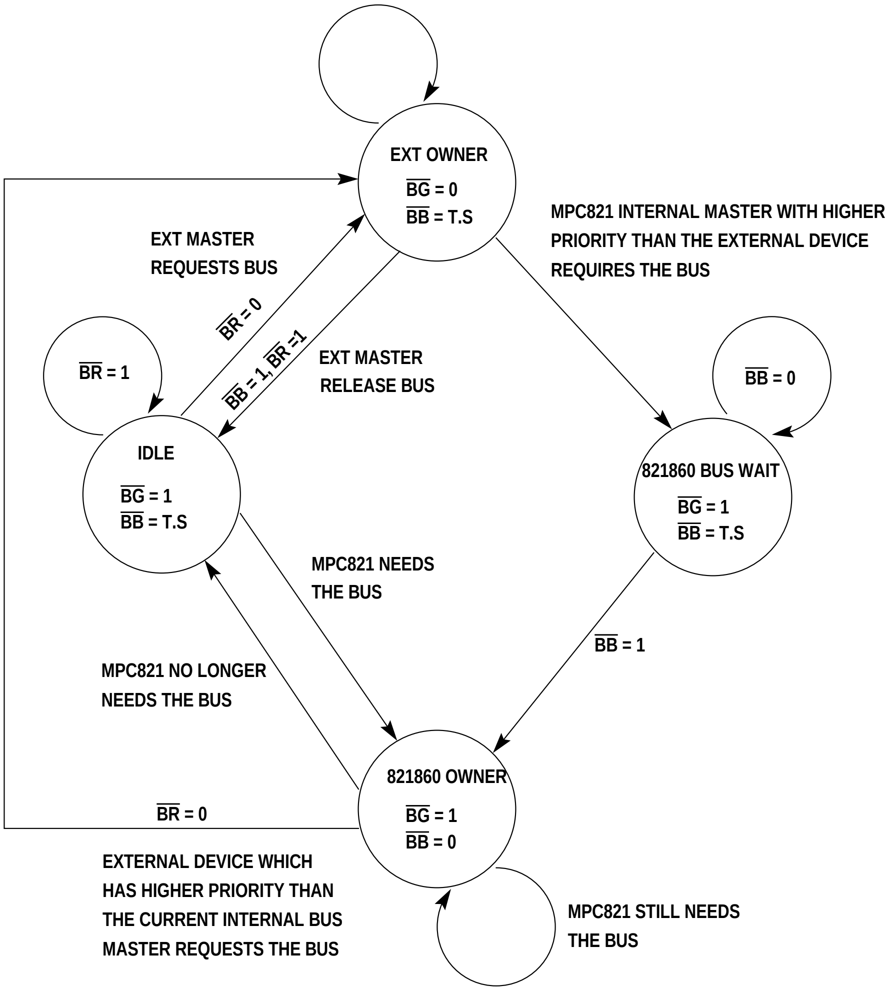


**Figure 13-24. Internal Bus Arbitration State Machine**

## 13.5.7 Address Transfer Phase-Related Signals

**13.5.7.1 TRANSFER START.** This signal ($\overline{TS}$) indicates the beginning of a transaction on the bus addressing a slave device. This signal should be asserted by a master only after the ownership of the bus was granted by the arbitration protocol. This signal is only asserted for the first cycle of the transaction and is negated in the successive clock cycles until the end of the transaction. The master should three-state this signal when it relinquishes the bus to avoid contention between two or more masters in this line. This situation indicates that an external pull-up resistor should be connected to the $\overline{TS}$ signal to avoid having a slave recognize this asserted signal when no master drives it. Refer to Figure 13-22 for more information.

**13.5.7.2 ADDRESS BUS.** The address bus is 32-bit and consists of address bits 0 to 31. Address bit 0 is the most-significant bit. The bus is byte addressable, so each address can address one or more bytes. The address and its attributes are driven on the bus with the transfer start signal and keep valid until the bus master received signal transfer acknowledge from the slave. To distinguish the individual byte, the slave device has to observe the TSIZ signals.

**13.5.7.3 TRANSFER ATTRIBUTES.** The transfer attributes signal comprehends the RD/WR, BURST, TSIZ(0:1), AT(0:3), STS, and BDIP signals. These signals (with the exception of the BDIP) are available at the same time as the address bus.

**13.5.7.3.1 Read/Write.** RD/WR high indicates a read access and low indicates a write access.

**13.5.7.3.2 Burst Indicator.** 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. The burst size is always fix (16 bytes long). 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 MPC821-bus supports critical data first access for fixed-size burst. The order of wraparound wraps back to the data 0. For example:

- Case burst of four:

  data 0 → data 1 → data 2 → data 3 → data 0

- Case burst of eight:

  data 0 → data 1 → data 2 → ......... → data 6 → data 7 → data 0

**13.5.7.3.3 Transfer Size.** TSIZ(0:1) indicates the size of the requested data transfer. The TSIZ signals may be used with BURST and A(30:31) to determine which byte lanes of the data bus are involved in the transfer. For nonburst transfers, the TSIZ signals specify the number of bytes starting from the byte location addressed by A(30:31). In burst transfers, the value of TSIZ is always 00.

**Table 13-4. BURST/TSIZE Encoding**

| BURST | TSIZ(0:1) | TRANSFER SIZE |
|---|---|---|
| N | 01 | Byte |
| N | 10 | Half-Word |
| N | 11 | x |
| N | 00 | Word |
| A | 00 | Burst (16 bytes) |

**13.5.7.3.4 Address Types.** The address type signals (AT0-AT3), PTR and 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, problem/privilege (user or supervisor), and instruction/data types. The address type signals are valid at the rising edge of the clock in which the special transfer start (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 copy back of a dirty line in the data cache occurs after the privilege state of the processor has been changed since the last access to the same line. A functional usage of the address type signals, PTR and RSV, is for the reservation protocol described in **Section 13.5.9 Storage Reservation**. Table 13-5 provides the space definition encoded by the STS, TS, AT(0:3), PTR, and RSV signals.

Table 13-5. Address Types Definition

| s$\overline{TS}$ | $\overline{TS}$ | AT(0) | AT(1) | AT(2) | AT(3) | PTR | RSV | ADDRESS SPACE DEFINITIONS |
|---|---|---|---|---|---|---|---|---|
| | | CPU/CPM | PROBLEM STATE/<br>PRIVILEGE STATE | INSTRUCTION/<br>DATA | RESERVATION/<br>PROGRAM<br>TRACE | PROGRAM<br>TRACE | RESERVATION | |
| 1 | x | x | x | x | x | 1 | 1 | No Transfer or no first transaction of a transfer |
| 0 | x | x | x | x | x | x | x | Start of a transaction |
| x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Core,NormalInstruction,ProgramTrace,PrivilegeState |
| | | | | | 1 | 1 | 1 | Core, Normal Instruction, Privilege State |
| | | | | 1 | 0 | 1 | 0 | Core, Reservation Data, Privilege State |
| | | | | | 1 | 1 | 1 | Core, Normal Data, Privilege State |
| | | | 1 | 0 | 0 | 0 | 1 | Core,NormalInstruction,ProgramTrace,ProblemState |
| | | | | | 1 | 1 | 1 | Core, Normal Instruction, Problem State |
| | | | | 1 | 0 | 1 | 0 | Core, Reservation Data, Problem State |
| | | | | | 1 | 1 | 1 | Core, Normal Data, Problem State |
| | | 1 | CH0 | CH1 | CH2 | 1 | 1 | No Core, Normal, (CH indicates channel number) |

### Table 13-5. Address Types Definition (Continued)

| s$\overline{TS}$ | $\overline{TS}$ | AT(0) | AT(1) | AT(2) | AT(3) | PTR | RSV | ADDRESS SPACE DEFINITIONS |
|---|---|---|---|---|---|---|---|---|
| | | CPU/CPM | PROBLEM STATE/<br>PRIVILEGE STATE | INSTRUCTION/<br>DATA | RESERVATION/<br>PROGRAM<br>TRACE | PROGRAM<br>TRACE | RESERVATION | |
| x | 1 | 0 | 0 | 0 | 0 | 0 | 1 | Core, Show Cycle Address Instruction, Program Trace, Privilege State |
| | | | | | 1 | 1 | 1 | Core, Show Cycle Address Instruction, Privilege State |
| | | | | 1 | 0 | 1 | 0 | Core, Reservation Show Cycle Data, Privilege State |
| | | | | | 1 | 1 | 1 | Core, Show Cycle Data, Privilege State |
| | | | 1 | 0 | 0 | 0 | 1 | Core, Show Cycle Address Instruction, Program Trace, Problem State |
| | | | | | 1 | 1 | 1 | Core, Show Cycle Address Instruction, Problem State |
| | | | | 1 | 0 | 1 | 0 | Core, Reservation Show Cycle Data, Problem State |
| | | | | | 1 | 1 | 1 | Core, Show Cycle Data, Problem State |
| | | 1 | CH0 | CH1 | CH2 | 1 | 1 | No Core, Show Cycle Data (CH indicates channel number) |

Show cycles are accesses to the CPU's internal bus devices. These accesses are driven externally for emulation, visibility, and debugging purposes. 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 and the data for both the read and write accesses should be driven by the bus master. This is different than the normal bus read and write accesses. The address of the show cycle should be valid on the bus for one clock and the data of the show cycle should be valid on the bus for one clock. The data phase should not require a transfer acknowledge to terminate the bus-show cycle. In a burst-show cycle only the first data beat will be shown externally.

**13.5.7.3.5 Burst Data in Progress.** This signal is from the master to the slave indicating that there is a data beat following the current data beat. The master uses this signal to give the slave advanced warning of the remaining data in the burst. This can also be used to early terminate the burst cycle during a burst. Refer to **Section 13.5.2 Single Beat Transfer** and **Section 13.5.4 Burst Mechanism** for more information.

## 13.5.8 Termination Signals

**13.5.8.1 TRANSFER ACKNOWLEDGE.** Transfer acknowledge indicates normal completion of the bus transfer. During burst cycle, the slave asserts this signal with every data beat returned or accepted.

**13.5.8.2 BURST INHIBIT.** The $\overline{\text{BI}}$ signal is sent from the slave to the master to indicate that the addressed device does not have burst capability. If this signal is asserted, the master must transfer in multiple cycles and increment the address for the slave to complete the burst transfer. For a system that does not use the burst mode at all, this signal can be tied to a low permanently.

**13.5.8.3 TRANSFER ERROR ACKNOWLEDGE.** Terminates the bus cycle under bus error condition(s). The current bus cycle should be aborted. This signal should override any other cycle termination signals, such as transfer acknowledge.

**13.5.8.4 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 the termination signals should be connected to power through a pull-up resistor to avoid the situation in which a master samples an undefined value in any of these signals when no real slave is addressed. Refer to Figures 13-25 and 13-26 for more information.

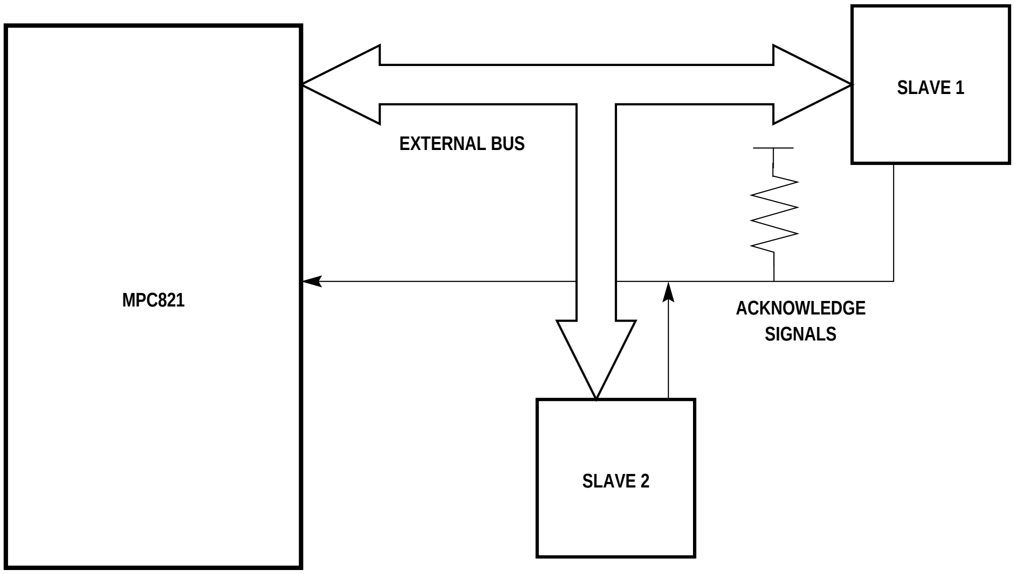


Figure 13-25. Termination Signals Protocol Basic Connection

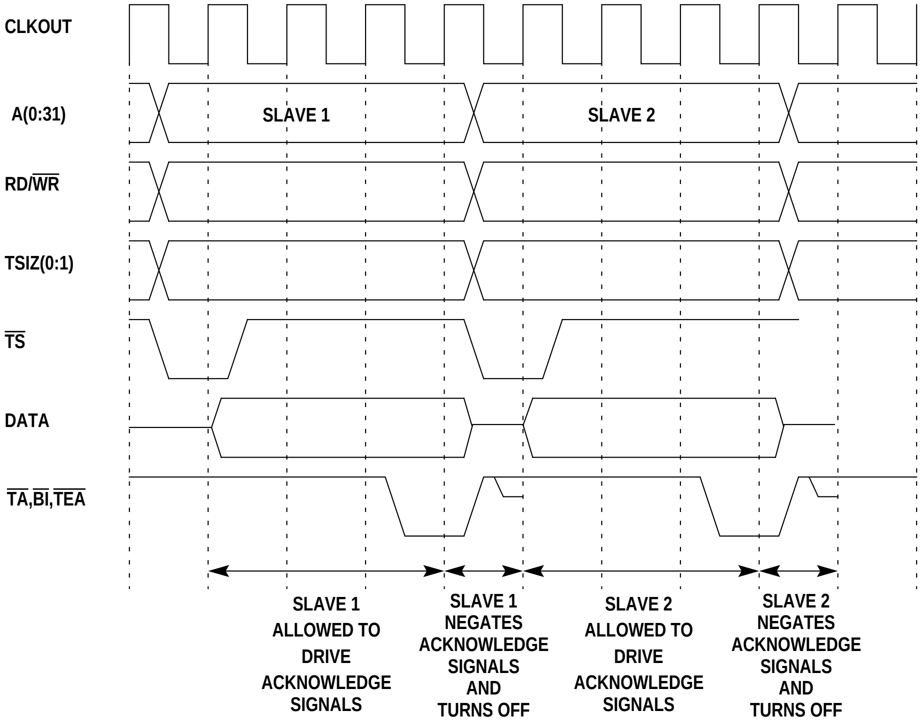

CLKOUT
A(0:31)
SLAVE 1
SLAVE 2
RD/$\overline{\text{WR}}$
TSIZ(0:1)
$\overline{\text{TS}}$
DATA
$\overline{\text{TA}}$,$\overline{\text{BI}}$,$\overline{\text{TEA}}$
SLAVE 1 ALLOWED TO DRIVE ACKNOWLEDGE SIGNALS
SLAVE 1 NEGATES ACKNOWLEDGE SIGNALS AND TURNS OFF
SLAVE 2 ALLOWED TO DRIVE ACKNOWLEDGE SIGNALS
SLAVE 2 NEGATES ACKNOWLEDGE SIGNALS AND TURNS OFF

Figure 13-26. Termination Signals Protocol Timing Diagram

### 13.5.9 Storage Reservation

The MPC821 storage reservation protocol supports multilevel bus structure. For each local bus, storage reservation is handled by the local reservation logic. The protocol tries to optimize reservation cancellation such that a PowerPC processor is notified of storage reservation loss on a remote bus only when it has issued a STWCX cycle to that address. That is, the reservation loss indication comes as part of the STWCX cycle. This method avoids the need to have very fast storage reservation loss indication signals routed from every remote bus to every PowerPC master. The storage reservation protocol presumes the following assumptions:

- Each “processor” has, at most, one reservation “flag”.
- lwarx sets the reservation “flag”.
- lwarx by the same processor clears the reservation “flag” related to a previous lwarx instruction and again sets the reservation “flag”.
- stwcx by the same processor clears the reservation “flag”.
- 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”.
- In case the storage reservation is lost, it is guaranteed that stwcx will not modify the storage.

The reservation protocol for a single-level (local) bus is illustrated in Figure 13-27. It assumes that an external logic on the bus carries out the following functions:

- Snoops accesses to all local bus slaves.
- Holds one reservation for each local master capable of storage reservations.
- Sets the reservation when that master issues a load and reserve request.
- Clears the reservation when some other master issues a store to the reservation address.

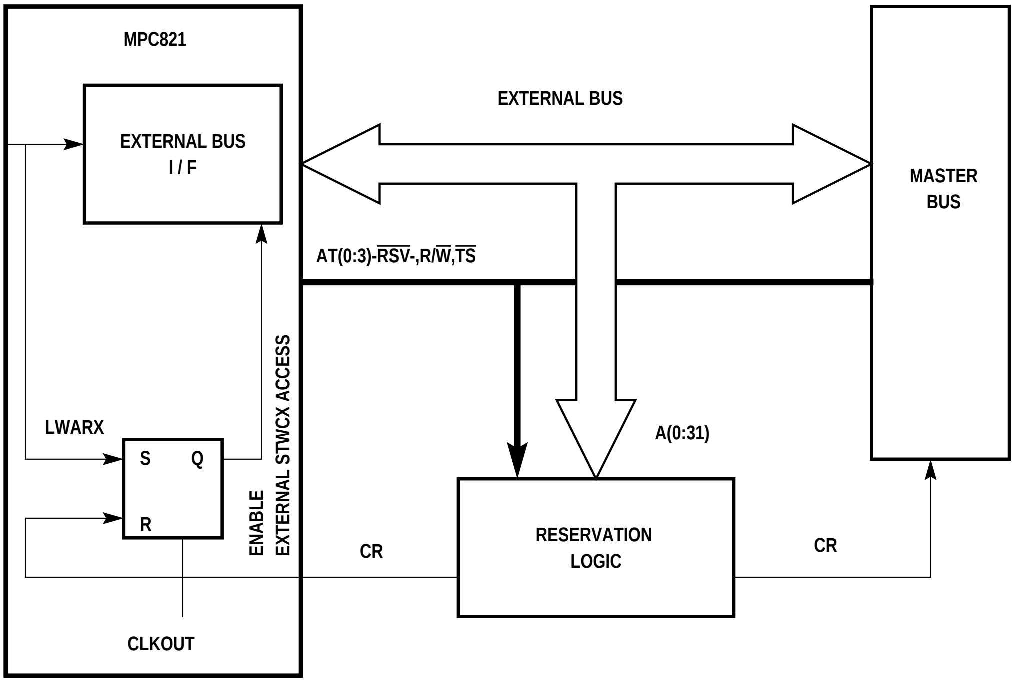


**Figure 13-27. Reservation On Local Bus**

The CR line is sampled by the MPC821 at the rising edge of the CLKOUT. When this signal is asserted, the reservation “flag” is reset. The external bus interface samples the logical value of the reservation “flag” prior to externally starting a bus cycle initiated by a STWCX instruction in the CPU core. If the reservation “flag” is set, the external bus interface begins with the bus cycle and if it is reset, no bus cycle is initiated externally and this situation is reported to the CPU core. The reservation protocol for a multi-level (local) bus is illustrated in Figure 13-27. The system describes the situation in which the reserved location is sited in the remote bus.

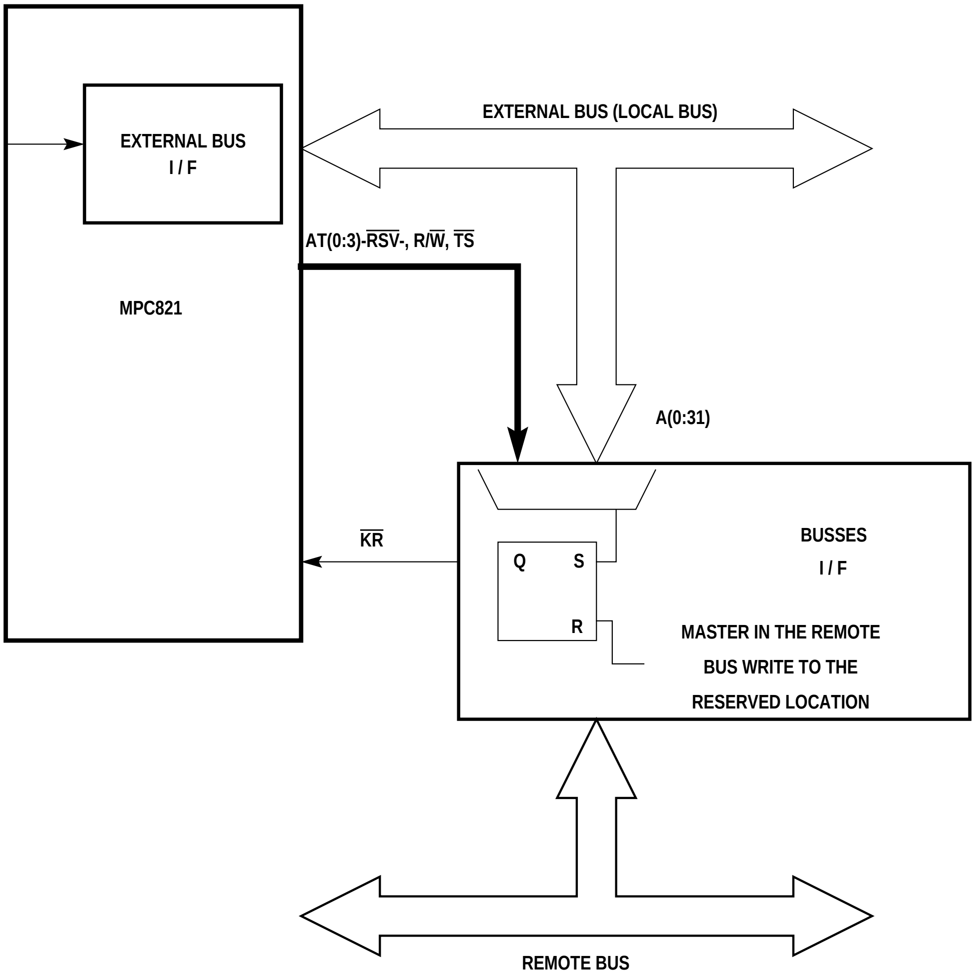


**Figure 13-28. Reservation On Multilevel Bus Hierarchy**

In this case, the busses’ I/F block implements a reservation “flag” for the local bus master. The reservation “flag” is set by the busses’ I/F when a load with reservation is issued by the local bus master and the reservation address is located on the remote bus. The “flag” is reset when an alternative master on the remote bus accesses the same location in a write cycle. If the MPC821 begins a memory cycle to the previously reserved address (located in the remote bus) as a result of a STWCX instruction, the following two cases can possibly occur:

- If the reservation “flag” is set, the busses I/F acknowledges the cycle in a normal way and if it is reset, the busses’ I/F should assert the $\overline{\text{KR}}$. However, the busses’ I/F should not perform the remote bus write access or abort it if the remote bus supports aborted cycles. In this case the failure of the STWCX instruction is reported to the CPU core.

### 13.5.10 Bus Exception Control Cycles

The MPC821 bus architecture requires assertion of the $\overline{\text{TA}}$ from an external device to signal that the bus cycle is complete. $\overline{\text{TA}}$ is not asserted in the following cases:

- The external device does not respond
- Various other application-dependent errors occur

External circuitry can provide $\overline{\text{TEA}}$ when no device responds by asserting $\overline{\text{TA}}$ within an appropriate period of time after the MPC821 initiates the bus cycle (it can be the internal MPC821- bus monitor). 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. $\overline{\text{TEA}}$ should be negated before the second rising edge after it was sample-asserted to avoid the detection of an error for the next initiated bus cycle. $\overline{\text{TEA}}$ is an open-drain pin that allows the “wire or” of any different sources of error generation.

**13.5.10.1 RETRY.** When an external device asserts the $\overline{\text{RETRY}}$ signal during a bus cycle, the MPC821 enters a sequence in which it terminates the current transaction, relinquishes the ownership of the bus, and retries the cycle using the same address, address attributes, and data (in the case of a write cycle). Figure 13-29 illustrates the behavior of the MPC821 when the $\overline{\text{RETRY}}$ signal is detected as a termination of a transfer. In the figure, it is shown that in the case when the internal arbiter is enabled, MPC821 negates the $\overline{\text{BB}}$ and asserts the $\overline{\text{BG}}$ in the clock cycle following the retry detection. This allows any external master to gain the bus ownership. In the next clock cycle, a normal arbitration procedure occurs again. The figure also shows that the external master did not use the bus, so the MPC821 initiates a new transfer with the same address and attributes as before. In Figure 13-30 the same situation is shown where the MPC821 is working with an external arbiter. In this case, in the clock cycle after the $\overline{\text{RETRY}}$ signal is detected asserted, the $\overline{\text{BR}}$ is negated together with the $\overline{\text{BB}}$. One clock cycle later, the normal arbitration procedure occurs again.

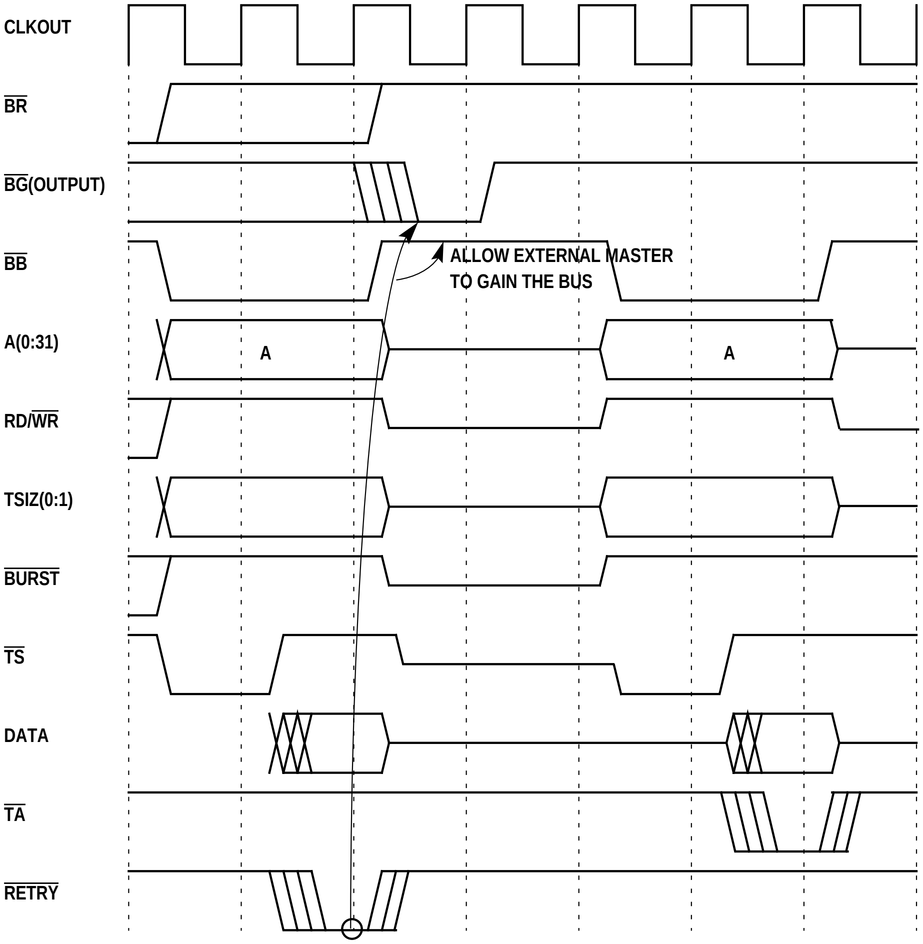


Figure 13-29. Retry Transfer Timing–Internal Arbiter

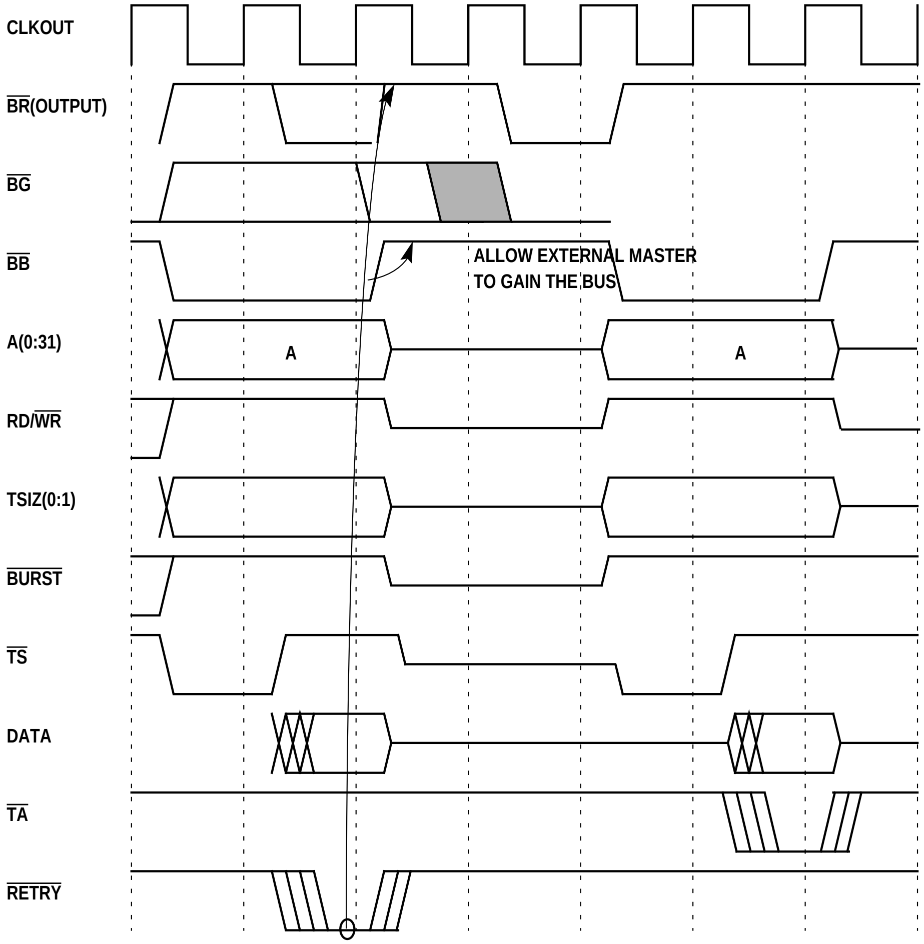


**Figure 13-30. Retry Transfer Timing–External Arbiter**

When a burst access is initiated by the MPC821, the bus interface only recognizes the $\overline{\text{RETRY}}$ assertion as a retry termination if it detects it before the first data beat was acknowledged by the slave device. When the $\overline{\text{RETRY}}$ signal is asserted as a termination signal on the second or third data beat of the access (being the first data beat acknowledged by a normal $\overline{\text{TA}}$ assertion), it is recognized by the MPC821 as a transfer error acknowledge.

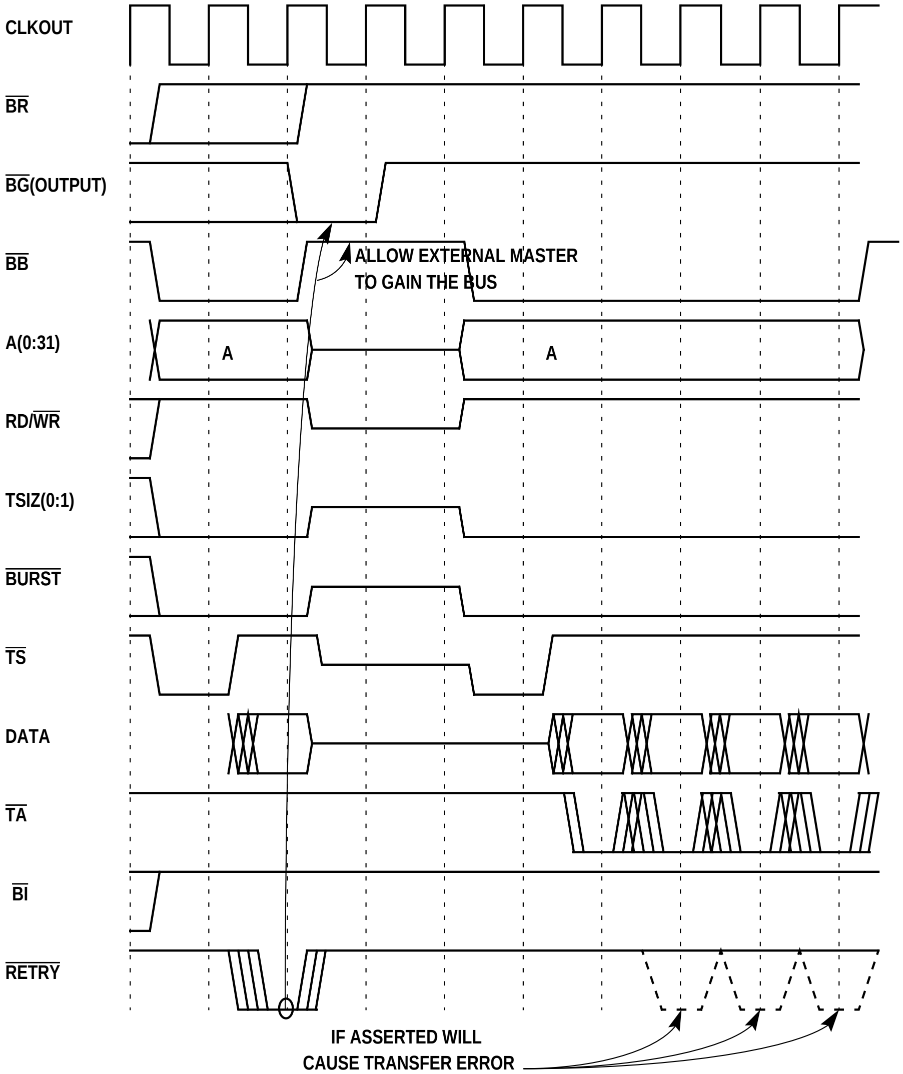


Figure 13-31. Retry On Burst Cycle

If a burst access is acknowledged on its first beat with a normal TA, but with the BI signal asserted, the following “single beat” transfers initiated by the MPC821 to complete the 16 byte transfers recognizes the RETRY signal assertion as a transfer error acknowledge.

Table 13-6 summarizes how the MPC821 recognizes the termination signals provided by the slave device that is addressed by the initiated transfer.

**Table 13-6. Termination Signals Protocol**

| TEA | TA | RETRY/KR | ACTION |
|---|---|---|---|
| Asserted | X | X | Transfer Error Termination |
| Negated | Asserted | X | Normal Transfer Termination |
| Negated | Negated | Asserted | Retry Transfer Termination / Kill Reservation |

# SECTION 14
# ENDIAN MODES

## 14.1 OVERVIEW

A general description on the different endian modes can be found in *The PowerPC™ Microprocessor Family: The Programming Environments* (MPCFPE/AD) that is available from Motorola. The MPC821 supports three different system endian configurations:

- Little-endian system
- Big-endian system
- PowerPC™ little-endian system

The term system refers to the devices that reside on the MPC821 bus. The MPC821 core operates in the big-endian mode of a big-endian system and in the PowerPC little-endian mode of two other configurations.

**Table 14-1. PowerPC Little-Endian Effective Address Modification For Individual Aligned Scalar**

| DATA LENGTH (BYTES) | ADDRESS MODIFICATION: |
|---|---|
| 1 | XOR with 0b111 |
| 2 | XOR with 0b110 |
| 4 | XOR with 0b100 |
| 8 | (No Change) |

NOTE: There are no 8-byte scalars in the MPC821.

For programming of the configurations, refer to the table below.

**Table 14-2. Endian Mode Programming For Core Data Structures**

| MODE | $\mathrm{MSR_{LE}}$ (AND $\mathrm{MSR_{ILE}}$) | $\mathrm{DCCST_{LES}}$ |
|---|---|---|
| Big-Endian Mode | 0 | 0 |
| Little-Endian Mode | 0 | 1 |
| PowerPC Little-Endian Mode | 1 | 0 |
| Reserved | 1 | 1 |

The hardware operations that are used to support the different endian modes are:

- Address munging (refer to *The PowerPC™ Microprocessor Family: The Programming Environments*) in the core that is controlled by the $MSR_{LE}$ bit.
- MPC821 internal bus signal driven by the master that informs the SIU to swap and perform address demunging or leave the current access as it is. This is defined by the $DCCST_{LES}$ bit for core and cache accesses in Table 10-1.
- Address munging and data bytes format in the CPM that is controlled by the BO field of the FCR (refer to the example in **Section 16.14.7.2 SCC Function Code Registers**).

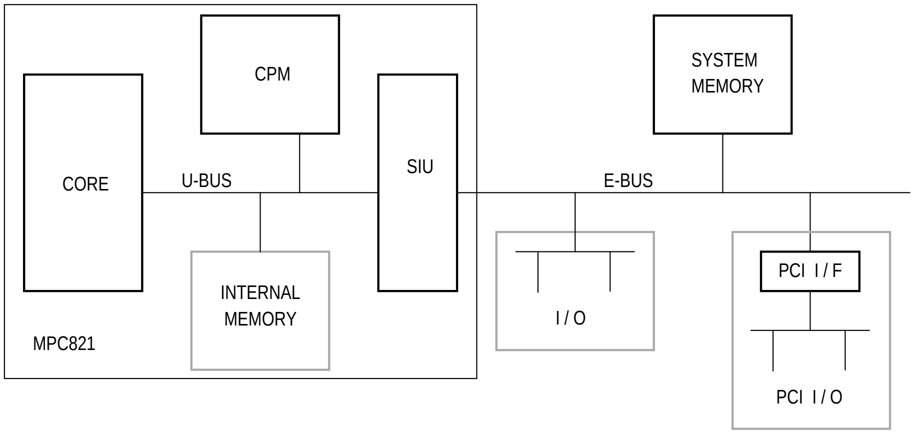


**Figure 14-1. General MPC821 System Diagram**

**NOTE**

The PCI bridge cannot be used in the little-endian system.

## 14.2 LITTLE ENDIAN FEATURES

The following is a list of the little-endian system’s important features.

- System memory organization and E-bus format is little-endian
- U-bus data, I-cache, D-cache, and internal memory format is big-endian
- Data access constraints, according to the PowerPC little-endian rules (no unaligned, multiple, or string accesses)
- Same byte order between the media and system memory (no swap when the I/O master writes/reads memory)
- For core accesses, swap and address demunging are performed by the SIU on the U-bus ↔ system path
- The core load/store unit swapper uses munged addresses to put the data on the right byte lanes when access of half-word or byte is performed
- CPM performs data swapping according to the information in the buffer descriptors. Generally, data should be referred to as big-endian when accessing internal memory and little-endian when accessing the external system memory. Because the buffer descriptors reside in the internal memory, they should be organized in big-endian format.

The following tables describe the handling of the little-endian program/data in the little-endian system built around the MPC821 for various port sizes:

**Table 14-3. Little-Endian Program/Data Path Between Register and 32-Bit Memory**

| FETCH/ LOAD STORE TYPE | LITTLE-ENDIAN ADDR | U-BUS AND CACHES ADDR | EXTERNAL BUS ADDR | DATA IN THE REGISTER | | | | U-BUS AND CACHES FORMAT | | | | E-BUS FORMAT | | | | LITTLE-ENDIAN PROGRAM/DATA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **3** | **3** | **2** | **1** | **0** |
| Word | 0 | 0 | 0 | 11 | 12 | 13 | 14 | 11 | 12 | 13 | 14 | 14 | 13 | 12 | 11 | 11 | 12 | 13 | 14 |
| Half-word | 0 | 2 | 0 | | | 21 | 22 | | | 21 | 22 | 22 | 21 | | | | | 21 | 22 |
| Half-word | 2 | 0 | 2 | | | 31 | 32 | 31 | 32 | | | | | 32 | 31 | 31 | 32 | | |
| Byte | 0 | 3 | 0 | | | | ‘a’ | | | | ‘a’ | ‘a’ | | | | | | | ‘a’ |
| Byte | 1 | 2 | 1 | | | | ‘b’ | | | ‘b’ | | | ‘b’ | | | | | ‘b’ | |
| Byte | 2 | 1 | 2 | | | | ‘c’ | | ‘c’ | | | | | ‘c’ | | | ‘c’ | | |
| Byte | 3 | 0 | 3 | | | | ‘d’ | ‘d’ | | | | | | | ‘d’ | ‘d’ | | | |

## Table 14-4. Little-Endian Program/Data Path Between Register and 16-Bit Memory

| FETCH/ LOAD STORE TYPE | LITTLE-ENDIAN ADDR | U-BUS AND CACHES ADDR | EXTERNAL BUS ADDR | DATA IN THE REGISTER | | | | U-BUS AND CACHES FORMAT | | | | E-BUS FORMAT | | | | LITTLE-ENDIAN PROGRAM/DATA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **3** | **3** | **2** | **1** | **0** |
| Word | 0 | 0 | 0 | 11 | 12 | 13 | 13 | 11 | 12 | 13 | 14 | 14 | 13 | | | | | 13 | 14 |
| | | | 2 | | | | | | | | | 12 | 11 | | | | | 11 | 12 |
| Half-word | 0 | 2 | 0 | | | 21 | 22 | | | 21 | 22 | 22 | 21 | | | | | 21 | 22 |
| Half-word | 2 | 0 | 2 | | | 31 | 32 | 31 | 32 | | | 32 | 31 | | | | | 31 | 32 |
| Byte | 0 | 3 | 0 | | | | ‘a’ | | | | ‘a’ | ‘a’ | | | | | | | ‘a’ |
| Byte | 1 | 2 | 1 | | | | ‘b’ | | | ‘b’ | | | ‘b’ | | | | | ‘b’ | |
| Byte | 2 | 1 | 2 | | | | ‘c’ | | ‘c’ | | | ‘c’ | | | | | | | ‘c’ |
| Byte | 3 | 0 | 3 | | | | ‘d’ | ‘d’ | | | | | ‘d’ | | | | | ‘d’ | |

## Table 14-5. Little-Endian Program/Data Path Between Register and 8-Bit Memory

| FETCH/ LOAD STORE TYPE | LITTLE-ENDIAN ADDR | U-BUS AND CACHES ADDR | EXTERNAL BUS ADDR | DATA IN THE REGISTER | | | | U-BUS AND CACHES FORMAT | | | | E-BUS FORMAT | | | | LITTLE-ENDIAN PROGRAM/DATA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **3** | **3** | **2** | **1** | **0** |
| Word | 0 | 0 | 0 | 11 | 12 | 13 | 14 | 11 | 12 | 13 | 14 | 14 | | | | | | | 14 |
| | | | 1 | | | | | | | | | 13 | | | | | | | 13 |
| | | | 2 | | | | | | | | | 12 | | | | | | | 12 |
| | | | 3 | | | | | | | | | 11 | | | | | | | 11 |
| Half-word | 0 | 2 | 0 | | | 21 | 22 | | | 21 | 22 | 22 | | | | | | | 22 |
| | | | 1 | | | | | | | | | 21 | | | | | | | 21 |
| Half-word | 2 | 0 | 2 | | | 31 | 32 | 31 | 32 | | | 32 | | | | | | | 32 |
| | | | 3 | | | | | | | | | 31 | | | | | | | 31 |
| Byte | 0 | 3 | 0 | | | | ‘a’ | | | | ‘a’ | ‘a’ | | | | | | | ‘a’ |
| Byte | 1 | 2 | 1 | | | | ‘b’ | | | ‘b’ | | ‘b’ | | | | | | | ‘b’ |
| Byte | 2 | 1 | 2 | | | | ‘c’ | | ‘c’ | | | ‘c’ | | | | | | | ‘c’ |
| Byte | 3 | 0 | 3 | | | | ‘d’ | ‘d’ | | | | ‘d’ | | | | | | | ‘d’ |

## 14.3 BIG-ENDIAN SYSTEM FEATURES

The following is a list of the big-endian system's important features:

- Caches, U-bus, E-bus, system memory, and I/O organization format is big-endian
- Same byte order between the media and system memory (no swap when the I/O master writes/reads memory)
- CPM writes/reads big-endian U-bus data
- The PCI bridge operates in big-endian mode as needed

## 14.4 POWERPC LITTLE-ENDIAN SYSTEM FEATURES

The following is a list of the PowerPC little-endian system's important features:

- Caches, U-bus, E-bus, system memory, and E-bus attached I/O organization format is big-endian
- PCI bus format is little-endian
- Data access constraints, according to the PowerPC little-endian rules (no unaligned, multiple or string accesses)
- Address munging in the core and CPM, according to Table 19-1
- The PCI bridge operates in the little-endian mode as needed. In this case, swap and address demunging is performed by the PCI bridge on the PCI I/O ↔ system memory path
- The stream hit mechanism of the I-cache and D-cache operates less efficiently when address munging is performed on the caches accesses. Therefore, some performance degradation is expected when working in this mode.

## 14.5 SETTING ENDIAN MODE OF OPERATION

Dynamic switch between the endian modes is not effectively supported. It is expected that the mode is set early in the reset routine and thereafter remains unchanged. The MPC821 core is in big-endian mode and the CPM is in the disable state after reset. To switch between the endian modes of operation, the core should run in the serialized mode and the caches should be disabled. To transfer the system to the PowerPC little-endian mode, the $MSR_{LE}$ and $MSR_{ILE}$ bits should be changed by using the mtmsr instruction resides (preferably) in physical address ended by 3'b100. Then the next executed instruction is fetched from this address plus 8. If the instruction resides in address ended by 3'b000, then this instruction is executed twice because of address munging.

The instruction to transfer the system back to the big-endian resides (preferably) in address ended by 3'b000. Then the next instruction is fetched from this address plus 12. Transferring to the little-endian mode (setting of bit $DCCST_{LES}$ in the D-cache) should be performed by the mtspr instruction that resides at address ended by 3'b000. Further instructions should reside in the little-endian format in the external system memory and in the big-endian format of the internal memory (if it exists). The BO field of the FCRs in the CPM should be set to the required endian format for the described buffer. Refer to the example in **Section 16.14.7.2 SCC Function Code Registers** for more information.

# SECTION 15
# MEMORY CONTROLLER

## 15.1 INTRODUCTION

The memory controller is responsible for the control of up to eight memory banks. 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 memory controller supports external address multiplexing, periodic timers, and timing generation for row address and column address strobes to allow for a glueless interface to DRAM devices. The periodic timers allow refresh cycles to be initiated while the address muxing provides row and column addresses.

The user is allowed to define different timing patterns for the control signals that govern a memory device. These patterns define how the external control signals behave in a read-access request, write-access request, burst read-access request, or burst write-access request. The user defines how the external control signals toggle when the periodic timers reach the maximum programmed value for refresh operation.

## 15.2 FEATURES

The following is a list of the memory controller’s important features:

- All eight memory banks support the following:
  - 32-bit address decode with mask
  - Various block sizes (32 kbytes to 4 Gbytes)
  - Byte parity generation/checking
  - Write-protection capability
  - “Address types” matching qualifying memory bank accesses for internal masters
  - Timing pattern machine selection according to the type of memory device accessed
  - Support for external masters access to memory banks
  - Synchronous and asynchronous external masters support
- General-purpose chip-select machine
  - Compatible with SRAM, EPROM, FEPROM, and peripherals
  - Global (boot) chip-select available at system reset
  - Boot chip-select support for 8-/16-/32-bit devices
  - Two clock accesses to external device
  - Four byte write enable ($\overline{\text{WE}}$) signals
  - Output enable ($\overline{\text{OE}}$) signal

- Two user-programmable machines
  - RAM-based machine controls the timing of the external signals with a granularity of one quarter of a system clock period
  - User-specified patterns run when a single read access, single write access, burst read access or burst write access is requested by an internal or external synchronous master
  - User specified patterns run when a single read access or single write access is requested by an external asynchronous master
  - UPM periodic timer initiates an automatic pattern when expired (refresh)
  - User specified pattern runs under software control
  - Each UPM can be defined to support DRAM devices with depths of 64K,128K, 256K, 512K, 1M, 2M, 4M, 8M, 16M, 32M, 64M, 128M, and 256M
  - Four byte select lines for each UPM
  - Six external general-purpose lines controlled by each UPM
  - Supports DRAM port size of 32 bits,16 bits, and 8 bits
  - Glueless interface to one bank of DRAM (only external buffers are required for additional SIMM banks)
  - Page mode support for successive transfers within a burst for all on-chip and external synchronous masters
  - Internal address multiplexing for all on-chip bus masters supporting 64K, 128K, 256K, 512K, 1M, 2M, 4M, 8M, 16M, 32M, 64M, 128M, 256M page banks
  - Glueless interface to EDO DRAM devices
  - Glueless interface to self refresh devices
  - Glueless interface to synchronous DRAM devices

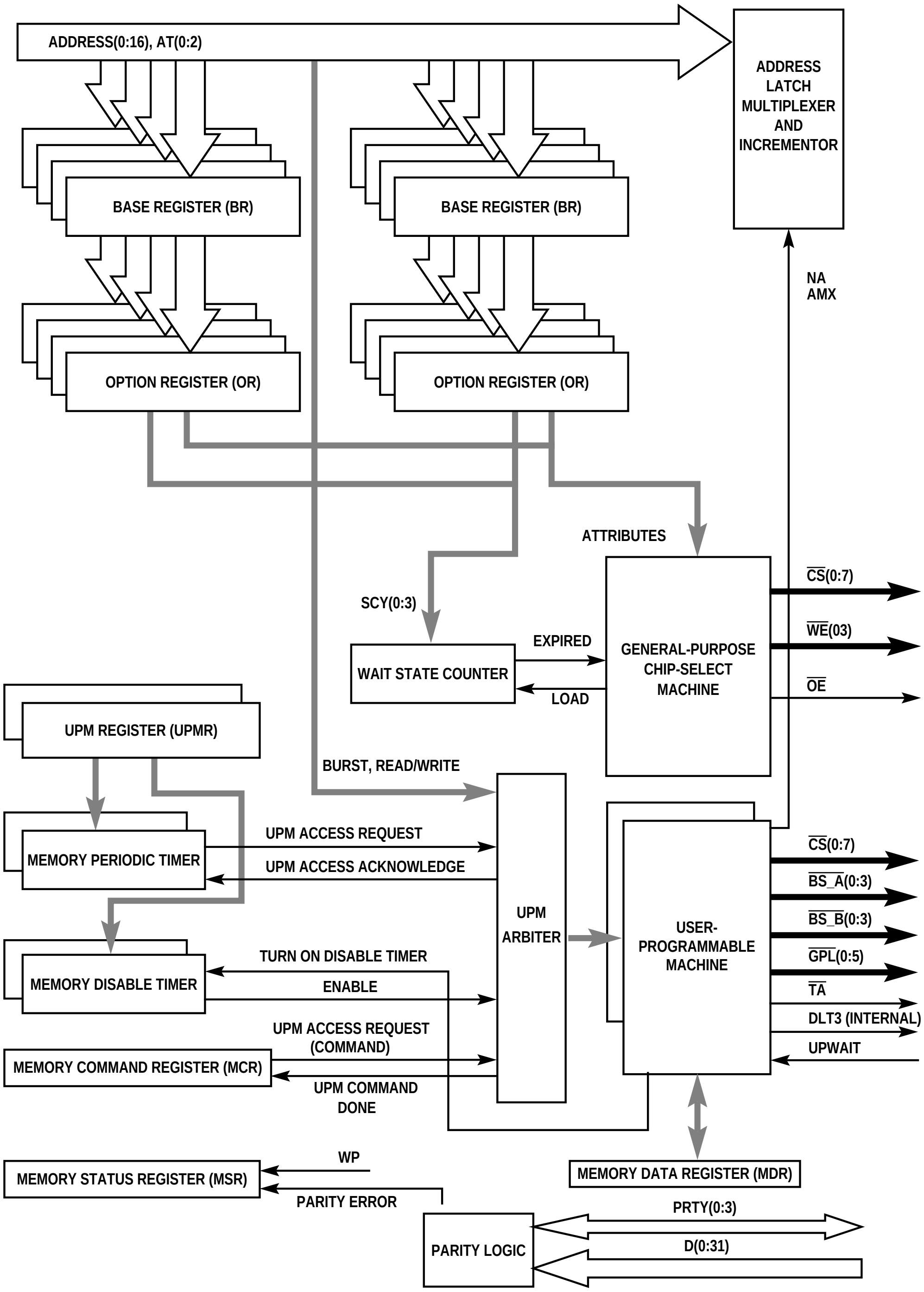


Figure 15-1. Memory Controller Block Diagram

## 15.3 MEMORY CONTROLLER ARCHITECTURE

### 15.3.1 General Overview

The memory controller consists of three basic machines as shown in Figure 15-1.

- General-purpose chip-select machine
- User-programmable machine A
- User-programmable machine B

Each bank can be assigned to any one of these machines by means of the machine-select bits (MS(0:1)) in the base register. Refer to Figure 15-3 for more information. When an access to one of the memory banks is initiated, the corresponding machine takes “ownership” of the external signals controlling the access until the cycle is terminated.

The general-purpose chip-selects machine (GPCM) provides a glueless interface to EPROM, SRAM, Flash EPROM (FEPROM), and other peripherals. The general-purpose chip-selects are available on lines $\overline{\text{CS0}}$ through $\overline{\text{CS7}}$. $\overline{\text{CS0}}$ also functions as the global (boot) chip-select for accessing the boot EPROM. The chip-select allows 0 to 30 wait states.

Some features are common to all eight memory banks. A full 32-bit address decode on each memory bank is made possible with 17 bits having address masking. The full 32-bit decode is available, even if all 32 address bits are not visible outside the MPC821. Each memory bank includes a variable block size of 32 kbytes and 64 kbytes on up to 256 Mbytes. Parity may be generated and checked for any memory bank and each memory bank can be selected for read-only or read/write operation. Finally, the access to a memory bank, may be restricted to certain address type codes for system protection. The address type comparison provides a mask option for additional flexibility.

The memory controller functionality allows MPC821-based systems to be designed with little or no glue logic required. In Figure 15-2, $\overline{\text{CS0}}$ is used as the 16-bit boot EPROM and $\overline{\text{CS1}}$ is used for the 32-bit DRAM as the RAS signal. The $\overline{\text{BS\_A}}$(0:3) signals are used as the CAS signals on the DRAM.

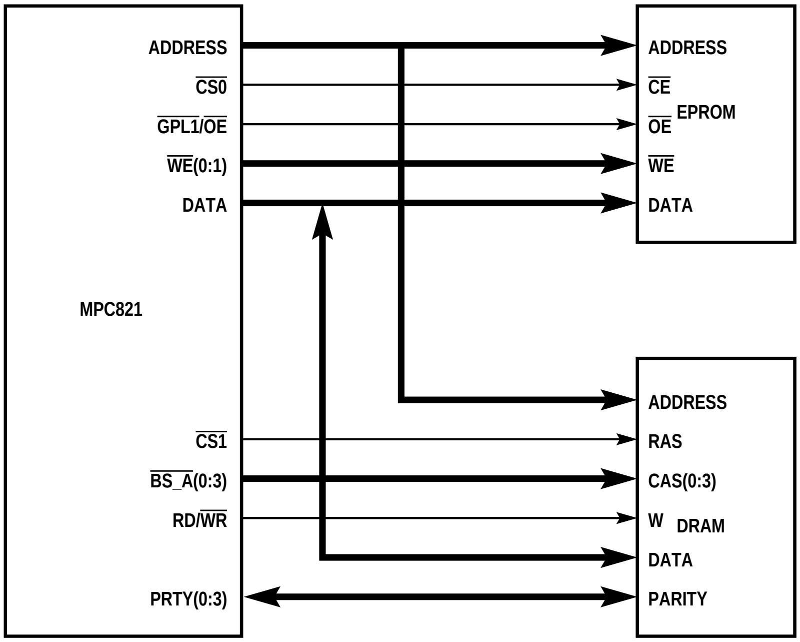


**Figure 15-2. MPC821 Simple System Configuration**

The two user-programmable machines in the memory controller (UPMA and UPMB) provide a very flexible interface to many types of memory devices. At the same time, each UPM can control the address multiplexing necessary to access DRAM devices, the timing of the BS(0:3) lines, and the timing of the general-purpose lines (GPL(0:5)). Each memory bank can be assigned to any UPM, thus each one controls eight CS lines (CS(0:7)).

Each user-programmable machine is a RAM-based machine that is under software control. The software toggles the memory controller external signals when an external single word read/write access or an external burst read/write access is initiated by an internal (or external) master. The UPM also controls address multiplexing, address increment, and the transfer acknowledge assertion for a specific memory access. The UPM can be programmed to run a specific pattern consisting of a specific number of clock cycles. At every clock cycle, the logical value of the external signals specified in the RAM is output on the corresponding pins.

When a new access to external memory is requested by any of the internal (or external) masters, the address of the transfer and the address type is compared to each one of the valid banks defined in the memory controller. Notice that 17 of the address bits and three of the address type bits are maskable.

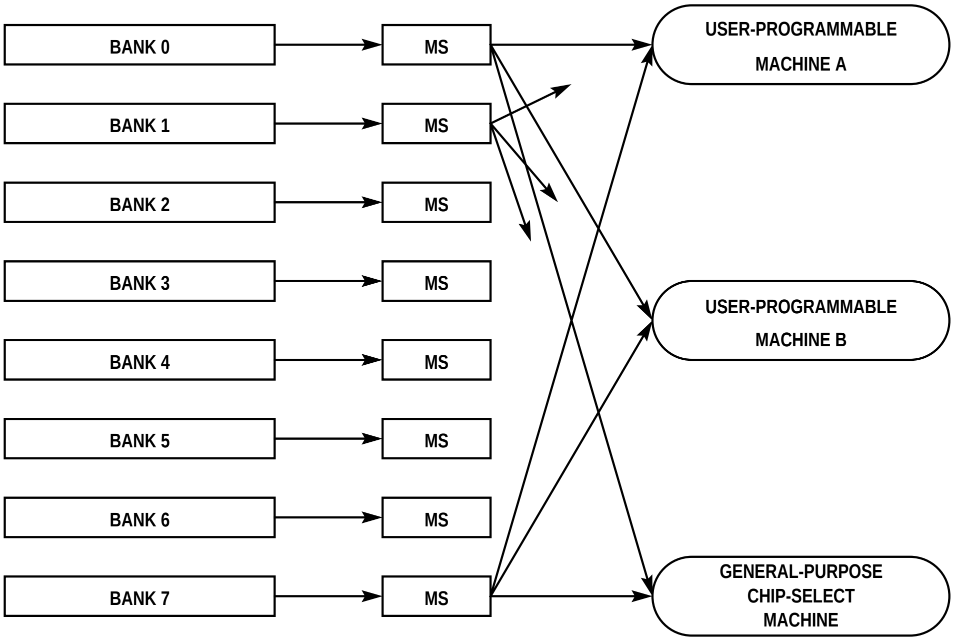

BANK 0
BANK 1
BANK 2
BANK 3
BANK 4
BANK 5
BANK 6
BANK 7
MS
MS
MS
MS
MS
MS
MS
MS
USER-PROGRAMMABLE MACHINE A
USER-PROGRAMMABLE MACHINE B
GENERAL-PURPOSE CHIP-SELECT MACHINE

**Figure 15-3. Memory Controller Machine Selection**

When an address match is found in one of the memory bank's chip select range, the base register MS bits define which machine handles the memory access. Refer to Figure 15-4 for details. The memory controller provides four parity lines (PRTY(0:3), one for each data byte on the MPC821 system bus.The parity on the bus is only checked if the memory bank accessed in the current transaction has parity enabled. Figure 15-1 illustrates the block diagram for the MPC821 memory controller.

The user can enable parity checking/generation for a specific memory bank in the base register. The type of parity is defined in the SIUMCR. Also, system protection is provided by defining each memory bank as read-only or read/write.

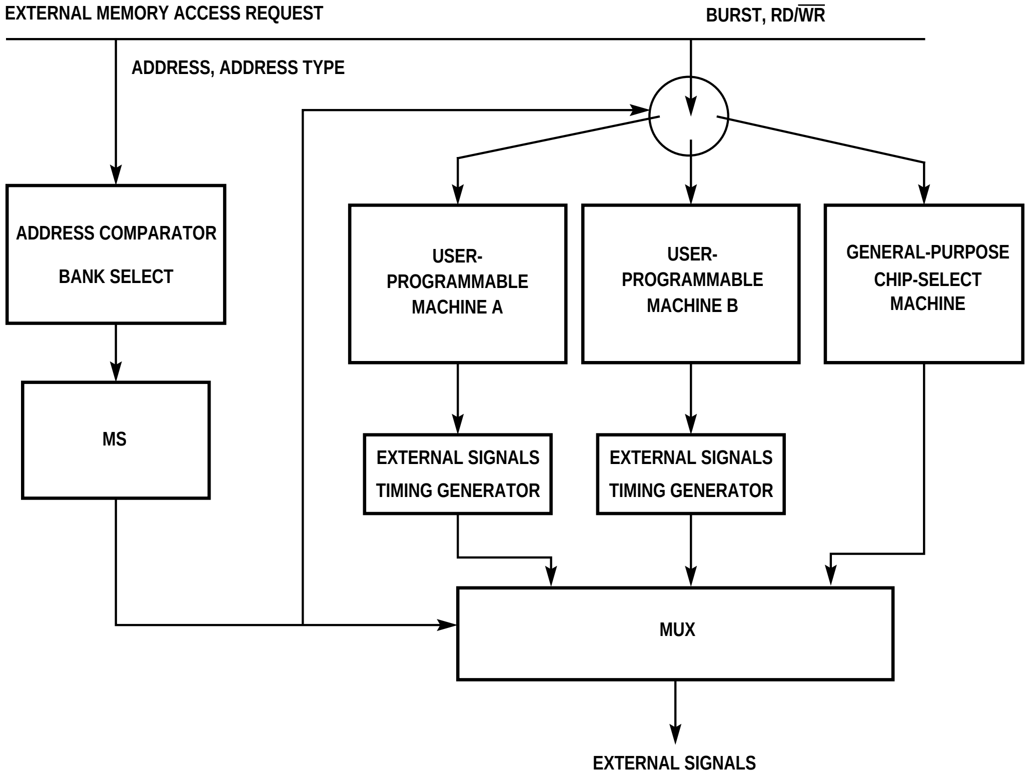

EXTERNAL MEMORY ACCESS REQUEST
BURST, RD/$\overline{WR}$
ADDRESS, ADDRESS TYPE
ADDRESS COMPARATOR
BANK SELECT
USER-PROGRAMMABLE MACHINE A
USER-PROGRAMMABLE MACHINE B
GENERAL-PURPOSE CHIP-SELECT MACHINE
MS
EXTERNAL SIGNALS TIMING GENERATOR
EXTERNAL SIGNALS TIMING GENERATOR
MUX
EXTERNAL SIGNALS

**Figure 15-4. Memory Controller Basic Operation**

**15.3.1.1 ASSOCIATED REGISTERS.** Status bits for each one of the memory banks are in the memory controller status register (MSTAT) and there is only one MSTAT for the entire memory controller. Each memory bank has a base register (BR) and a option register (OR). The MSTAT reports write-protect violations that have occurred and parity errors for every bank. The BRx and the ORx registers are specific memory to bank *x*. The BR contains a valid (V) bit that indicates that the register information for that chip-select is valid.

Each one of the OR registers define the attributes for the general-purpose-chip select machine when accessing the corresponding bank. The OR registers also define the initial address multiplexing for a memory cycle controlled by a UPM. The machine A mode register (MAMR) and machine B mode register (MBMR) define most of the global features for each one of the user-programmable machines.

The memory command register (MCR), with the memory data register (MDR), are the method for initializing the UPMs RAMs and specifying which pattern should be run when required by the software. The memory address register (MAR) allows a specific pattern to output the data stored in this register to the address pins.

**15.3.1.2 8-, 16-, AND 32-BIT PORT SIZE CONFIGURATION.** The memory controller supports multiple port sizes. Predefined 8-bit ports can be accessed as odd or even bytes, predefined 16-bit ports can be accessed as odd bytes, even bytes, or even half-words on data bus bits 0 through 15, predefined 32-bit ports can be accessed as odd bytes, even bytes, odd half-words, even half-words or words on word boundaries. The port size is specified by the PS bits in the BR.

**15.3.1.3 WRITE-PROTECT CONFIGURATION.** The WP bit in each base register restricts write access to a certain address range. Any attempt to write to this area results in the WPER bit being set in the MSTAT.

**15.3.1.4 ADDRESS AND ADDRESS SPACE CHECKING.** The defined base address is written to the BR. The address mask bits for that address are written to the OR. The address type access value, if preferred, is written to the AT bits in the BR. The ATM bits in the OR may be used to mask this selection. If the address type checking is not preferred, program the ATM bits to zero. Each time an external bus cycle access is requested, the address, and its corresponding address type is compared with each one of the banks. If a match is found on one of the memory controller banks, the attributes defined for that bank in it's BR and OR are used to control the memory access. If a match is found in more than one bank the lowest number bank matched handles the memory access (Bank 0 has priority over Bank 1). It should be noted that when external masters access slaves on the bus, the internal AT(0:2) lines to the memory controller are forced to '100'.

**15.3.1.5 PARITY GENERATION AND CHECKING.** Parity can be configured for any bank, if it is preferred. Parity is generated and checked on a per-byte basis using PRTY(0:3), for the bank if the PARE bit is set in the BR. The OPAR bit determines the type of parity (odd or even). Any parity error results in the assertion of the associated PERx bit in the MSTAT register and interrupt generation. Refer to **Section 12.4.1.5 Transfer Error Status Register** for details.

**15.3.1.6 TRANSFER ERROR ACKNOWLEDGE GENERATION.** An internal transfer error indication signal is asserted by the memory controller in the case of a parity error (when enabled) or by the bus monitor of the SIU as a result of a write protect violation.

## 15.3.2 General-Purpose Chip-Select Machine

The GPCM allows a glueless and flexible interface between the MPC821 and SRAM, EPROM, FEPROM, ROM devices, and external peripherals. If the MS bits in the BRx of the selected bank (Bank *x*) select the GPCM machine, the attributes for the memory cycle initiated are taken from the ORx register. These attributes include the CSNT, ACS(0:1), SCY(0:3), TRLX, EHTR, and SETA fields.

Anywhere from 0 to 30 wait states can be programmed for $\overline{\text{TA}}$ generation. Byte write enable signals ($\overline{\text{WE}}$(0:3)) are available for each byte that is written to memory. Also, an output enable ($\overline{\text{OE}}$) signal is provided to eliminate external glue logic. The memory banks selected to work with the GPCM machine have features unique to the GPCM. On system reset, a global (boot) chip-select is available that provides a boot ROM chip-select prior to the system being fully configured.

Next, the banks selected to work with the GPCM support an option to output the $\overline{\text{CS}}$ line at different timings with respect to the external address bus. $\overline{\text{CS}}$ can be output in any of three configurations:

- Simultaneous with the external address
- One quarter of a clock later
- One half of a clock later

This depends on the value of the ACS field, plus an additional cycle if the TRLX bit is set. The GPCM allows connection to devices that have long disconnect times on data by delaying new bus transactions addressing other memory banks for additional clock cycles. Finally, the banks selected to work with the GPCM support termination of an external cycle by sensing the $\overline{\text{TA}}$ signal asserted by the addressed external slave.

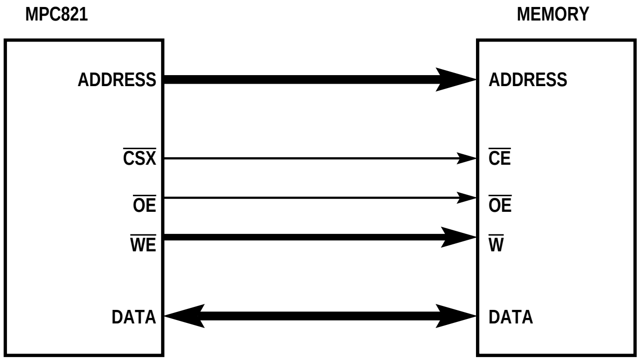


**Figure 15-5. MPC821 GPCM–Memory Devices Interface**

Figure 15-5 describes the basic connection between the MPC821 and a “static” memory device. In this case $\overline{\text{CSx}}$ is connected directly to the chip enable ($\overline{\text{CE}}$) of the memory device. The $\overline{\text{WE}}$(0:3) lines are connected to the respective $\overline{\text{W}}$ in the memory device where each $\overline{\text{WE}}$ line corresponds to a different data byte. As illustrated in Figure 15-6, the $\overline{\text{CSx}}$ timing is the same as the address lines output. The strobes for the transaction are supplied by the $\overline{\text{OE}}$ or $\overline{\text{WE}}$ lines, depending on the transaction direction (read or write). This $\overline{\text{CS}}$ timing is generated when the ACS bits in the corresponding ORx register are set to ‘00’.

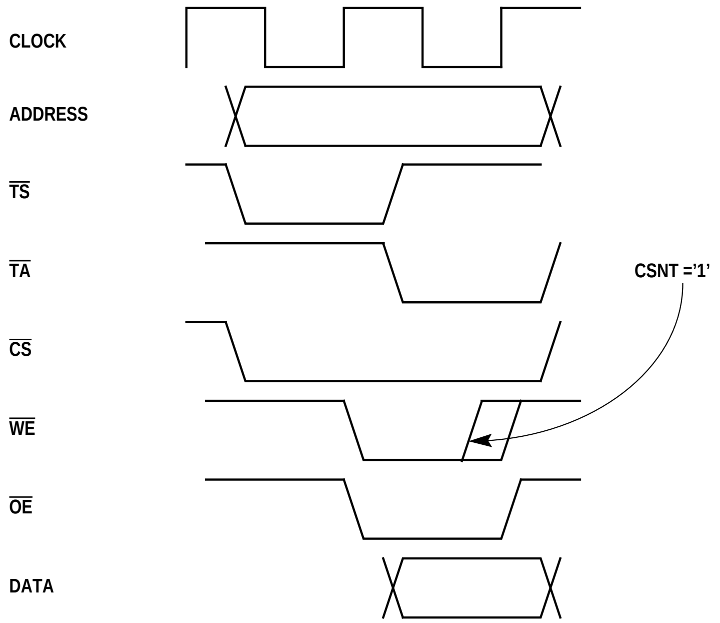


**Figure 15-6. MPC821 GPCM–Memory Devices Basic Timing (ACS = 00,TRLX = 0)**

Figure 15-7 illustrates the basic connection between the MPC821 and an external peripheral device. In this case CSx is connected directly to the chip enable (CE) of the memory device and the R/W line is connected to the respective R/W in the peripheral device. In this case the CSx line is the strobe output for the memory access.

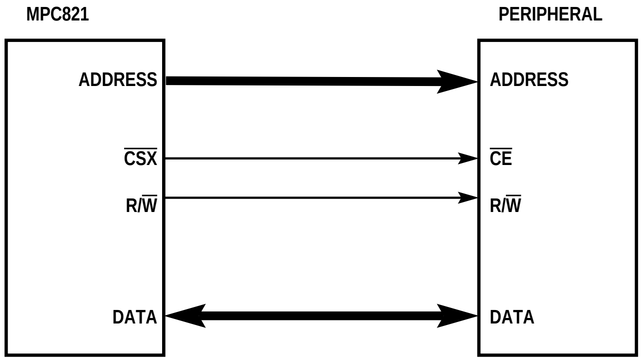


**Figure 15-7. MPC821 GPCM–Peripheral Devices Interface**

Figure 15-8 illustrates the CSx timing as defined by the setup time required between the address lines and the CE line. The MPC821 memory controller allows the user to specify the CS timing to meet this requirement through the ACS field in the option register.

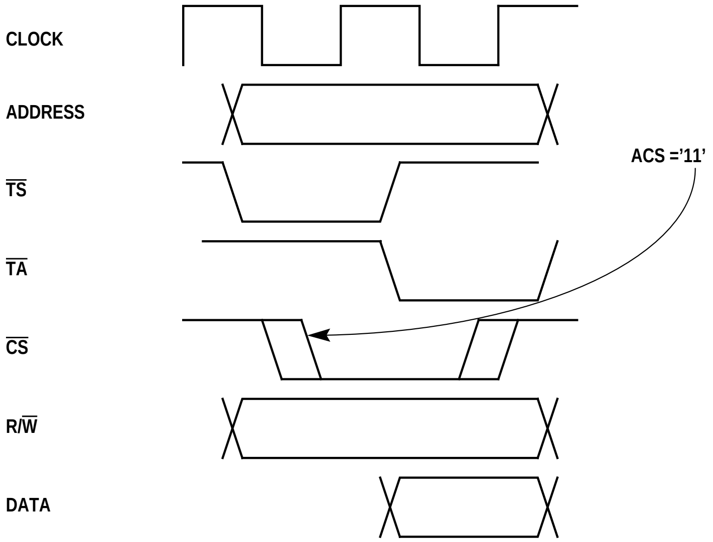


**Figure 15-8. MPC821 GPCM–Peripheral Devices Basic Timing (ACS = 10, ACS = 11,TRLX = 0)**

The GPCM also provides an attribute that controls the negation timing of the appropriate strobe in write cycles. When this attribute (CSNT) is asserted, the strobe is negated one quarter of a clock before the normal case. For example, when ACS(0:1) == ‘00’ and CSNT == ‘1’, $\overline{\mathrm{WE}}$(0:3) is negated one quarter of a clock earlier and when ACS(0:1) <> ‘00’ and CSNT == ‘1’, $\overline{\mathrm{WE}}$(0:3) and $\overline{\mathrm{CS}}$ are negated one quarter of a clock earlier. For more information refer to Figure 15-6 and Figure 15-8.

The TRLX field is provided for memory systems that require more relaxed timing between signals. When TRLX is set and ACS(0:1) <> 00 an additional cycle between address and strobes ($\overline{\mathrm{CS}}$ line and $\overline{\mathrm{WE}}$/$\overline{\mathrm{OE}}$) is inserted by the MPC821 memory controller. Refer to Figure 15-9 for more information.

When TRLX is set and CSNT == ‘1’, in a write-memory access, the strobe lines ($\overline{\mathrm{WE}}$(0:3) and $\overline{\mathrm{CS}}$, if ACS(0:1) <> ‘00’) are negated one clock earlier than in the normal case. Refer to Figure 15-10, Figure 15-11, and Figure 15-12 for details. Notice that in the case of a bank selected to work with external transfer acknowledge (SETA == ‘1’) and TRLX == ‘1’, the memory controller does not support external devices providing $\overline{\mathrm{TA}}$ to complete the transfer with zero wait states. The minimum access duration in this case is 3 clock cycles.

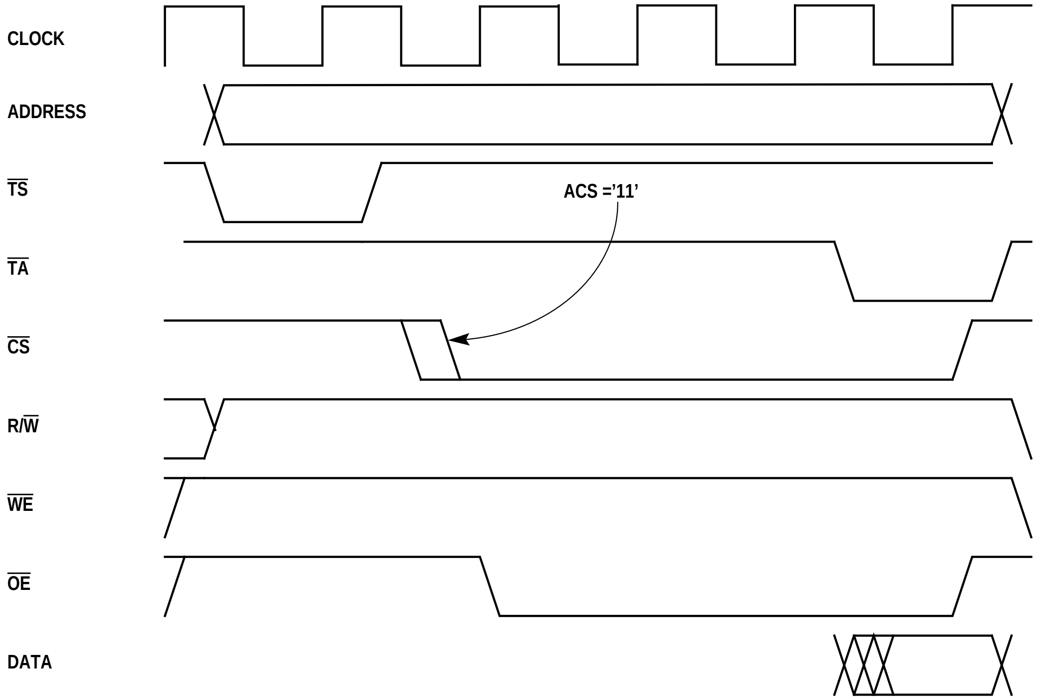


**Figure 15-9. MPC821 GPCM–Relaxed Timing–Read Access (ACS = 10, ACS = 11, SCY = 1, TRLX =1)**

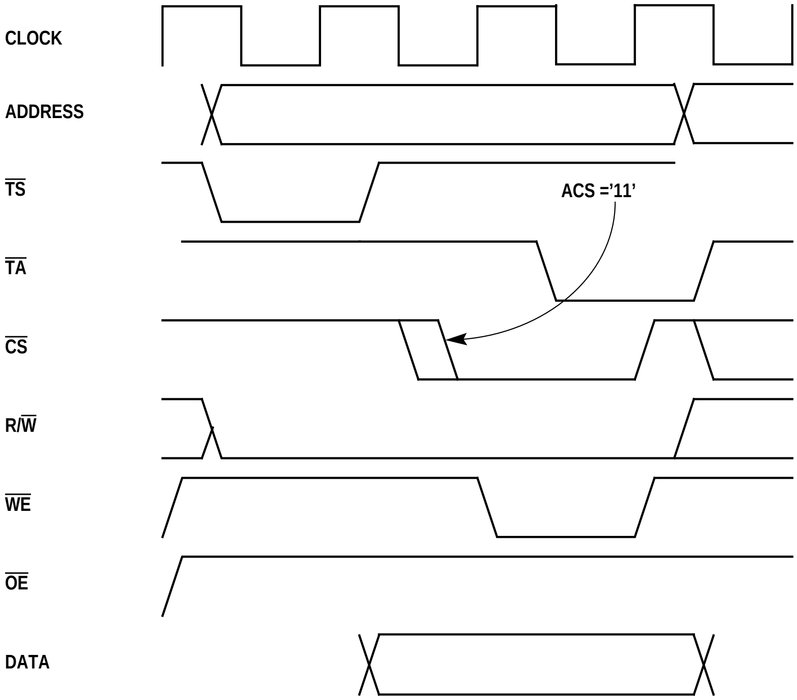


**Figure 15-10. MPC821 GPCM–Relaxed Timing–Write Access (ACS = 10, ACS = 11, SCY = 0, CSNT = 0, TRLX =1)**

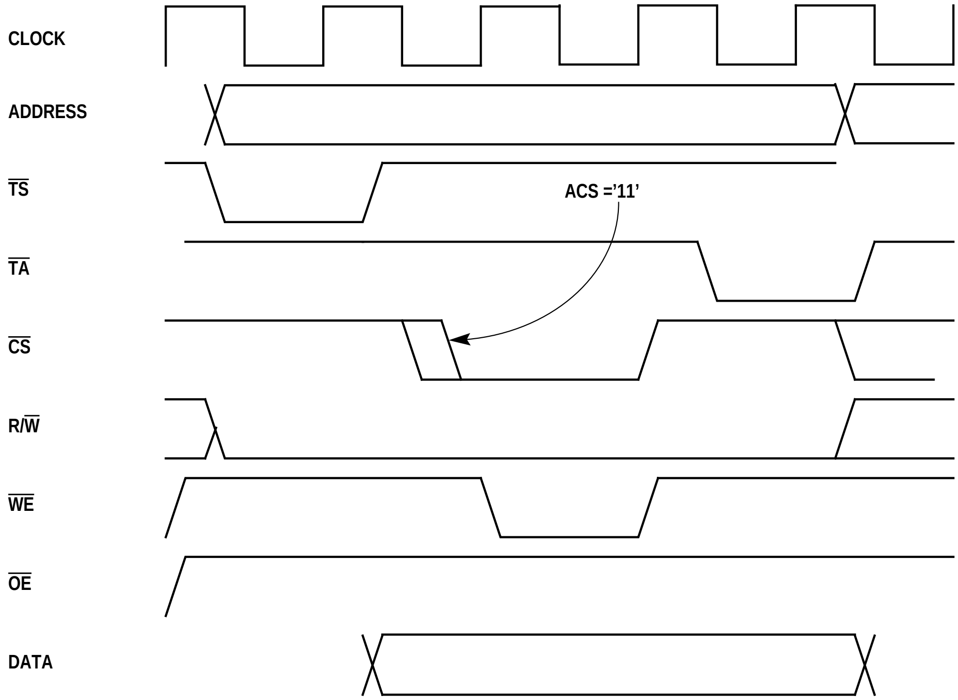


**Figure 15-11. MPC821 GPCM–Relaxed Timing–Write Access (ACS = 10, ACS = 11, SCY = 0, CSNT = 1, TRLX =1)**

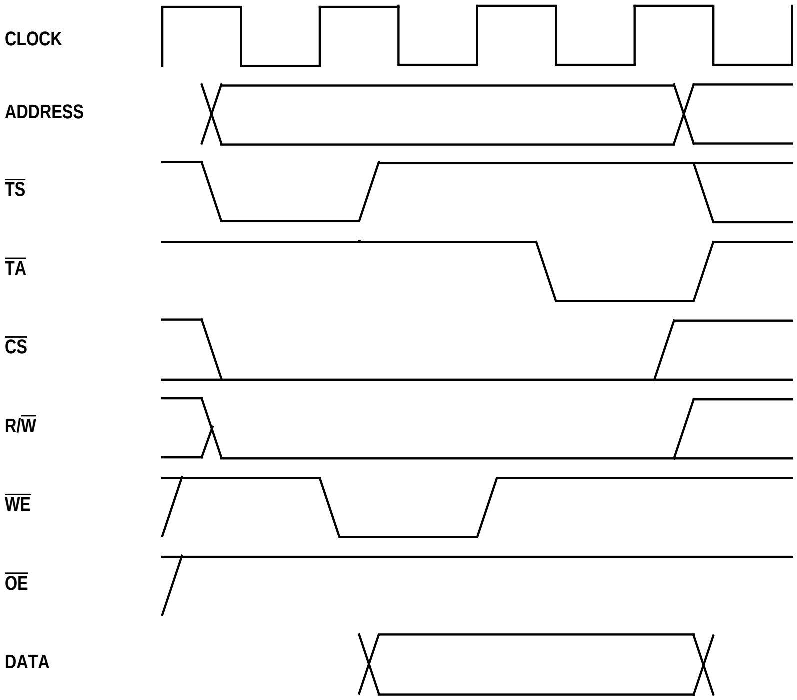


**Figure 15-12. MPC821 GPCM–Relaxed Timing–Write Access (ACS = 00, SCY = 0, CSNT = 1, TRLX =1**

**15.3.2.1 PROGRAMMABLE WAIT STATE CONFIGURATION.** The GPCM supports internal TA generation. It allows "fast" accesses (zero wait states) to external memory through an internal bus master or it allows a maximum of 17 clock accesses (15 wait states). This is programmed using the SCY bits in the option register. The internal TA generation mode will be enabled if the SETA bit in the OR is negated. If the TA pin is asserted externally at least two clock cycles before the wait states counter has expired, this terminates the current memory cycle. When TRLX is set, the number of wait states inserted by the memory controller is defined by NumberofWaitStates = 2 x SCY.

**15.3.2.2 EXTENDED HOLD TIME ON READ ACCESSES.** Slow memory devices requiring a long delay on data read accesses, should set EHTR in the corresponding OR register. In this case any MPC821 access to the external bus following a read access to the slower memory bank is delayed by one clock cycle, unless it is a read access to the same bank. Refer to Figures 15-13 through 15-16 for details.

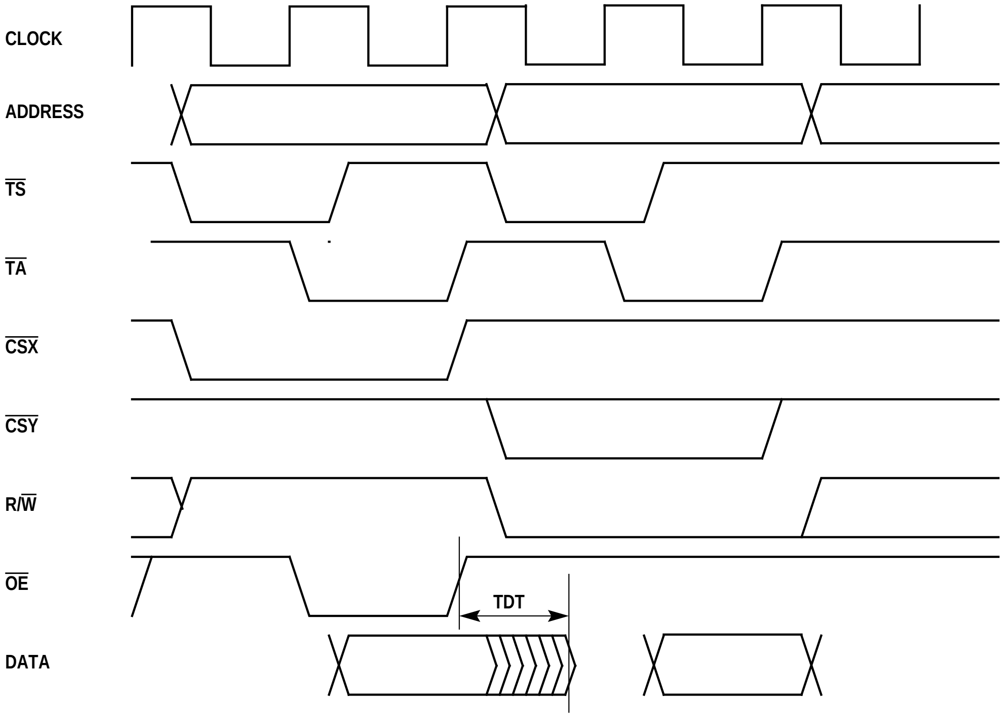


**Figure 15-13. MPC821 Consecutive Accesses Write After Read–(ORx-EHTR = 0)**

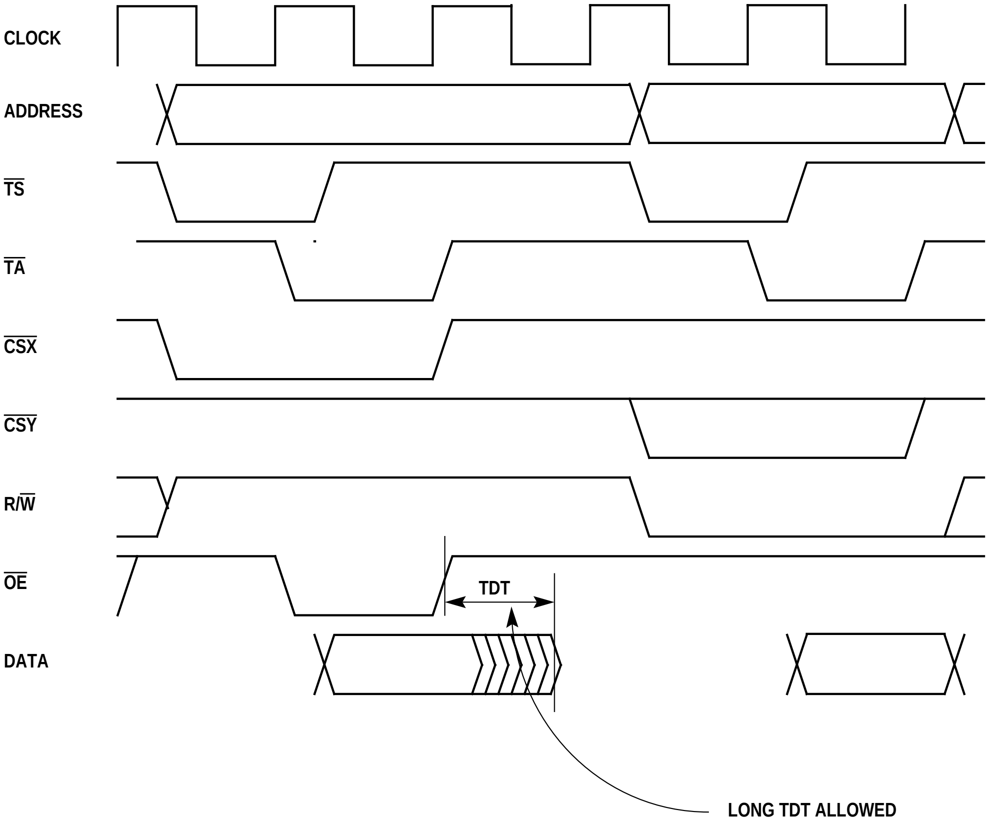


**Figure 15-14. MPC821 Consecutive Accesses Write After Read–(ORx-EHTR = 1)**

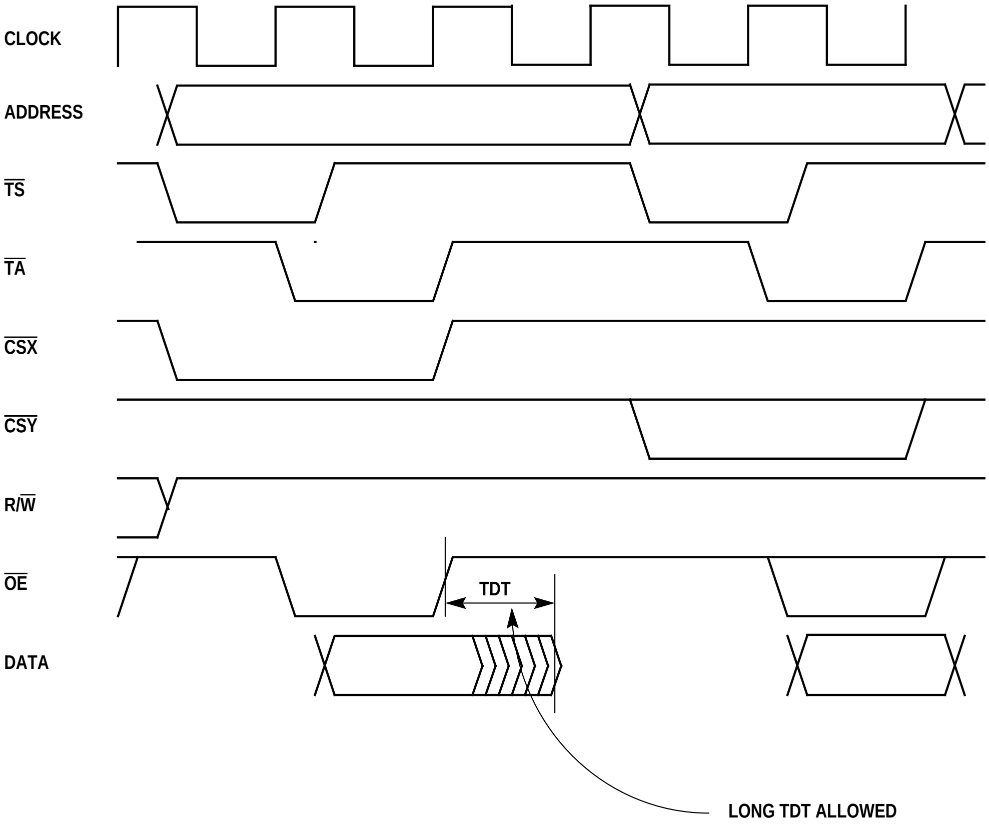


**Figure 15-15. MPC821 Consecutive Accesses Read After Read From Different Banks–(ORx-EHTR = 1)**

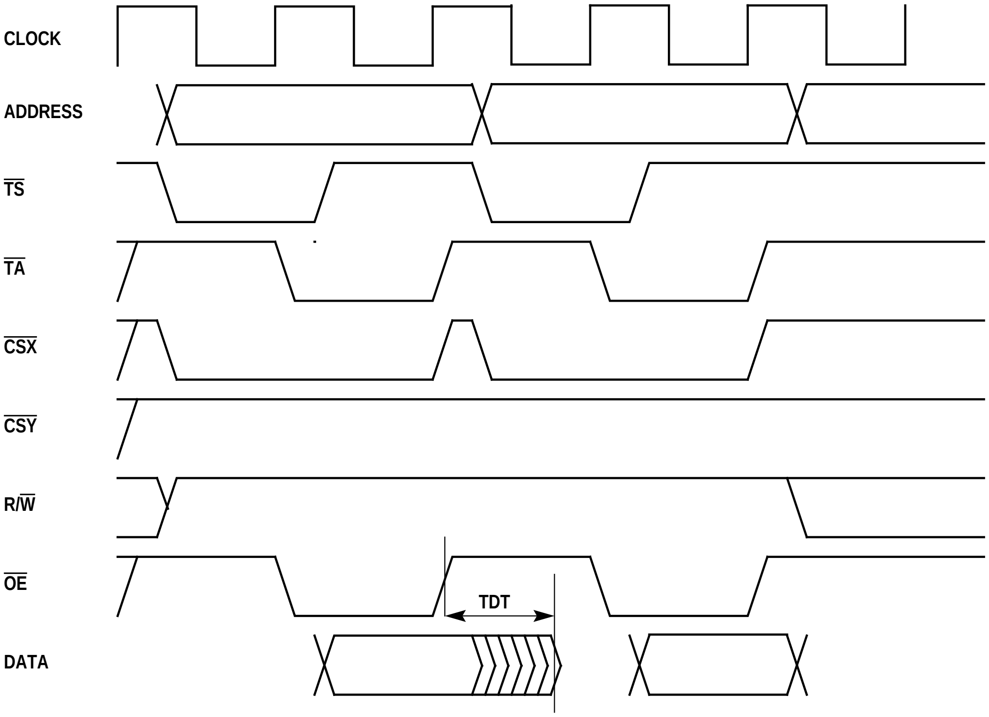


**Figure 15-16. MPC821 Consecutive Accesses Read After Read From Same Bank– (ORx-EHTR = 1)**

**15.3.2.3 GLOBAL (BOOT) CHIP-SELECT OPERATION.** Global (boot) chip-select operation allows address decoding for a boot ROM before system initialization occurs. $\overline{\text{CS0}}$ is the global chip-select output and its operation differs from the other external chip-select outputs on a system reset. When the MPC821 internal core begins accessing memory following a system reset, $\overline{\text{CS0}}$ is asserted for every address, unless an internal register is accessed.

The global chip-select provides a programmable port size during system reset by using the CONFIG pins. Setting the CONFIG pins appropriately allows a boot ROM to be located anywhere in the address space. The global chip-select does not provide write protection and responds to all address types. $\overline{\text{CS0}}$ operates in this way until the first write to the $\overline{\text{CS0}}$ option register (OR0) and it can be programmed to continue decoding a range of addresses once the preferred address range is loaded into base register 0 (BR0). After the first write to the OR0, the global chip-select can only be restarted on system reset. The initial values of the “boot bank” in the memory controller are described in Table 15-1.

**Table 15-1. Boot Bank Fields Values After Reset**

| FIELD | VALUE |
|---|---|
| PS | From Reset Configuration |
| PARE | 0 |
| WP | 0 |
| MS(0:1) | 00 |
| V | From Reset Configuration |
| AM(0:16) | 0x0 |
| ATM(0:2) | 0x0 |
| CSNT | 1 |
| ACS(0:1) | 11 |
| $\overline{\text{BI}}$ | 1 |
| SCY(0:3) | 1111 |
| SETA | 0 |
| TRLX | 1 |
| EHTR | 0 |

**15.3.2.4 SRAM INTERFACE.** Figure 15-17 illustrates a simple connection between a SRAM device and the MPC821.

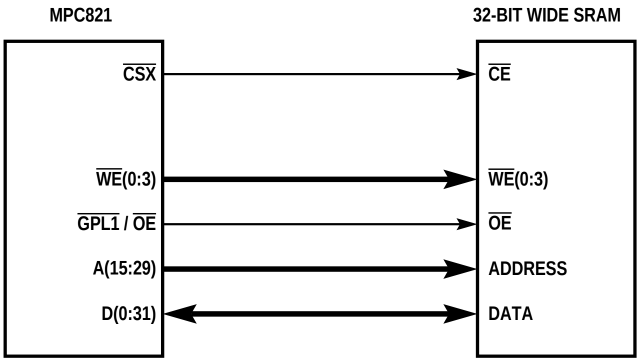


**Figure 15-17. MPC821–128 Kbyte SRAM Simple Configuration**

**15.3.2.5 GPCM- EXTERNAL ASYNCHRONOUS MASTER SUPPORT.** Figure 15-18 illustrates the basic interface between an asynchronous external master and the MPC821 to allow connection to a “static RAM” type of memory.

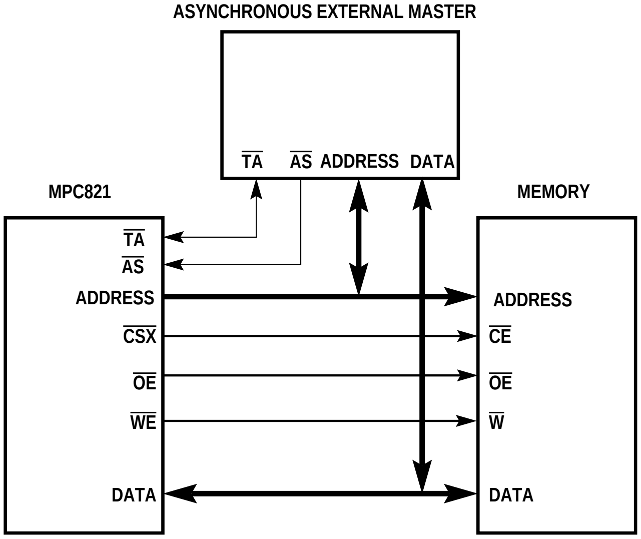


**Figure 15-18. MPC821–Asynchronous External Master Configuration For GPCM–Handled Memory Devices**

Figure 15-19 illustrates the timing for TRLX = 0 when an external asynchronous master accesses SRAM. Notice that the $\overline{\text{TA}}$ line remains asserted with the $\overline{\text{WE}}$ (if a write access is performed) and $\overline{\text{OE}}$ (if a read access is performed) until $\overline{\text{AS}}$ is negated by the external master.

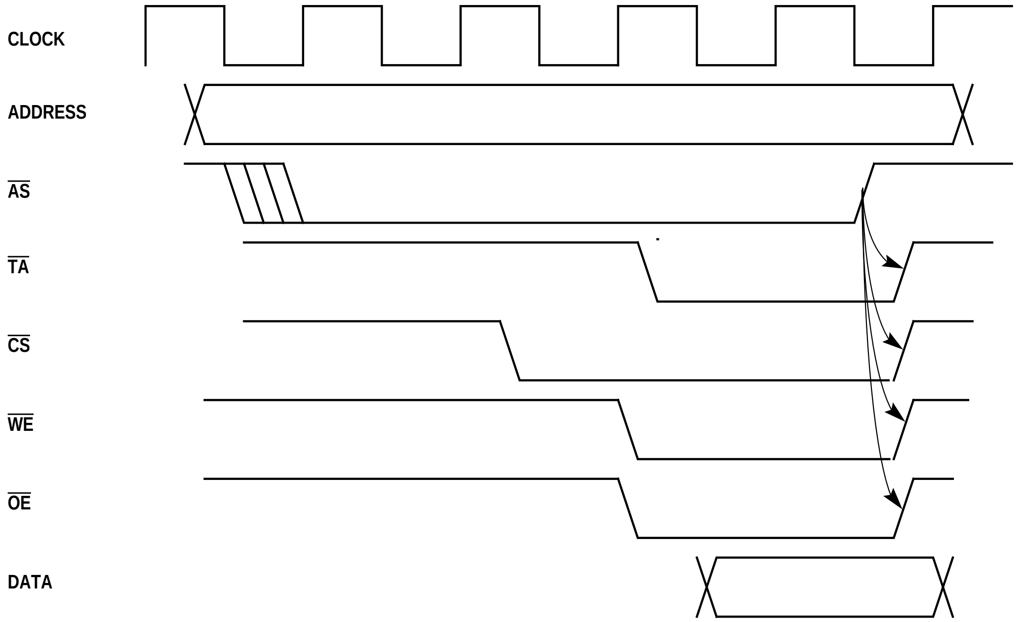


**Figure 15-19. Asynchronous Master GPCM–Memory Devices Basic Timing (TRLX = 0)**

When an external asynchronous master performs an access to a memory device by means of the GPCM in the MPC821 memory controller, the CSNT bit in the option register is configured as don’t care.

## 15.4 USER-PROGRAMMABLE MACHINE

The user-programmable machine (UPM) is a very flexible interface allowing connection to a wide range of memory devices. The basis of the UPM is an internal memory RAM that specifies what the logical value driven on the external memory controller pins are for a given clock cycle. Each word in the RAM provides bits that allow a memory access to be controlled with resolution of one quarter of the system clock period on byte-select and chip-select lines. Figure 15-20 illustrates the basic operation of the UPM. Three basic actions can initiate a UPM cycle:

- Any internal or external master requests an external memory access.
- An internal periodic timer expires, requesting a transaction.
- A valid command is written to the memory command register (MCR).

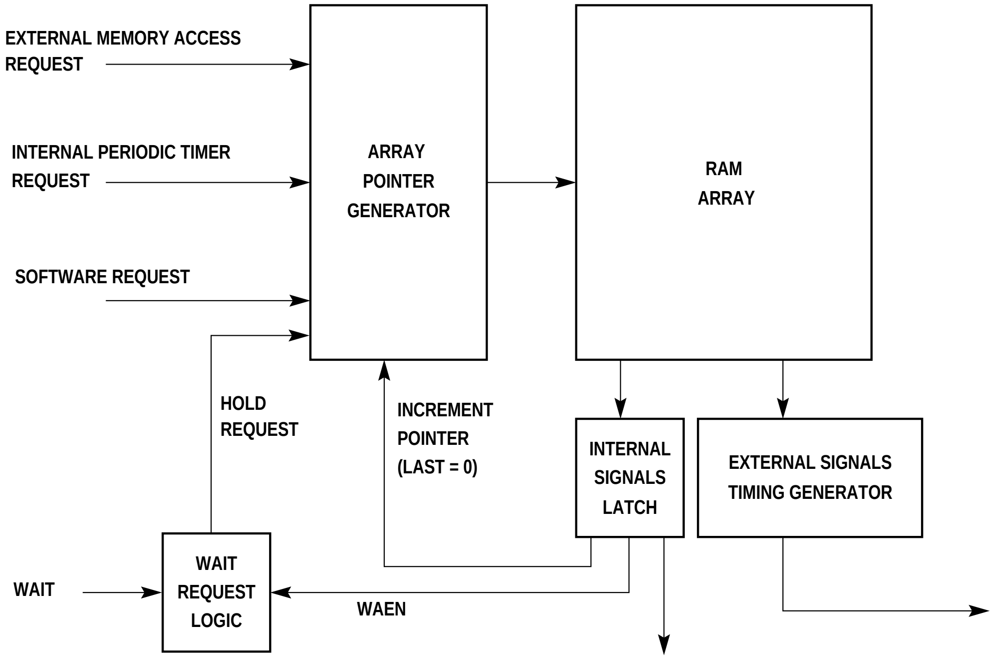

EXTERNAL MEMORY ACCESS REQUEST
INTERNAL PERIODIC TIMER REQUEST
SOFTWARE REQUEST
ARRAY POINTER GENERATOR
RAM ARRAY
HOLD REQUEST
INCREMENT POINTER (LAST = 0)
INTERNAL SIGNALS LATCH
EXTERNAL SIGNALS TIMING GENERATOR
WAIT
WAIT REQUEST LOGIC
WAEN

**Figure 15-20. General Description of a UPM**

When a new access to external memory is requested by any of the internal masters, the address of the transfer and the address type is compared to each one of the valid banks defined in the memory controller. When an address match is found in one of the memory banks, the MS bits in its base register select which UPM handles this memory access. In this case, a service request from the selected UPM, is required for this access.

The first location that is pointed to in the RAM array at the time a request is initiated is of four possible fixed addresses determined by the attributes of the requested cycle:

- — Read single beat start address (RSSA) RAM ADDRESS = 0x’00
- — Write single beat start address (WSSA) RAM ADDRESS = 0x’18
- — Read burst cycle start address (RBSA) RAM ADDRESS = 0x’08
- — Write burst cycle start address (WBSA) RAM ADDRESS = 0x’20

Each UPM has a machine mode register (MAMR and MBMR) and these registers define general attributes for the operation of the UPM machine. The PTA bits in MAMR and the PTB bits in MBMR define the period for the periodic timers associated with UPMA and UPMB. If the PTAE is asserted, the periodic timer of UPMA requests a transaction. If the PTBE is asserted, the periodic timer of UPMB requests a transaction.The first location pointed to in the RAM array when the PIT request is serviced is fixed at RAM ADDRESS = 0x’30. Figure 15-21 illustrates the hardware associated with the memory periodic timer request generation. In general, the periodic timer is used for refresh cycle operation.

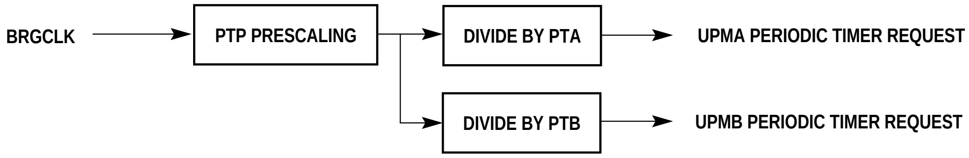


**Figure 15-21. Memory Periodic Timer Request Block Diagram**

The software can request a special service from the UPM by writing a valid command to the MCR and MDR. The commands allow the RAM to be read/written or to start running a pattern in the RAM from an arbitrary location. When a request is serviced, the RAM is read each clock cycle from consecutive addresses until the LAST bit in a RAM word is found. The words read from the RAM provide information about the value and timing of the external signals controlled by the UPM and specific strobes that control internal memory controller resources.

When the wait enable (WAEN) bit in the RAM word read is set, the external UPWAIT signal is sampled and synchronized by the memory controller. If it is asserted, the logical value of the external signals are frozen to the value defined in the last RAM word accessed and the RAM address increment is disabled until the UPWAIT signal is negated. This allows wait states to be inserted as required by an external device through an external signal. Associated with each UPM, is a memory disable timer (MDTA and MDTB). This timer counts down to zero starting at the value programmed in the DSA (DSB) field in the MAMR (MBMR) register. The one-shot timer trigger is controlled by the TODT in the RAM array. When an access to a memory bank controlled by the UPMx has the memory disable timer turned on, a new UPM access to this bank is held off until the timer expires. In general, the disable timer is a simple way to assure that a RAS precharge is met.

Each of the RAM arrays can control the way in which the address of the current access is output to the A(0:31) external pins. The address multiplex field (AMA(0:2) in MAMR and AMB(0:2) in MBMR) allows each of the UPMs to select an address multiplexing configuration. The AMX bits in the RAM array controls the multiplexing/nonmultiplexing value of the address pins in a cycle-by-cycle basis. The AMX bits can also control whether or not to output the memory address register (MAR) contents to the external address pins.

The RAM word includes bits that specify the value for the various external signals at each clock edge. The external signal timing generator causes the external signals to behave according to the pattern specified in the current word. Figure 15-22 and Figure 15-23 show the clock scheme of the user-programmable machines in the memory controller. The value of the external signals can be changed if specified in the RAM) after any of the edges of GCLK1 and GCLK2, plus a circuit delay time as specified in Table 21-1.

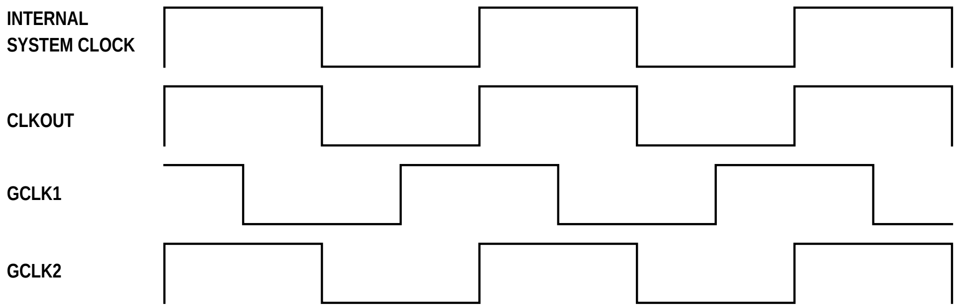


**Figure 15-22. Memory Controller UPM Clock Scheme (For System_To CLKOUT Division Factor 1–EBDF = 00)**

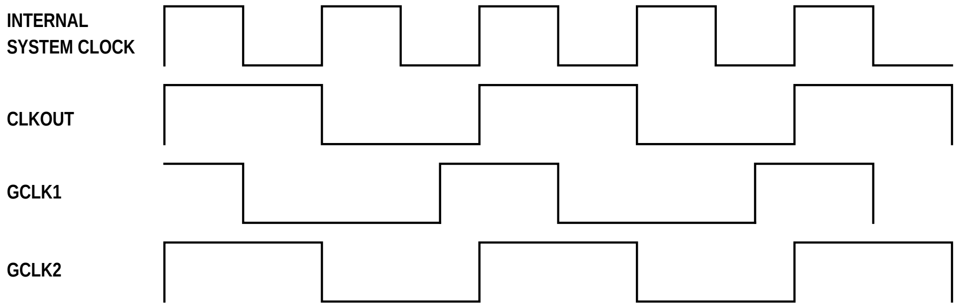


**Figure 15-23. Memory Controller UPM Clock Scheme (For System_To CLKOUT Division Factor 2–EBDF = 01)**

The $\overline{\mathrm{CS}}$ lines are handled in a similar way, except that only the $\overline{\mathrm{CS}}$ line corresponding to the currently accessed bank is modified. The byte select lines assertion/negation timing is also specified for each cycle in the RAM, but the final value of each one of these lines depends on the port size of the specified bank, the external address accessed, and the value of the transfer size (TSIZ pins).

Figure 15-24 and Figure 15-25 provide examples on how the timing of the $\overline{\mathrm{CSx}}$ line and the $\overline{\mathrm{GPL1}}$ and $\overline{\mathrm{GPL2}}$ pins can be controlled. A word is read from the RAM that specifies on every clock cycle the logical bits CST4, CST1, CST2, CST3, G1T4, G1T3, G2T4, and G2T3. These bits indicate what the electrical value will be for the corresponding output pins at the appropriate timing.

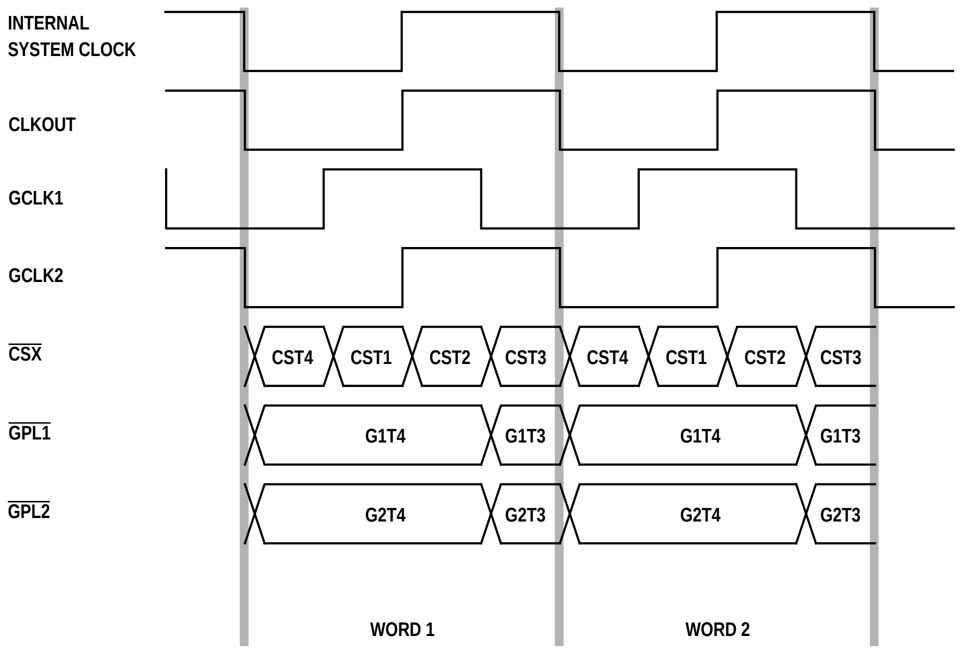


**Figure 15-24. UPM Signals Timing Example (For System_To CLKOUT Division Factor 1–EBDF = 00)**

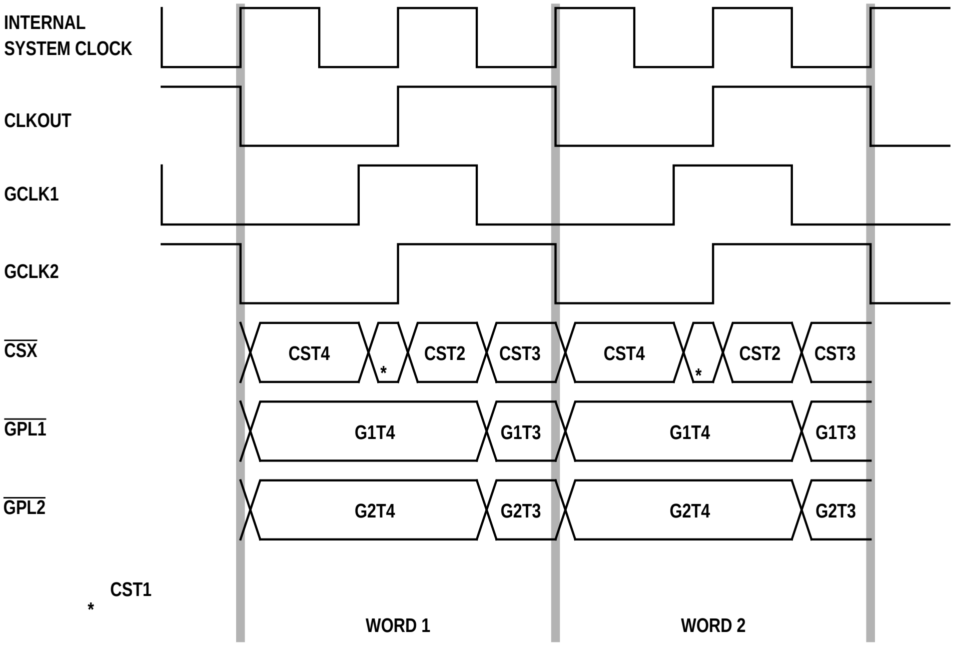


**Figure 15-25. UPM Signals Timing Example (For System_To CLKOUT Division Factor 2–EBDF = 01)**

The RAM array size for each UPM is 64 locations deep and 32 bits wide. Refer to Figure 15-26 for more information.

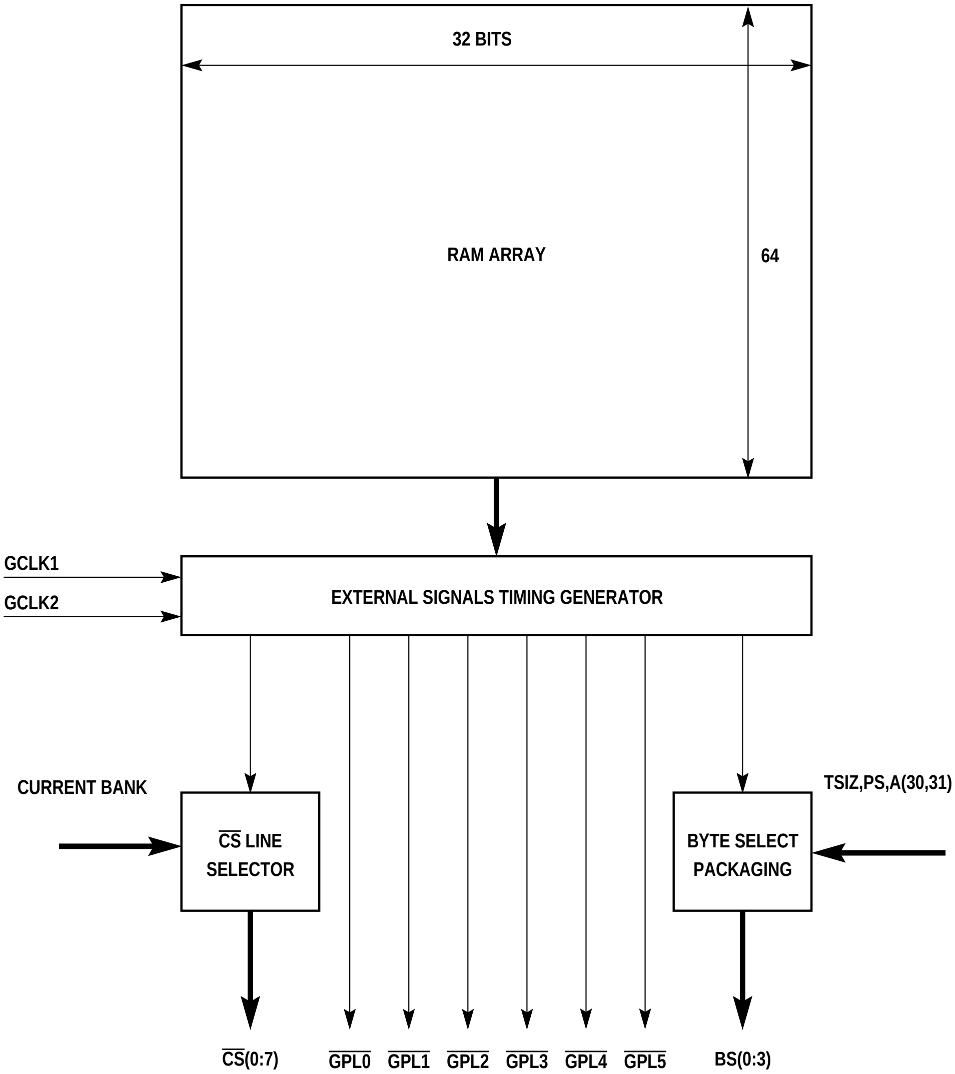


**Figure 15-26. User-Programmable Machine External Signals Generation**

**15.4.0.6 RAM WORD STRUCTURE AND TIMING SPECIFICATION.** The RAM word structure is illustrated in Figure 15-27 and described in Table 15-2. Notice that in the case of typical DRAM, the $\overline{CS}$ lines correspond to RAS and the $\overline{BS}$ lines correspond to CAS. Likewise, the $\overline{GPL}$ lines can be used as output enables.

| BIT 0 | BIT 1 | BIT 2 | BIT 3 | BIT 4 | BIT 5 | BIT 6 | BIT 7 |
|---|---|---|---|---|---|---|---|
| CST4 | CST1 | CST2 | CST3 | BST4 | BST1 | BST2 | BST3 |

| BIT 8 | BIT 9 | BIT 10 | BIT 11 | BIT 12 | BIT 13 | BIT 14 | BIT 15 |
|---|---|---|---|---|---|---|---|
| G0L0 | G0L1 | G0H0 | G0H1 | G1T4 | G1T3 | G2T4 | G2T3 |

| BIT 16 | BIT 17 | BIT 18 | BIT 19 | BIT 20 | BIT 21 | BIT 22 | BIT 23 |
|---|---|---|---|---|---|---|---|
| G3T4 | G3T3 | G4T4/DLT3 | G4T3/WAEN | G5T4 | G5T3 | | |

| BIT 24 | BIT 25 | BIT 26 | BIT 27 | BIT 28 | BIT 29 | BIT 30 | BIT 31 |
|---|---|---|---|---|---|---|---|
| LOOP | EXEN | AMX0 | AMX1 | NA | UTA | TODT | LAST |

**Figure 15-27. RAM Word Structure**

**Table 15-2. UPM RAM Word**

| BITS | MNEMONIC | FUNCTION |
|---|---|---|
| 0 | CST4 | CST4 = 0 the value of the $\overline{CS}$ line at the trailing edge of GCLK2 will be ‘0’<br>CST4 = 1 the value of the $\overline{CS}$ line at the trailing edge of GCLK2 will be ‘1’ |
| 1 | CST1 | CST1 = 0 the value of the $\overline{CS}$ line at the rising edge of GCLK1 will be ‘0’<br>CST1 = 1 the value of the $\overline{CS}$ line at the rising edge of GCLK1 will be ‘1’ |
| 2 | CST2 | CST2 = 0 the value of the $\overline{CS}$ line at the rising edge of GCLK2 will be ‘0’<br>CST2 = 1 the value of the $\overline{CS}$ line at the rising edge of GCLK2 will be ‘1’ |
| 3 | CST3 | CST3 = 0 the value of the $\overline{CS}$ line at the trailing edge of GCLK1 will be ‘0’<br>CST3 = 1 the value of the $\overline{CS}$ line at the trailing edge of GCLK1 will be ‘1’ |
| 4 | BST4 | BST4 = 0 the value of the $\overline{BS}$ lines at the trailing edge of GCLK2 will be ‘0’<br>BST4 = 1 the value of the $\overline{BS}$ lines at the trailing edge of GCLK2 will be ‘1’<br>NOTE: The final value of the $\overline{BS}$ lines depends on the value of the PS bits of the BR accessed, the value of the TSIZ lines for the access and the value of the address lines A(30:31) For more information on $\overline{BS}$ lines, see **Section 15.4.0.8 Byte Selects**. |

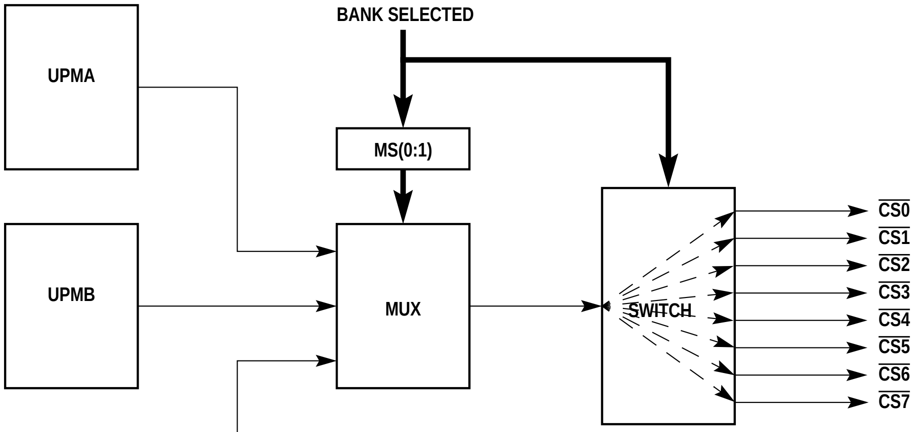
## Table 15-2. UPM RAM Word (Continued)

| BITS | MNEMONIC | FUNCTION |
|---|---|---|
| 5 | BST1 | BST1 = 0 the value of the $\overline{\text{BS}}$ lines at the rising edge of GCLK1 will be ‘0’<br>BST1 = 1 the value of the $\overline{\text{BS}}$ lines at the rising edge of GCLK1 will be ‘1’<br>NOTE: The final value of the $\overline{\text{BS}}$ lines depends on the value of the PS bits of the BR accessed, the value of the TSIZ lines for the access and the value of the address lines A(30:31) |
| 6 | BST2 | BST2 = 0 the value of the $\overline{\text{BS}}$ lines at the rising edge of GCLK2 will be ‘0’<br>BST2 = 1 the value of the $\overline{\text{BS}}$ lines at the rising edge of GCLK2 will be ‘1’<br>NOTE: The final value of the $\overline{\text{BS}}$ lines depends on the value of the PS bits of the BR accessed, the value of the TSIZ lines for the access, and the value of the address lines A(30:31) |
| 7 | BST3 | BST3 = 0 the value of the $\overline{\text{BS}}$ lines at the trailing edge of GCLK1 will be ‘0’<br>BST3 = 1 the value of the $\overline{\text{BS}}$ lines at the trailing edge of GCLK1 will be ‘1’<br>NOTE: The final value of the $\overline{\text{BS}}$ lines depends on the value of the PS bits of the BR accessed, the value of the TSIZ lines for the access, and the value of the address lines A(30:31) |
| 8-9 | G0L(0:1) | G0L = 10 the value of the $\overline{\text{GPL0}}$ line at the trailing edge of GCLK2 will be ‘0’<br>G0L = 11 the value of the $\overline{\text{GPL0}}$ line at the trailing edge of GCLK2 will be ‘1’<br>G0L = 00 the value of the $\overline{\text{GPL0}}$ line at the trailing edge of GCLK2 will be as defined in the G0CL field in the MxMR’ |
| 10-11 | G0H(0:1) | G0H = 10 the value of the $\overline{\text{GPL0}}$ line at the trailing edge of GCLK1 will be ‘0’<br>G0H = 11 the value of the $\overline{\text{GPL0}}$ line at the trailing edge of GCLK1 will be ‘1’<br>G0H = 00 the value of the $\overline{\text{GPL0}}$ line at the trailing edge of GCLK1 will be as defined in the G0CL field in the MxMR’ |
| 12 | G1T4 | G1T4 = 0 the value of the $\overline{\text{GPL1}}$ line at the trailing edge of GCLK2 will be ‘0’<br>G1T4 = 1 the value of the $\overline{\text{GPL1}}$ line at the trailing edge of GCLK2 will be ‘1’ |
| 13 | G1T3 | G1T3 = 0 the value of the $\overline{\text{GPL1}}$ line at the trailing edge of GCLK1 will be ‘0’<br>G1T3 = 1 the value of the $\overline{\text{GPL1}}$ line at the trailing edge of GCLK1 will be ‘1’ |
| 14 | G2T4 | G2T4 = 0 the value of the $\overline{\text{GPL2}}$ line at the trailing edge of GCLK2 will be ‘0’<br>G2T4 = 1 the value of the $\overline{\text{GPL2}}$ line at the trailing edge of GCLK2 will be ‘1’ |
| 15 | G2T3 | G2T3 = 0 the value of the $\overline{\text{GPL2}}$ line at the trailing edge of GCLK1 will be ‘0’<br>G2T3 = 1 the value of the $\overline{\text{GPL2}}$ line at the trailing edge of GCLK1 will be ‘1’ |
| 16 | G3T4 | G3T4 = 0 the value of the $\overline{\text{GPL3}}$ line at the trailing edge of GCLK2 will be ‘0’<br>G3T4 = 1 the value of the $\overline{\text{GPL3}}$ line at the trailing edge of GCLK2 will be ‘1’ |
| 17 | G3T3 | G3T3 = 0 the value of the $\overline{\text{GPL3}}$ line at the trailing edge of GCLK1 will be ‘0’<br>G3T3 = 1 the value of the $\overline{\text{GPL3}}$ line at the trailing edge of GCLK1 will be ‘1’ |
| 18 | G4T4/DLT3 | When GPL4_xDIS = 0 in the corresponding MxMR register:<br>G4T4/DLT3 = 0 the value of the $\overline{\text{GPL4}}$ line at the trailing edge of GCLK2 will be ‘0’<br>G4T4/DLT3 = 1 the value of the $\overline{\text{GPL4}}$ line at the trailing edge of GCLK2 will be ‘1’<br><br>When GPL4_xDIS = 1 in the corresponding MxMR register:<br>G4T4/DLT3 = 1 in the current word, indicates that the data bus should be sampled at the falling edge of GCLK2 (if a read burst or a single read service is executed).<br>G4T4/DLT3 = 0 in the current word, indicates that the data bus should be sampled at the rising edge of GCLK2 (if a read burst or a single read service is executed). |

### Table 15-2. UPM RAM Word (Continued)

| BITS | MNEMONIC | FUNCTION |
|---|---|---|
| 19 | G4T3/WAEN | When GPL4_xDIS = 0 in the corresponding MxMR register:<br>G4T3/WAEN = 0 the value of the $\overline{\text{GPL4}}$ line at the trailing edge of GCLK1 will be ‘0’<br>G4T3/WAEN = 1 the value of the $\overline{\text{GPL4}}$ line at the trailing edge of GCLK1 will be ‘1’<br><br>When GPL4_xDIS = 1 in the corresponding MxMR register:<br>G4T3/WAEN = 1 in the current word indicates that a “freeze” in the external signals logical value will occur if the external $\overline{\text{WAIT}}$ signal is detected asserted. This condition lasts until the $\overline{\text{WAIT}}$ signal is negated. |
| 20 | G5T4 | G5T4 = 0 the value of the $\overline{\text{GPL5}}$ line at the trailing edge of GCLK2 will be ‘0’<br>G5T4 = 1 the value of the $\overline{\text{GPL5}}$ line at the trailing edge of GCLK2 will be ‘1’ |
| 21 | G5T3 | G5T3 = 0 the value of the $\overline{\text{GPL5}}$ line at the trailing edge of GCLK1 will be ‘0’<br>G5T3 = 1 the value of the $\overline{\text{GPL5}}$ line at the trailing edge of GCLK1 will be ‘1’ |
| 22-23 | Reserved | — |
| 24 | LOOP | LOOP = 1 Indicates that the current word is the start or end of a loop subpattern. The first word in a pattern in which the LOOP bit is ‘1’ is marked as the LOOP START WORD. The next word in the same pattern in which the LOOP bit is ‘1’ is marked as the LOOP END WORD. The UPM runs the subpattern between the LOOP START WORD and the LOOP END WORD a number of times as defined in the corresponding loop field in the MxMR register. |
| 25 | EXEN | EXEN = 1 in the current word indicates that a “branch” to the exception pattern is enabled after the current cycle if an exception condition is detected. The exception condition can be an external device asserting $\overline{\text{TEA}}$ or an external reset request. |
| 26-27 | AMX(0:1) | AMX = 00 the value of the address lines A(0:31) at the trailing edge of GCLK1 will be the address requested by the internal master for the external access. Ex: Row address.<br>AMX = 10 the value of the address lines A(0:31) at the trailing edge of GCLK1 will be the address requested by the internal master for the external access multiplexed according to the specified in the AMA (AMB) bits in the MAMR (MBMR) register. Ex: Column address.<br>AMX = 11 the value of the address lines A(0:31) at the trailing edge of GCLK1 will be the contents of the memory address register (MAR). Ex: UPM word or pattern. |
| 28 | NA | NA = 1, if the port size of the accessed bank is 32 bits, the value of the address lines A(28:31) at the trailing edge of GCLK1 will be incremented by 4.<br>if the port size of the accessed bank is 16 bits, the value of the address lines A(28:31) at the trailing edge of GCLK1 will be incremented by 2.<br>if the port size of the accessed bank is 8 bits, the value of the address lines A(28:31) at the trailing edge of GCLK1 will be incremented by 1.<br>NA = 0, the address increment is disabled.<br>NOTE: The value of the NA bit is relevant only when the UPM serves a burst-read request or a burst-write request. Under other patterns this bit is reserved. |
| 29 | UTA | This line indicates the value of the $\overline{\text{TA}}$ line sampled by the SIU in the current cycle. The $\overline{\text{TA}}$ line is output at the rising edge of GCLK2. |
| 30 | TODT | TODT = 1 the disable timer for the current accessed bank is turned on. This avoids a new access to the same bank (when controlled by any of the UPMs) until the disable timer is expired. Ex: Precharge time. |
| 31 | LAST | LAST = 1 the service to the UPM request is done |

**15.4.0.7 $\overline{\text{CS}}$ LINES.** If the MS bits in the BRx of the accessed memory bank selects the UPM machine on the currently requested cycle, the UPM can only affect the electrical value of the $\overline{\text{CSx}}$ line and the timing of the change is specified in the UPM internal RAM. Figure 15-28 illustrates how the $\overline{\text{CS}}$ lines are controlled by the UPMs.



| MS(0:1) | $\overline{\text{CS}}$ |
|---|---|
| 00 | GPCM |
| 01 | X |
| 10 | UMPA |
| 11 | UMPB |

**Figure 15-28. $\overline{\text{CS}}$ Control Model**

**15.4.0.8 BYTE SELECTS.** If the MS bits in the BRx of the memory bank being accessed selects the UPMA (UPMB) machine to handle the current cycle, the UPMA (UPMB) only determines the timing and value of the BS signals if allowed by the port size of the accessed bank, the transfer size of the transaction, and the address accessed. Figure 15-29 explains how the byte select lines are controlled by the UPMs.

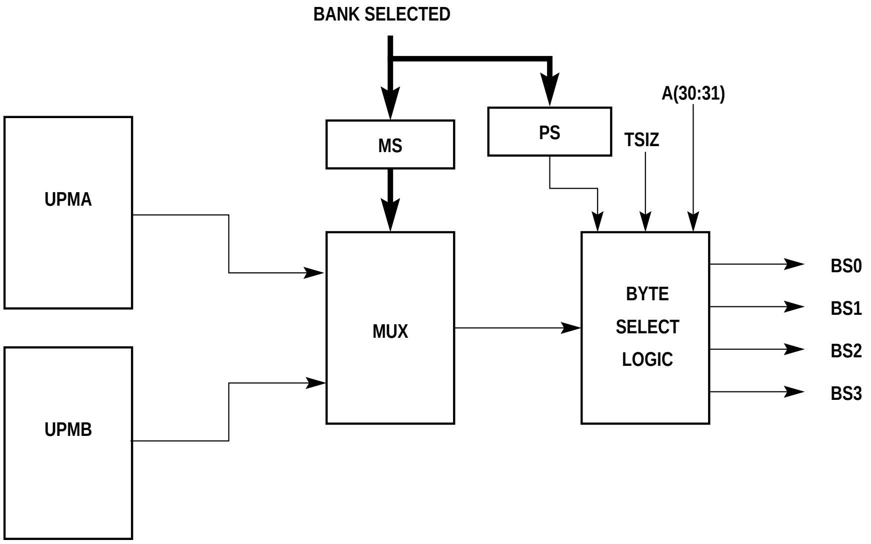


**Figure 15-29. Byte Select Control Model**

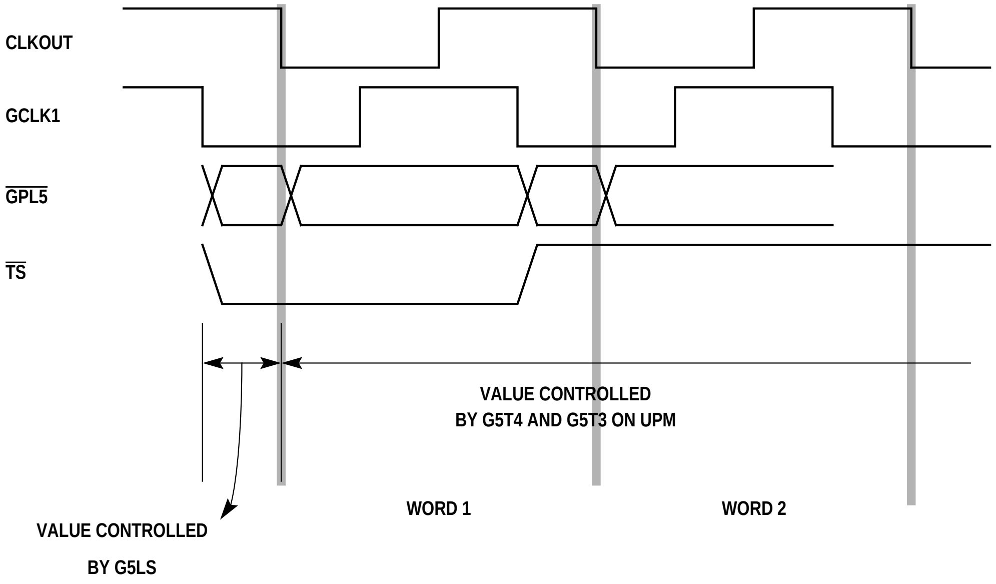
The upper-upper byte select (BS0) indicates that the upper eight bits of the data bus (D0-D7) contain valid data during a cycle and the upper-middle write enable (BS1) indicates that the upper-middle eight bits of the data bus (D8-D15) contain valid data during a cycle. The lower-middle write enable (BS2) indicates that the lower-middle eight bits of the data bus (D16-D23) contain valid data during a cycle and the lower-lower write enable (BS3) indicates that the lower eight bits of the data bus contain valid data during a cycle. The manner in which the byte select lines are affected in a transaction for 32-bit port, 16-bit port, and 8-bit port are shown in Table 15-3. It should be noted that for a periodic timer request and a memory command request, the BS lines are only determined by the port size of the bank.

### Table 15-3. Byte Selects Enable Function

| 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 | | | |

**15.4.0.9 GENERAL-PURPOSE LINES.** The UPM controls each of the $\overline{\text{GPL}}$ lines via two bits in the UPM word. These two bits define the logical value of the line to be changed at the falling edge of GCLK2 and/or at the falling edge of GCLK1. $\overline{\text{GPL5}}$ and $\overline{\text{GPL0}}$ offer enhancements beyond the other $\overline{\text{GPLx}}$ lines:

- The logical value $\overline{\text{GPL5}}$ can be controlled at the falling edge of GCLK1 in the first clock cycle of a write or read memory access, according to the value of GL5S in the corresponding option register.



- The $\overline{\text{GPL0}}$ line can output the value of an address line as specified in the corresponding MxMR while under UPM control. This is helpful when some control is needed over the value output to the address lines that connect to some types of memory devices.

**15.4.0.10 LOOP CONTROL BIT.** The LOOP bit in the UPM allows the user to run a repetitive subpatterns included in a memory cycle pattern a specific number of times. The first word in which the LOOP bit is found asserted in a pattern is marked by the memory controller as the LOOP START WORD. At this time, the memory LOOP counter is loaded with the corresponding contents of the LOOP field. The next word in which the LOOP bit is found asserted in the same pattern is marked by the memory controller as the LOOP END WORD. At this time, the memory LOOP counter is decremented by one.

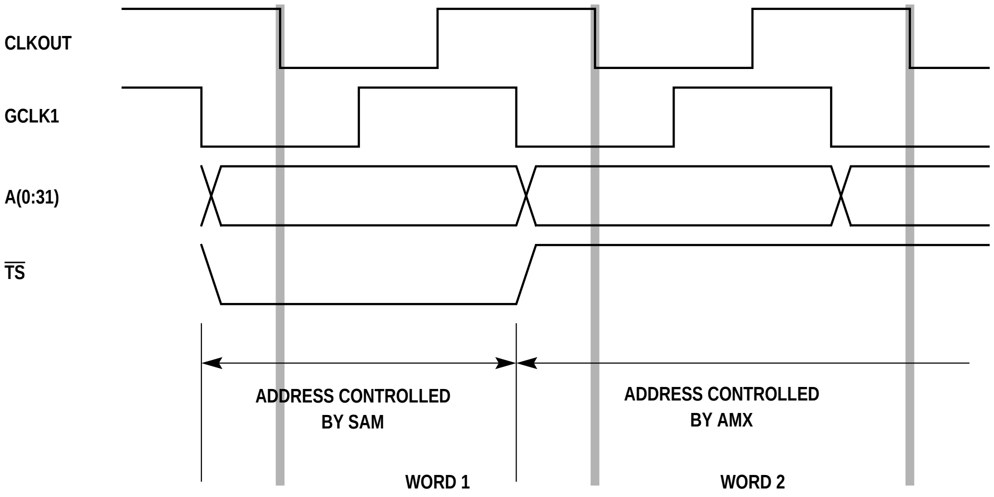
Whether or not the word following the LOOP END WORD is run depends on the value of the memory LOOP counter. If it is not zero, the next word is the LOOP START WORD, if it is zero the UPM continues with the word following the LOOP END WORD. After exiting a loop, the next word read in the UPM with a LOOP bit set, is marked as the new LOOP START WORD of the new loop. Every time the LAST bit is found in a pattern, the loop condition is reset. The LOOP field loaded into the loop counter when a request is serviced by the UPM. The decoding of the loop bits is shown in Table 15-4.

**Table 15-4. Loop Field For UPM Service Requests**

| REQUEST SERVICED | UPM LOOP FIELD LOADED |
|---|---|
| Read Single Beat Cycle | RLFx |
| Read Burst Cycle | RLFx |
| Write Single Beat Cycle | WLFx |
| Write Burst Cycle | WLFx |
| Periodic Timer Expired | TLFx |

**15.4.0.11 EXCEPTION HANDLING.** When an access to a memory device is initiated by the MPC821 under control of a UPM on the memory controller, there may be a case in which an external device asserts the $\overline{\text{TEA}}$ or RESET line. The MPC821 attempts to close the bus transfer immediately. The UPM in the memory controller provides a mechanism by which the memory control lines can be handled as required by the user to meet the timing requirements of the device and assume no data is lost. When one of the exceptions mentioned above is recognized, and the EXEN in the UPM is “1”, the next word read and run by the UPM is found at the fixed address “exception start address”—EXSA (RAM ADDRESS = 0x’3C). Normally, a pattern is at this location that allows the immediate negation of the control signals. If the EXEN bit is “0” the UPM continues with the remaining words until the EXEN bit is “1” and a branch to the exception start address is performed or until the LAST bit is read by the UPM. When the “branch” to the EXSA is performed, the UPM continues reading from successive locations until the LAST bit is found equal to 1 in a UPM word.

**15.4.0.12 ADDRESS CONTROL BITS.** The address lines can be controlled by the pattern written by the user in the UPM. The AMA/AMB bits can choose whether between outputting an address requested by the internal master as is, or to output it according to the multiplexing specified by the AMA (AMB) bits in the machine mode register. See Table 15-5 for details. The last option is to output the contents of the MAR on the address pins. Notice that the address for the first clock cycle of a write or read memory access is generated according to the value of SAM in the corresponding option register.



### Table 15-5. Address Multiplexing

| AMA/ AMB | A16 | A17 | A18 | A19 | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | ▨ | ▨ | 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 15-6 shows how the AMA/AMB bits can be defined to interface with a wide range of DRAM modules.

Table 15-6. AMA/AMB Definition For DRAM Interface

| WIDTH | SIZE (KB) | NUMBER OF ROW ADDRESS LINES | NUMBER OF COLUMN ADDRESS LINES | ADDRESS CONNECTION | AMA/AMB |
|---|---|---|---|---|---|
| 8 Bits | 64k | 8 | 8 | A24 - A31 | 000 |
| | 128k | 9 | | A23 - A31 | |
| | 256k | 10 | | A22 - A31 | |
| | 512k | 11 | | A21 - A31 | |
| | 1M | 12 | | A20 - A31 | |
| | 2M | 13 | | A19 - A31 | |
| | 4M | 14 | | A18 - A31 | |
| | 256k | 9 | 9 | A23 - A31 | 001 |
| | 512k | 10 | | A22 - A31 | |
| | 1M | 11 | | A21 - A31 | |
| | 2M | 12 | | A20 - A31 | |
| | 4M | 13 | | A19 - A31 | |
| | 8M | 14 | | A18 - A31 | |
| | 16M | 15 | | A17 - A31 | |
| | 1M | 10 | 10 | A22 - A31 | 010 |
| | 2M | 11 | | A21 - A31 | |
| | 4M | 12 | | A20 - A31 | |
| | 8M | 13 | | A19 - A31 | |
| | 16M | 14 | | A18 - A31 | |
| | 32M | 15 | | A17 - A31 | |
| | 64M | 16 | | A16 - A31 | |
| | 4M | 11 | 11 | A21 - A31 | 011 |
| | 8M | 12 | | A20 - A31 | |
| | 16M | 13 | | A19 - A31 | |
| | 32M | 14 | | A18 - A31 | |
| | 64M | 15 | | A17 - A31 | |
| | 16M | 12 | 12 | A20 - A31 | 100 |
| | 32M | 13 | | A19 - A31 | |
| | 64M | 14 | | A18 - A31 | |
| | 128M | 15 | | A17 - A31 | |
| | 256M | 16 | | A16 - A31 | |

### Table 15-6. AMA/AMB Definition For DRAM Interface (Continued)

| WIDTH | SIZE (KB) | NUMBER OF ROW ADDRESS LINES | NUMBER OF COLUMN ADDRESS LINES | ADDRESS CONNECTION | AMA/AMB |
|---|---|---|---|---|---|
| 8 Bits | 64M | 13 | 13 | A19 - A31 | 101 |
| | 128M | 14 | | A18 - A31 | |
| | 256M | 15 | | A17 - A31 | |
| 16 Bits | 128k | 8 | 8 | A23 - A30 | 000 |
| | 256k | 9 | | A22 - A30 | |
| | 512k | 10 | | A21 - A30 | |
| | 1M | 11 | | A20 - A30 | |
| | 2M | 12 | | A19 - A30 | |
| | 4M | 13 | | A18 - A30 | |
| | 512k | 9 | 9 | A22 - A30 | 001 |
| | 1M | 10 | | A21 - A30 | |
| | 2M | 11 | | A20 - A30 | |
| | 4M | 12 | | A19 - A30 | |
| | 8M | 13 | | A18 - A30 | |
| | 16M | 14 | | A17 - A30 | |
| | 2M | 10 | 10 | A21 - A30 | 010 |
| | 4M | 11 | | A20 - A30 | |
| | 8M | 12 | | A19 - A30 | |
| | 16M | 13 | | A18 - A30 | |
| | 32M | 14 | | A17 - A30 | |
| | 64M | 15 | | A16 - A30 | |
| | 8M | 11 | 11 | A20 - A30 | 011 |
| | 16M | 12 | | A19 - A30 | |
| | 32M | 13 | | A18 - A30 | |
| | 64M | 14 | | A17 - A30 | |
| | 32M | 12 | 12 | A19 - A30 | 100 |
| | 64M | 13 | | A18 - A30 | |
| | 128M | 14 | | A17 - A30 | |
| | 256M | 15 | | A16 - A30 | |
| | 128M | 13 | 13 | A18 - A30 | 101 |
| | 256M | 13 | | A17 - A30 | |

### Table 15-6. AMA/AMB Definition For DRAM Interface (Continued)

| WIDTH | SIZE (KB) | NUMBER OF ROW ADDRESS LINES | NUMBER OF COLUMN ADDRESS LINES | ADDRESS CONNECTION | AMA/AMB |
|---|---|---|---|---|---|
| 32 Bits | 256k | 8 | 8 | A22 - A29 | 000 |
| | 512k | 9 | | A21 - A29 | |
| | 1M | 10 | | A20 - A29 | |
| | 2M | 11 | | A19 - A29 | |
| | 4M | 12 | | A18 - A29 | |
| | 1M | 9 | 9 | A21 - A29 | 001 |
| | 2M | 10 | | A20 - A29 | |
| | 4M | 11 | | A19 - A29 | |
| | 8M | 12 | | A18 - A29 | |
| | 16M | 13 | | A17 - A29 | |
| | 4M | 10 | 10 | A20 - A29 | 010 |
| | 8M | 11 | | A19 - A29 | |
| | 16M | 12 | | A18 - A29 | |
| | 32M | 13 | | A17 - A29 | |
| | 64M | 14 | | A16 - A29 | |
| | 16M | 11 | 11 | A19 - A29 | 011 |
| | 32M | 12 | | A18 - A29 | |
| | 64M | 13 | | A17 - A29 | |
| | 64M | 12 | 12 | A18 - A29 | 100 |
| | 128M | 13 | | A17 - A29 | |
| | 256M | 14 | | A16 - A29 | |
| | 256M | 13 | 13 | A17 - A29 | 101 |

**15.4.0.13 DISABLE TIMER MECHANISM.** The disable timer associated with each UPM allows the user to guarantee a minimum time during which two successive accesses to the same memory bank can be disabled. This feature is critical in the case of DRAM that requires a RAS precharge time. The timer is turned on by the TODT bit in the RAM array and prevents UPM access to the same bank until the timer expires. Notice that if a different memory bank requests service from the UPM it is accommodated. To avoid conflicts between different banks accessing the same UPM, it is recommended that each pattern on the UPM be equal to or greater than (in clock cycles) than the defined disable timer period.

**15.4.0.14 TRANSFER ACKNOWLEDGE AND DATA SAMPLE CONTROL.** During UPM memory access, the value of the $\overline{TA}$ line driven by the memory controller and sampled by the external bus interface, is indicated in the UPM transfer acknowledge (UTA) bit in the UPM RAM word. The $\overline{TA}$ line is driven at the falling edge of the GCLK1 line. 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 MPC821. Figure 15-30 illustrates a schematic description of the hardware controlled by the UPM.

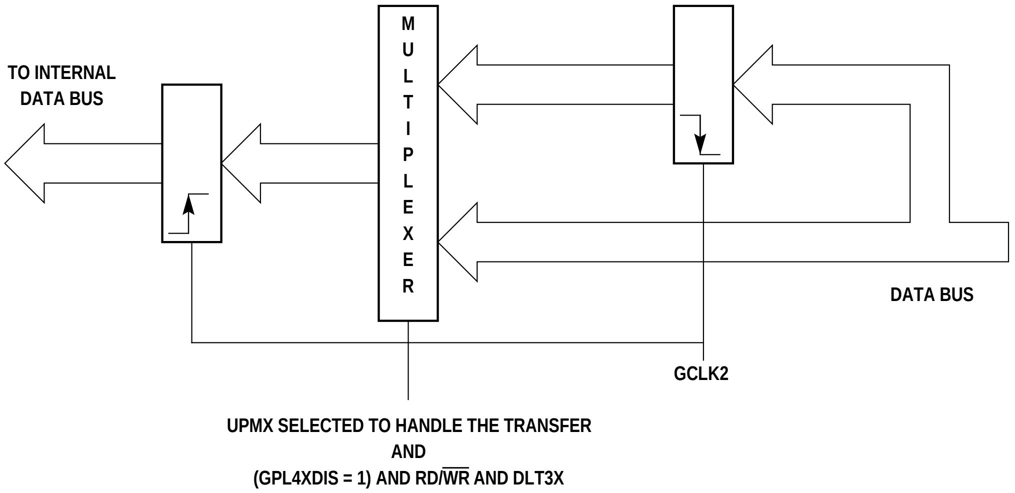

TO INTERNAL
DATA BUS
MULTIPLEXER
DATA BUS
GCLK2
UPMX SELECTED TO HANDLE THE TRANSFER
AND
(GPL4XDIS = 1) AND RD/$\overline{WR}$ AND DLT3X

**Figure 15-30. UPM Data Handling In Read Accesses**

### 15.4.0.15 WAIT MECHANISM.

**15.4.0.15.1 Internal and External Synchronous Master.** Figure 15-31 illustrates how the WAEN bit in the word read by the UPM and the UPWAIT signal are used to hold the UPM machine in a particular state until the negation of the UPWAIT signal.

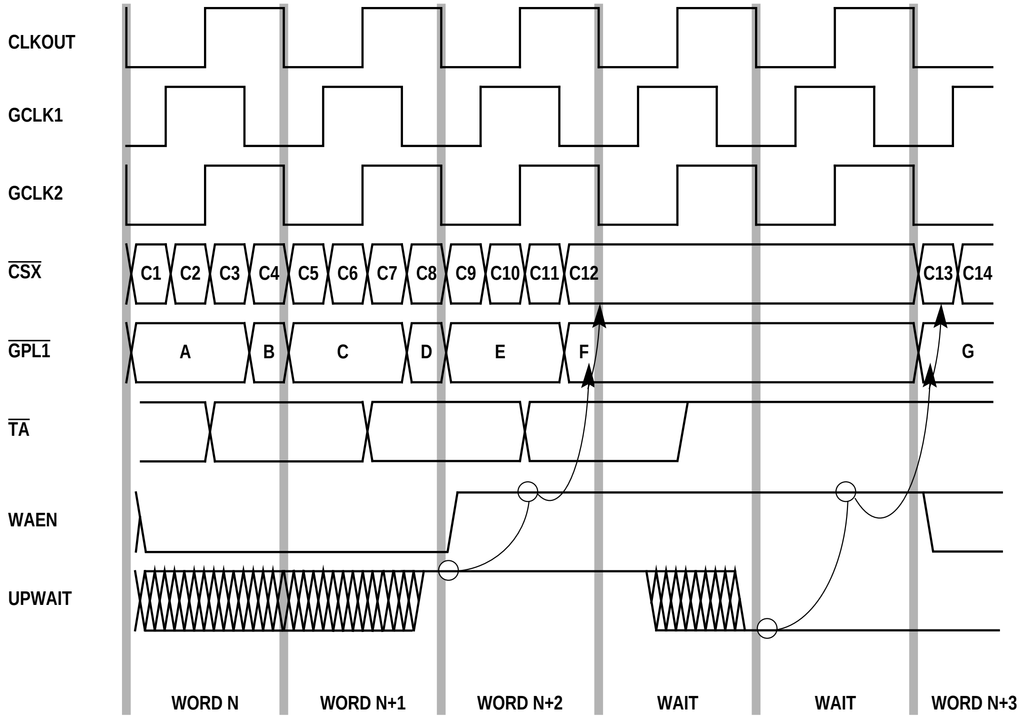


**Figure 15-31. UPM Wait Mechanism Timing For Internal and External Synchronous Master**

The UPWAIT signal is sampled at the falling edge of the CLKOUT. If the signal is asserted and the WAEN bit in the current UPM is enabled word, the UPM is frozen until the UPWAIT signal is negated. The value of the external pins driven by the UPM remains as indicated in the previous word read by the UPM. When the UPWAIT signal is negated, the UPM continues with its normal functions. Notice that during the WAIT cycles, the $\overline{TA}$ signal is negated by the UPM.

**15.4.0.15.2 External Asynchronous Master.** When the UPM machine is activated to support an asynchronous external master, the wait mechanism works such that the $\overline{AS}$ signal behaves as the external wait signal. The UPM enters a WAIT state if, after synchronizing it, the $\overline{AS}$ line is detected asserted and the WAEN bit in the current UPM word is enabled. In an analogous way to the behavior explained above, the value of the external pins driven by the UPM remains as indicated in the previous word read by the UPM.

To exit the WAIT state the $\overline{\text{AS}}$ signal should be negated. The $\overline{\text{AS}}$ negation causes all external signals controlled by the UPM to be driven high a circuit delay after $\overline{\text{AS}}$ negation. The external signals are driven in this state until the LAST bit is found in a UPM word. Refer to Figure 15-32 for details. Notice that the $\overline{\text{TA}}$ signal driven by the UPM remains in its previous value until the $\overline{\text{AS}}$ signal is negated. The TODT bit is relevant in the words read by the UPM after the negation of the $\overline{\text{AS}}$. Refer to **Section 15.5 Memory Controller External Master Support** for more information.

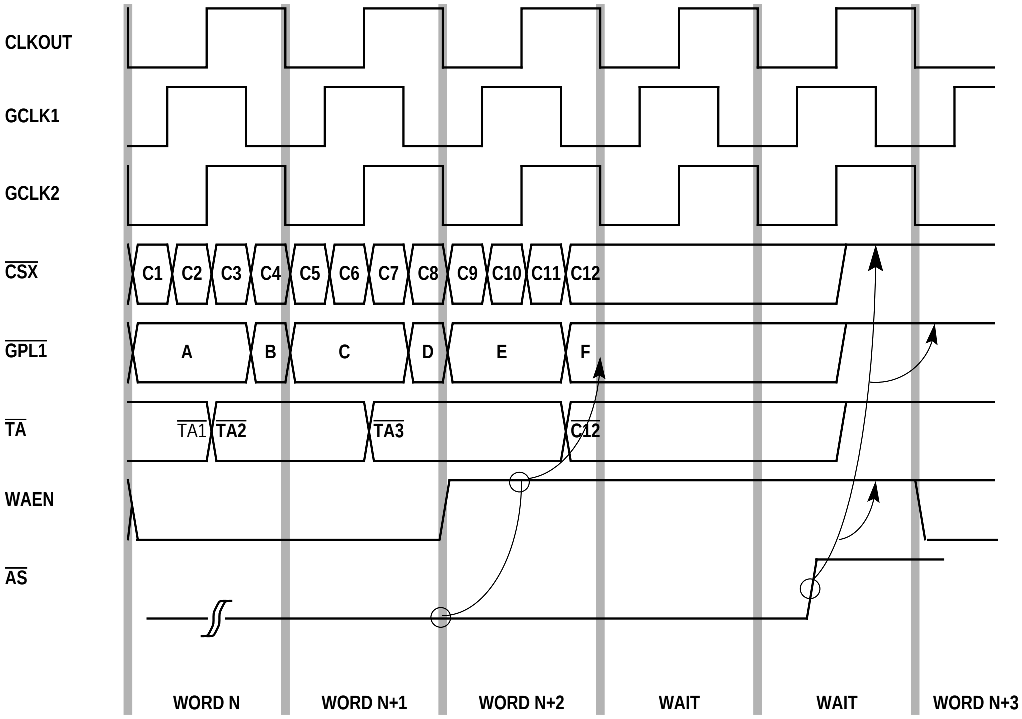


**Figure 15-32. UPM Wait Mechanism Timing For External Asynchronous Master**

**15.4.0.16 LAST BIT.** When the LAST bit is read in a word of the UPM RAM array, the highest priority pending request (if any) is serviced without any “gap cycle” in the external memory transactions dependent on the disable timer values.

**15.4.0.17 UPM START ADDRESSES LOCATION.** Table 15-7 provides the starting addresses of the UPM RAM words for each transaction type.

**Table 15-7. UPM Start Address Locations**

| REQUEST SERVICED | UPM START ADDRESS |
|---|---|
| Read Single Beat Cycle | 0x'00 |
| Read Burst Cycle | 0x'08 |
| Write Single Beat Cycle | 0x'18 |
| Write Burst Cycle | 0x'20 |
| Periodic Timer Expired | 0x'30 |
| Exception | 0x'3C |

**15.4.0.18 EXAMPLE DRAM INTERFACE.** Connecting the MPC821 to a DRAM device requires a detailed examination of the timing diagrams representing the possible memory cycles that the MPC821 must perform when accessing this device.

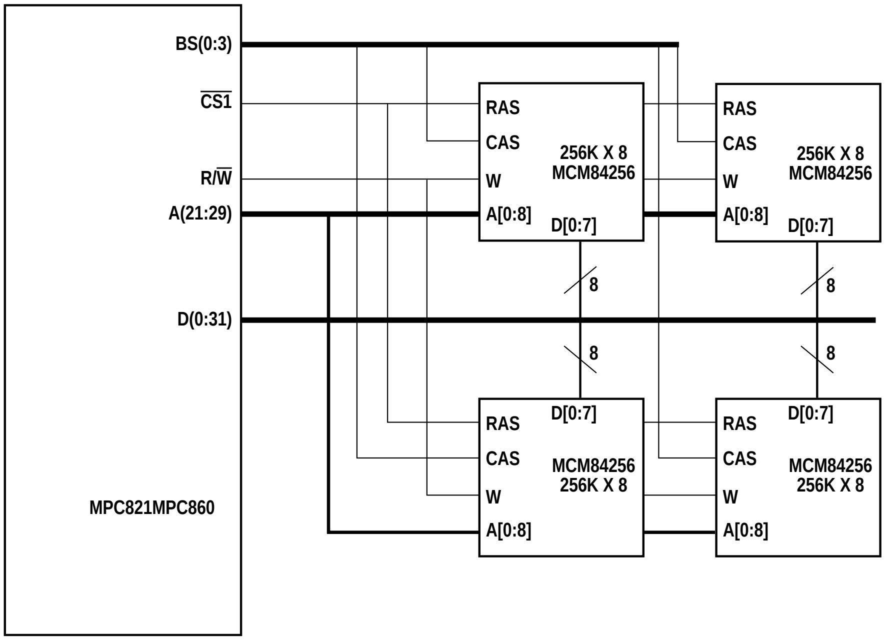


**Figure 15-33. MPC821–DRAM Interface Connection**

After the timings are created, the programming process of the UPM moves into translating these timings into tables representing the RAM array contents for each possible cycle. When the table are completed, the global parameters of the UPM must be defined for handling the disable timer (precharge) and the periodic timer (refresh) relative to Figure 15-33. The following table shows the different fields contents.

### Table 15-8. UPM RAM Word Bit Field Example

| FIELD | VALUE |
|---|---|
| MS | 10 |
| PS | 00 |
| WP | 0 |
| PTA | x'0C |
| PTAE | 1 |
| AMA | 001 |
| DSA | 01 |
| WA/$\overline{\text{GPLA}}$ | 0 |
| SAM | 1 |
| $\overline{\text{BI}}$ | 0 |

Now the RAM array of the UPM can be written through use of the MCR. The OR and BR of the specific bank must be initialized according to the address mapping of the DRAM device being used. The MS field should indicate the specific UPM selected to handle the cycle. Figure 15-34 illustrates the first locations addressed by the UPM, according to the different services required by DRAM.

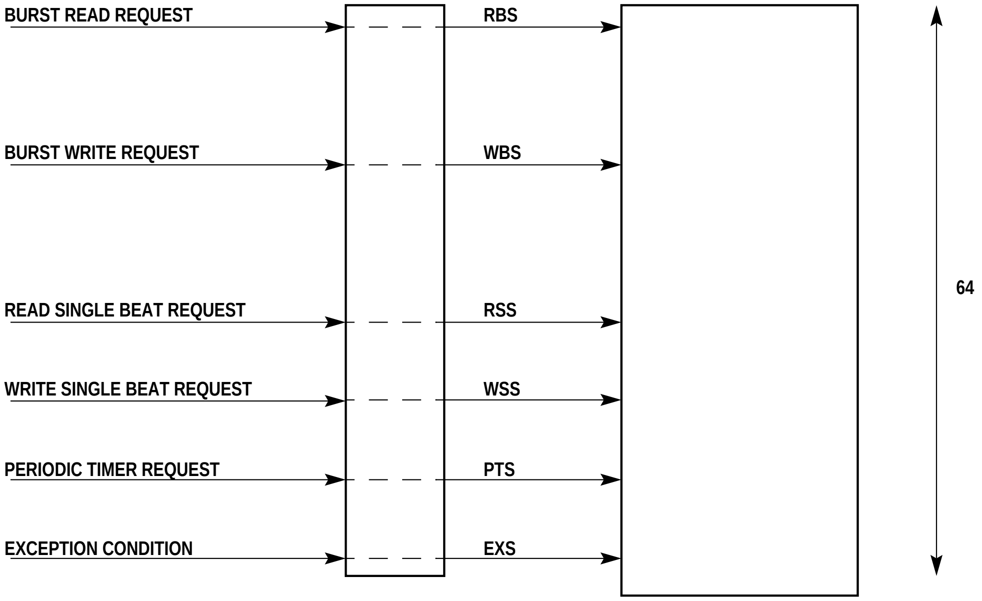
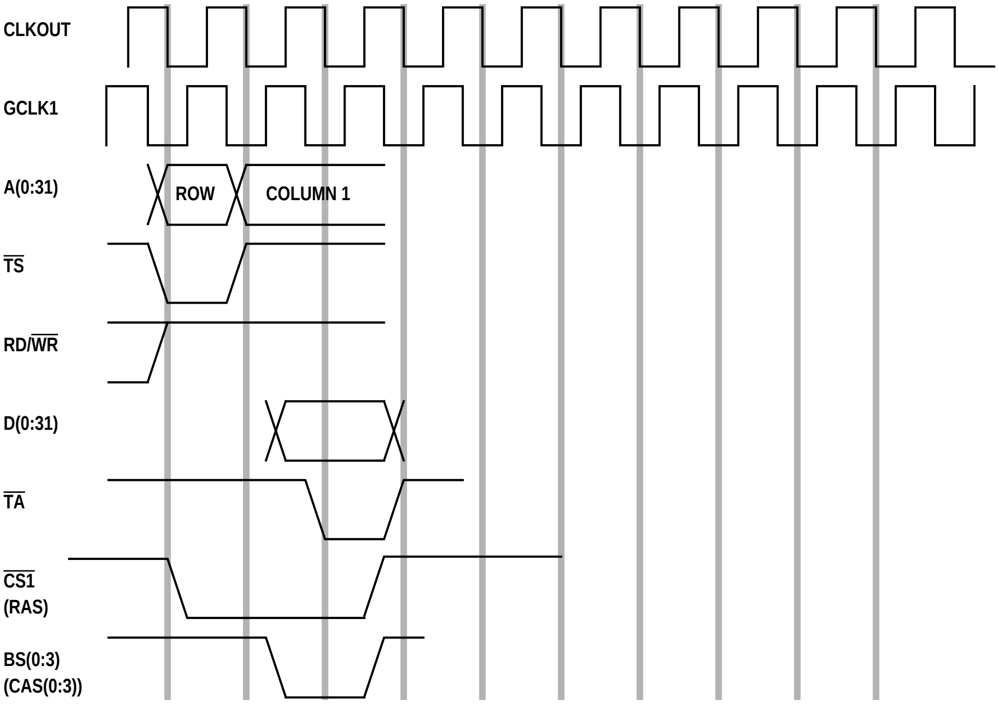


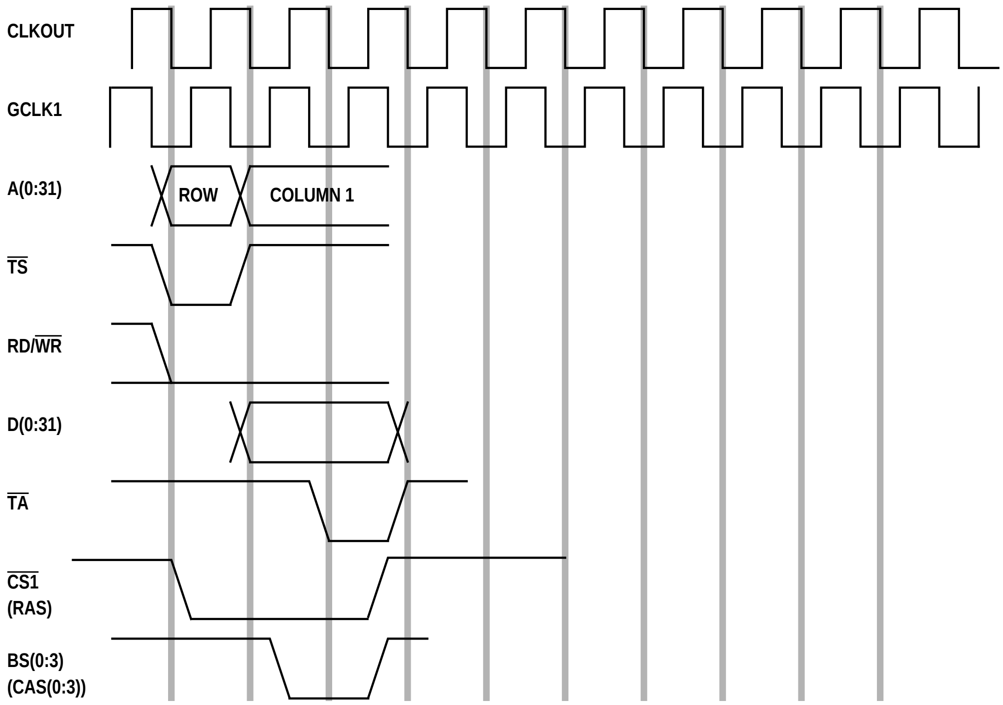
**Figure 15-34. Address Start Pointers of the UPM RAM Array**

In Figure 15-35 through Figure 15-41, the table portion at the bottom represents the RAM array contents to handle each of the possible cycles. Each column represents a different word in the RAM array. Notice that the SAM bit in the OR determines address multiplexing for the first clock cycle and subsequent cycles are controlled by the UPM RAM words. Also notice that the AMX bits in the UPM RAM word control the address multiplexing for the following clock cycles rather than the current cycle.



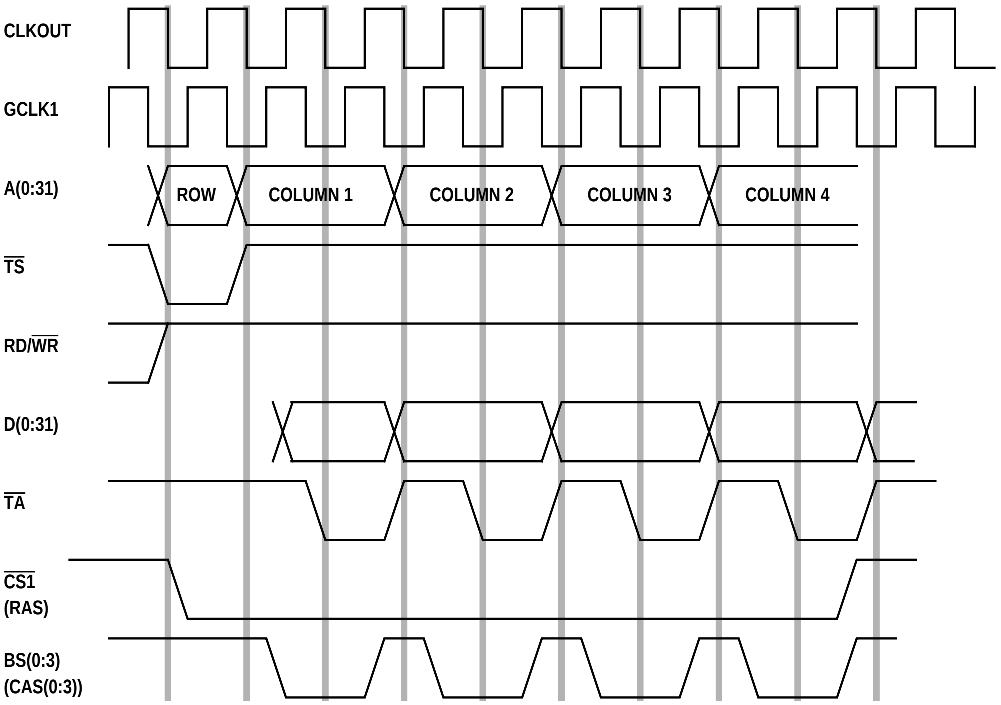
| | | | | | |
|---|---|---|---|---|---|
| cst4 | 0 | 0 | 0 | | Bit 0 |
| cst1 | 0 | 0 | 0 | | Bit 1 |
| cst2 | 0 | 0 | 1 | | Bit 2 |
| cst3 | 0 | 0 | 1 | | Bit 3 |
| bst4 | 1 | 1 | 0 | | Bit 4 |
| bst1 | 1 | 0 | 0 | | Bit 5 |
| bst2 | 1 | 0 | 1 | | Bit 6 |
| bst3 | 1 | 0 | 1 | | Bit 7 |
| g0l0 | | | | | Bit 8 |
| g0l1 | | | | | Bit 9 |
| g0h0 | | | | | Bit 10 |
| g0h1 | | | | | Bit 11 |
| g1t4 | | | | | Bit 12 |
| g1t3 | | | | | Bit 13 |
| g2t4 | | | | | Bit 14 |
| g2t3 | | | | | Bit 15 |
| g3t4 | | | | | Bit 16 |
| g3t3 | | | | | Bit 17 |
| g4t4 | | | | | Bit 18 |
| g4t3 | | | | | Bit 19 |
| g5t4 | | | | | Bit 20 |
| g5t3 | | | | | Bit 21 |
| - | | | | | Bit 22 |
| - | | | | | Bit 23 |
| loop | 0 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 0 | 0 | | Bit 28 |
| uta | 1 | 0 | 1 | | Bit 29 |
| todt | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 1 | | Bit 31 |
| | RSS | RSS+1 | RSS+2 | | |

**Figure 15-35. Single Beat Read Access To Page Mode DRAM**



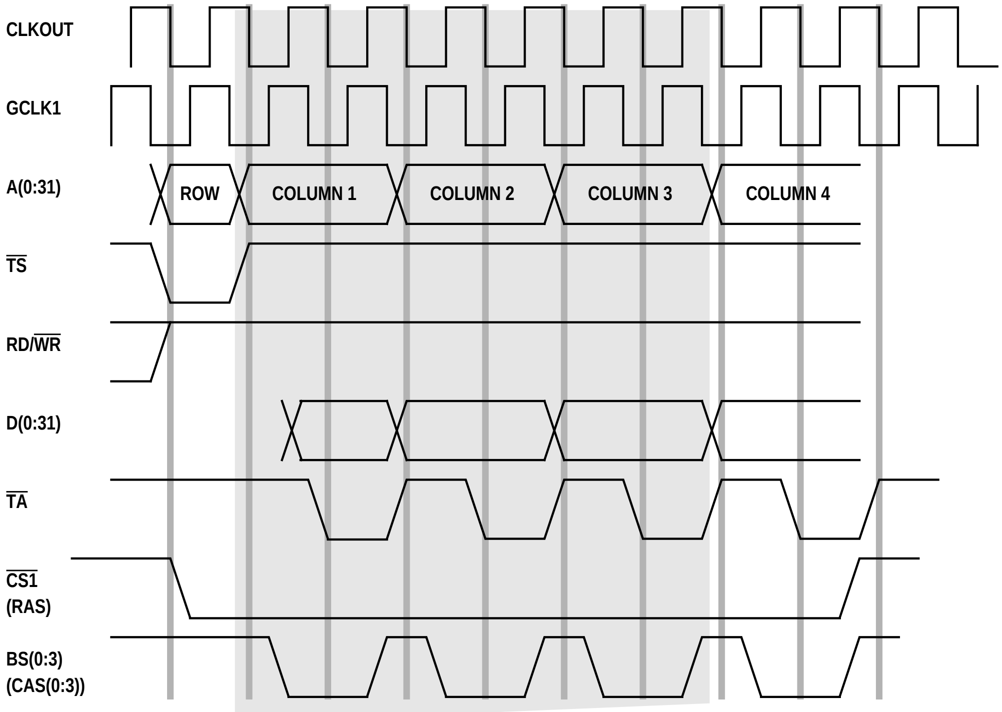
| | | | | | |
|---|---|---|---|---|---|
| cst4 | 0 | 0 | 0 | | Bit 0 |
| cst1 | 0 | 0 | 0 | | Bit 1 |
| cst2 | 0 | 0 | 1 | | Bit 2 |
| cst3 | 0 | 0 | 1 | | Bit 3 |
| bst4 | 1 | 1 | 0 | | Bit 4 |
| bst1 | 1 | 0 | 0 | | Bit 5 |
| bst2 | 1 | 0 | 1 | | Bit 6 |
| bst3 | 1 | 0 | 1 | | Bit 7 |
| g0l0 | | | | | Bit 8 |
| g0l1 | | | | | Bit 9 |
| g0h0 | | | | | Bit 10 |
| g0h1 | | | | | Bit 11 |
| g1t4 | | | | | Bit 12 |
| g1t3 | | | | | Bit 13 |
| g2t4 | | | | | Bit 14 |
| g2t3 | | | | | Bit 15 |
| g3t4 | | | | | Bit 16 |
| g3t3 | | | | | Bit 17 |
| g4t4 | | | | | Bit 18 |
| g4t3 | | | | | Bit 19 |
| g5t4 | | | | | Bit 20 |
| g5t3 | | | | | Bit 21 |
| - | | | | | Bit 22 |
| - | | | | | Bit 23 |
| loop | 0 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 0 | 0 | | Bit 28 |
| uta | 1 | 0 | 1 | | Bit 29 |
| todt | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 1 | | Bit 31 |
| | WSS | WSS+1 | WSS+2 | | |

**Figure 15-36. Single Beat Write Access To Page Mode DRAM**



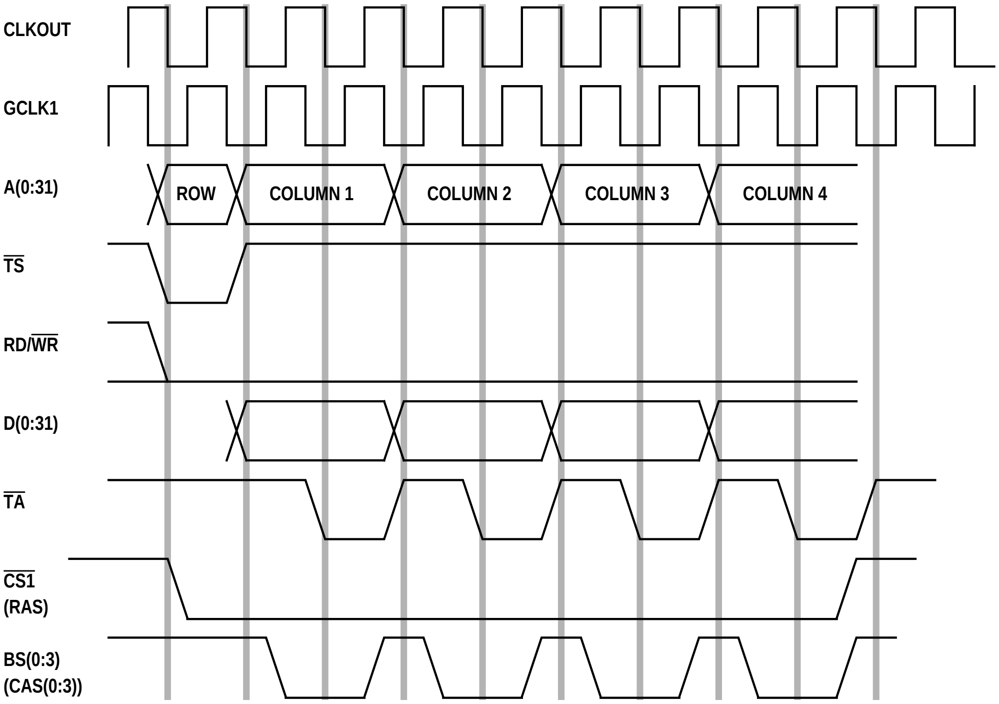
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| cst4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 0 |
| cst1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 1 |
| cst2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 2 |
| cst3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 3 |
| bst4 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | Bit 4 |
| bst1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 5 |
| bst2 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 6 |
| bst3 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 7 |
| g0l0 | | | | | | | | | | | Bit 8 |
| g0l1 | | | | | | | | | | | Bit 9 |
| g0h0 | | | | | | | | | | | Bit 10 |
| g0h1 | | | | | | | | | | | Bit 11 |
| g1t4 | | | | | | | | | | | Bit 12 |
| g1t3 | | | | | | | | | | | Bit 13 |
| g2t4 | | | | | | | | | | | Bit 14 |
| g2t3 | | | | | | | | | | | Bit 15 |
| g3t4 | | | | | | | | | | | Bit 16 |
| g3t3 | | | | | | | | | | | Bit 17 |
| g4t4 | | | | | | | | | | | Bit 18 |
| g4t3 | | | | | | | | | | | Bit 19 |
| g5t4 | | | | | | | | | | | Bit 20 |
| g5t3 | | | | | | | | | | | Bit 21 |
| - | | | | | | | | | | | Bit 22 |
| - | | | | | | | | | | | Bit 23 |
| loop | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | | Bit 28 |
| uta | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 29 |
| todt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 31 |
| | RBS | RBS+1 | RBS+2 | RBS+3 | RBS+4 | RBS+5 | RBS+6 | RBS+7 | RBS+8 | | |

**Figure 15-37. Burst Read Access To Page Mode DRAM (No LOOP)**



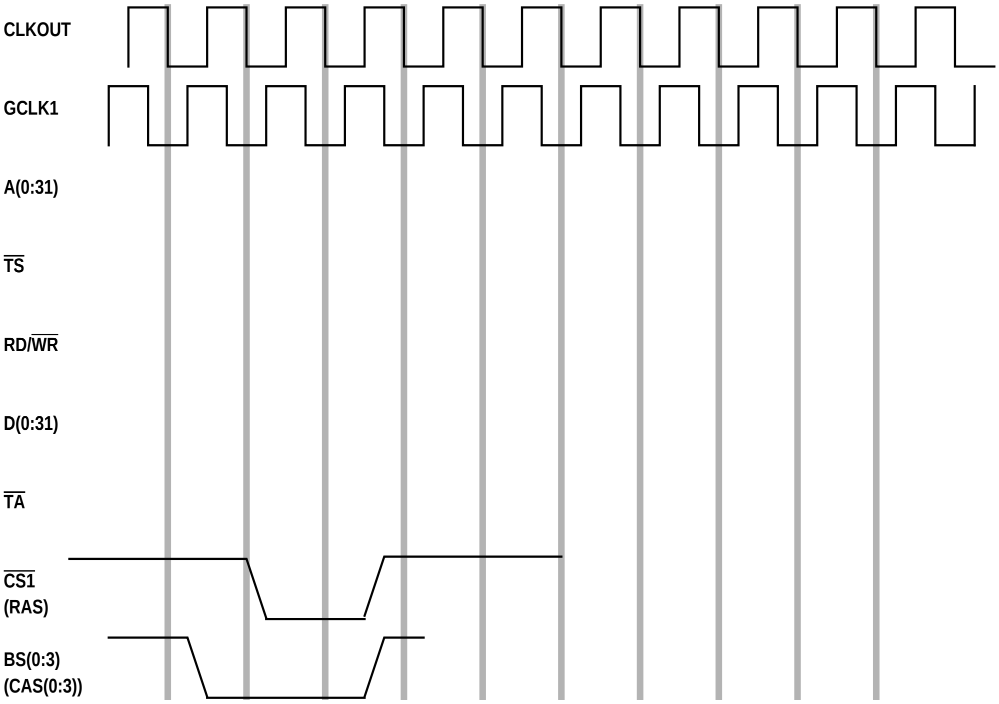
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| cst4 | 0 | 0 | 0 | 0 | 0 | | Bit 0 |
| cst1 | 0 | 0 | 0 | 0 | 0 | | Bit 1 |
| cst2 | 0 | 0 | 0 | 0 | 1 | | Bit 2 |
| cst3 | 0 | 0 | 0 | 0 | 1 | | Bit 3 |
| bst4 | 1 | 1 | 0 | 1 | 0 | | Bit 4 |
| bst1 | 1 | 0 | 0 | 0 | 0 | | Bit 5 |
| bst2 | 1 | 0 | 1 | 0 | 1 | | Bit 6 |
| bst3 | 1 | 0 | 1 | 0 | 1 | | Bit 7 |
| g0l0 | | | | | | | Bit 8 |
| g0l1 | | | | | | | Bit 9 |
| g0h0 | | | | | | | Bit 10 |
| g0h1 | | | | | | | Bit 11 |
| g1t4 | | | | | | | Bit 12 |
| g1t3 | | | | | | | Bit 13 |
| g2t4 | | | | | | | Bit 14 |
| g2t3 | | | | | | | Bit 15 |
| g3t4 | | | | | | | Bit 16 |
| g3t3 | | | | | | | Bit 17 |
| g4t4 | | | | | | | Bit 18 |
| g4t3 | | | | | | | Bit 19 |
| g5t4 | | | | | | | Bit 20 |
| g5t3 | | | | | | | Bit 21 |
| - | | | | | | | Bit 22 |
| - | | | | | | | Bit 23 |
| loop | 0 | 1 | 1 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 1 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 0 | 1 | 0 | 0 | | Bit 28 |
| uta | 1 | 0 | 1 | 0 | 1 | | Bit 29 |
| todt | 0 | 0 | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 0 | 0 | 1 | | Bit 31 |
| | RBS | RBS+1 | RBS+2 | RBS+3 | RBS+4 | | |

**Figure 15-38. Burst Read Access To Page Mode DRAM (LOOP)**



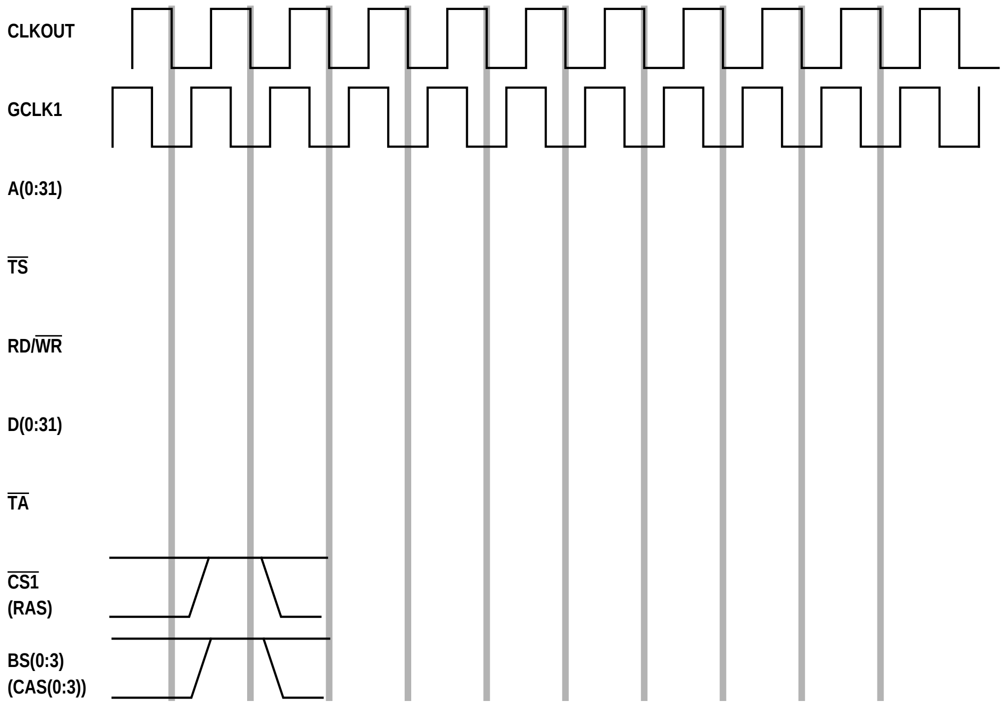
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| cst4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 0 |
| cst1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 1 |
| cst2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 2 |
| cst3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 3 |
| bst4 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | Bit 4 |
| bst1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 5 |
| bst2 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 6 |
| bst3 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 7 |
| g0l0 | | | | | | | | | | | Bit 8 |
| g0l1 | | | | | | | | | | | Bit 9 |
| g0h0 | | | | | | | | | | | Bit 10 |
| g0h1 | | | | | | | | | | | Bit 11 |
| g1t4 | | | | | | | | | | | Bit 12 |
| g1t3 | | | | | | | | | | | Bit 13 |
| g2t4 | | | | | | | | | | | Bit 14 |
| g2t3 | | | | | | | | | | | Bit 15 |
| g3t4 | | | | | | | | | | | Bit 16 |
| g3t3 | | | | | | | | | | | Bit 17 |
| g4t4 | | | | | | | | | | | Bit 18 |
| g4t3 | | | | | | | | | | | Bit 19 |
| g5t4 | | | | | | | | | | | Bit 20 |
| g5t3 | | | | | | | | | | | Bit 21 |
| - | | | | | | | | | | | Bit 22 |
| - | | | | | | | | | | | Bit 23 |
| loop | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | | Bit 28 |
| uta | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 29 |
| todt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 31 |
| | **WBS** | **WBS+1** | **WBS+2** | **WBS+3** | **WBS+4** | **WBS+5** | **WBS+6** | **WBS+7** | **WBS+8** | | |

**Figure 15-39. Burst Write Access To Page Mode DRAM (No LOOP)**



| | | | | | |
|---|---|---|---|---|---|
| cst4 | 1 | 0 | 0 | | Bit 0 |
| cst1 | 1 | 0 | 0 | | Bit 1 |
| cst2 | 1 | 0 | 1 | | Bit 2 |
| cst3 | 1 | 0 | 1 | | Bit 3 |
| bst4 | 1 | 0 | 0 | | Bit 4 |
| bst1 | 0 | 0 | 0 | | Bit 5 |
| bst2 | 0 | 0 | 1 | | Bit 6 |
| bst3 | 0 | 0 | 1 | | Bit 7 |
| g0l0 | | | | | Bit 8 |
| g0l1 | | | | | Bit 9 |
| g0h0 | | | | | Bit 10 |
| g0h1 | | | | | Bit 11 |
| g1t4 | | | | | Bit 12 |
| g1t3 | | | | | Bit 13 |
| g2t4 | | | | | Bit 14 |
| g2t3 | | | | | Bit 15 |
| g3t4 | | | | | Bit 16 |
| g3t3 | | | | | Bit 17 |
| g4t4 | | | | | Bit 18 |
| g4t3 | | | | | Bit 19 |
| g5t4 | | | | | Bit 20 |
| g5t3 | | | | | Bit 21 |
| - | | | | | Bit 22 |
| - | | | | | Bit 23 |
| loop | 0 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 0 | 0 | | Bit 28 |
| uta | 0 | 0 | 0 | | Bit 29 |
| todt | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 1 | | Bit 31 |
| | PTS | PTS+1 | PTS+2 | | |

**Figure 15-40. Refresh Cycle (CBR) To Page Mode DRAM**



| | | | |
|---|---|---|---|
| cst4 | 1 | | Bit 0 |
| cst1 | 1 | | Bit 1 |
| cst2 | 1 | | Bit 2 |
| cst3 | 1 | | Bit 3 |
| bst4 | 1 | | Bit 4 |
| bst1 | 1 | | Bit 5 |
| bst2 | 1 | | Bit 6 |
| bst3 | 1 | | Bit 7 |
| g0l0 | | | Bit 8 |
| g0l1 | | | Bit 9 |
| g0h0 | | | Bit 10 |
| g0h1 | | | Bit 11 |
| g1t4 | | | Bit 12 |
| g1t3 | | | Bit 13 |
| g2t4 | | | Bit 14 |
| g2t3 | | | Bit 15 |
| g3t4 | | | Bit 16 |
| g3t3 | | | Bit 17 |
| g4t4 | | | Bit 18 |
| g4t3 | | | Bit 19 |
| g5t4 | | | Bit 20 |
| g5t3 | | | Bit 21 |
| - | | | Bit 22 |
| - | | | Bit 23 |
| loop | 0 | | Bit 24 |
| exen | 0 | | Bit 25 |
| amx0 | 0 | | Bit 26 |
| amx1 | 0 | | Bit 27 |
| na | 0 | | Bit 28 |
| uta | 0 | | Bit 29 |
| todt | 1 | | Bit 30 |
| last | 1 | | Bit 31 |
| | EXS | | |

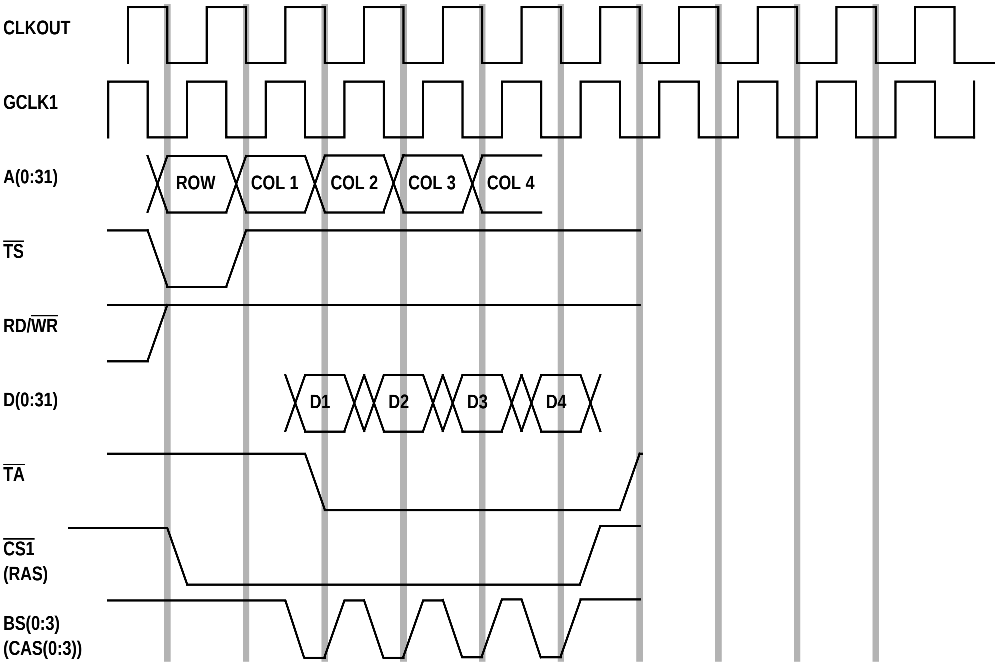
**Figure 15-41. Exception Cycle**

If the $\overline{\text{GPL\_A4}}$ line is not used as an output line, the performance for a page read access can be increased significantly if the GPL_x4DIS is defined as ”1”. In this case the data bus is sampled at the falling edge of GCLK1 if instructed by the UPM word. The following example shows how the burst read access to page mode DRAM (no LOOP) can be modified using this feature. In this case the configuration registers are defined in the following way.

**Table 15-9. UPM RAM Word Bit Field Example**

| FIELD | VALUE |
|---|---|
| MS | 10 |
| PS | 00 |
| WP | 0 |
| PTA | x’0C |
| PTAE | 1 |
| AMA | 001 |
| DSA | 01 |
| GPL_A4DIS | 1 |
| SAM | 1 |
| $\overline{\text{BI}}$ | 0 |

The timing diagram in Figure 15-42 illustrates how the nine cycles of the burst read access shown in Figure 15-37 can be reduced to 6 clock cycles (for 32-bit port size memory). When a 16-bit port size memory is connected, the reduction is from 17 to 10 cycles and when an 8-bit port size memory is connected, the reduction is from 33 to 18 cycles.



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| cst4 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 0 |
| cst1 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 1 |
| cst2 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 2 |
| cst3 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 3 |
| bst4 | 1 | 1 | 1 | 1 | 1 | 1 | | Bit 4 |
| bst1 | 1 | 0 | 0 | 0 | 0 | 1 | | Bit 5 |
| bst2 | 1 | 0 | 0 | 0 | 0 | 1 | | Bit 6 |
| bst3 | 1 | 0 | 0 | 0 | 0 | 1 | | Bit 7 |
| g0l0 | | | | | | | | Bit 8 |
| g0l1 | | | | | | | | Bit 9 |
| g0h0 | | | | | | | | Bit 10 |
| g0h1 | | | | | | | | Bit 11 |
| g1t4 | | | | | | | | Bit 12 |
| g1t3 | | | | | | | | Bit 13 |
| g2t4 | | | | | | | | Bit 14 |
| g2t3 | | | | | | | | Bit 15 |
| g3t4 | | | | | | | | Bit 16 |
| g3t3 | | | | | | | | Bit 17 |
| g4t4 -> DLT3 | 1 | 1 | 1 | 1 | 1 | 1 | | Bit 18 |
| g4t3 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 19 |
| g5t4 | | | | | | | | Bit 20 |
| g5t3 | | | | | | | | Bit 21 |
| - | | | | | | | | Bit 22 |
| - | | | | | | | | Bit 23 |
| loop | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 0 | 0 | 1 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 1 | 1 | 1 | 0 | 0 | | Bit 28 |
| uta | 1 | 0 | 1 | 0 | 0 | 1 | | Bit 29 |
| todt | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 31 |
| | RBS | RBS+1 | RBS+2 | RBS+3 | RBS+4 | RBS+5 | | |

**Figure 15-42. Page Mode DRAM Burst Read Access (Data Sampling on Falling Edge of CLKOUT)**

**15.4.3.14 EDO INTERFACE EXAMPLE.** Figure 15-43 illustrates a memory connection to extended data-out type devices. For this connection, $\overline{\text{GPL1}}$ is connected to the memory device $\overline{\text{OE}}$ pins.

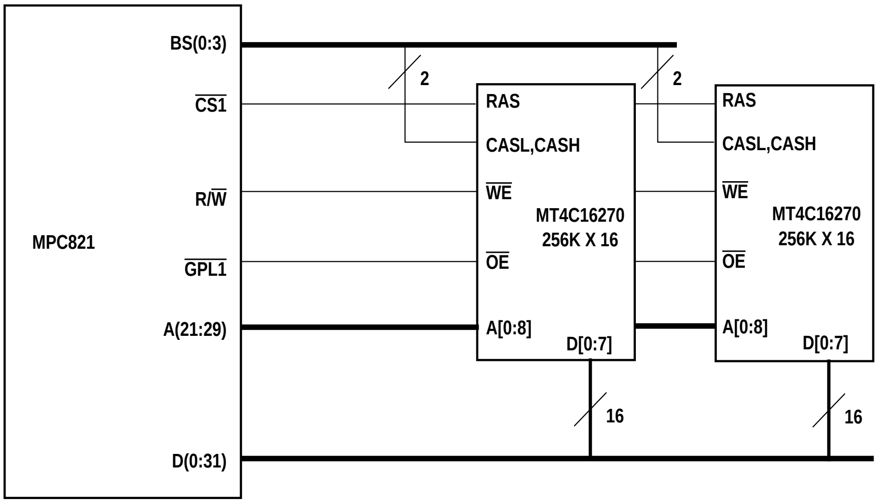
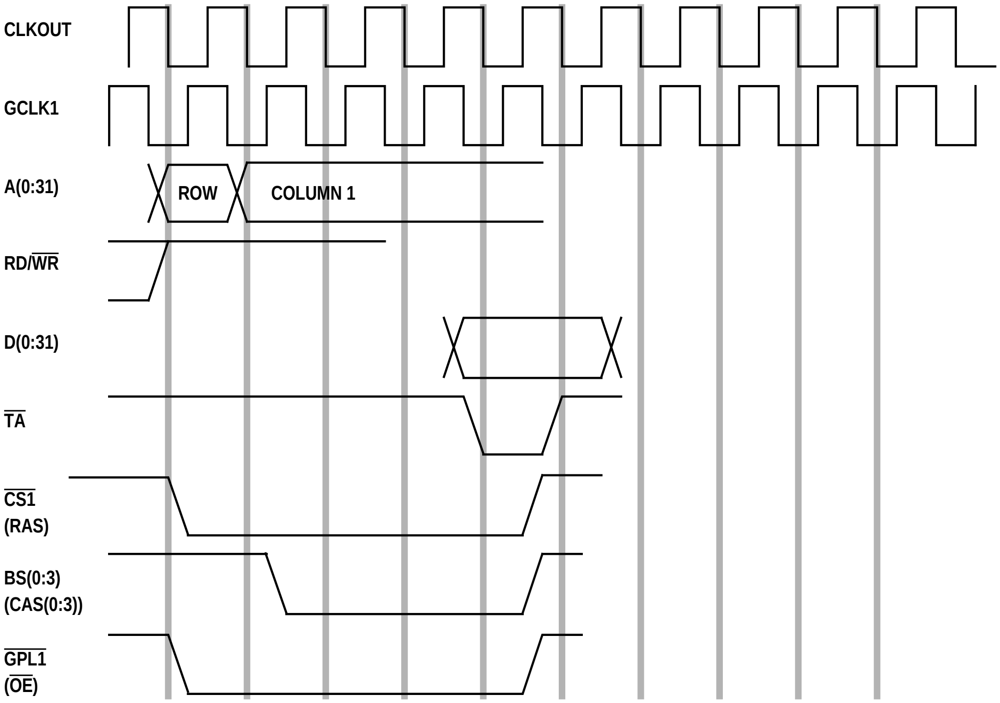


**Figure 15-43. EDO Interface Connection**

Table 15-10 shows the programming of the register field for supporting the configuration shown in Figure 15-43. The assumption is that the BRGCLK frequency is 25 MHz and that the device needs a 512-cycle refresh every 8 milliseconds. The example assumes a CLKOUT frequency of 50 MHz.

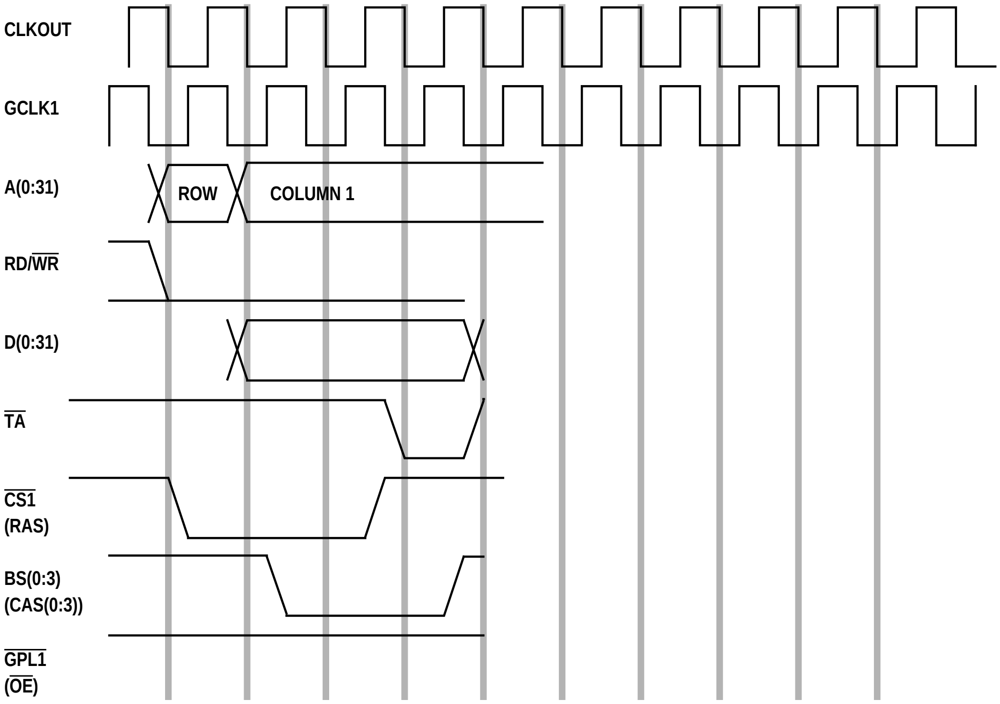
**Table 15-10. EDO Connection Field Value Example**

| FIELD | VALUE |
|---|---|
| MS | 10 |
| PS | 00 |
| WP | 0 |
| PTP | x'02 |
| PTA | x'0C |
| PTAE | 1 |
| AMA | 001 |
| DSA | 10 |
| SAM | 1 |
| $\overline{\text{BI}}$ | 0 |



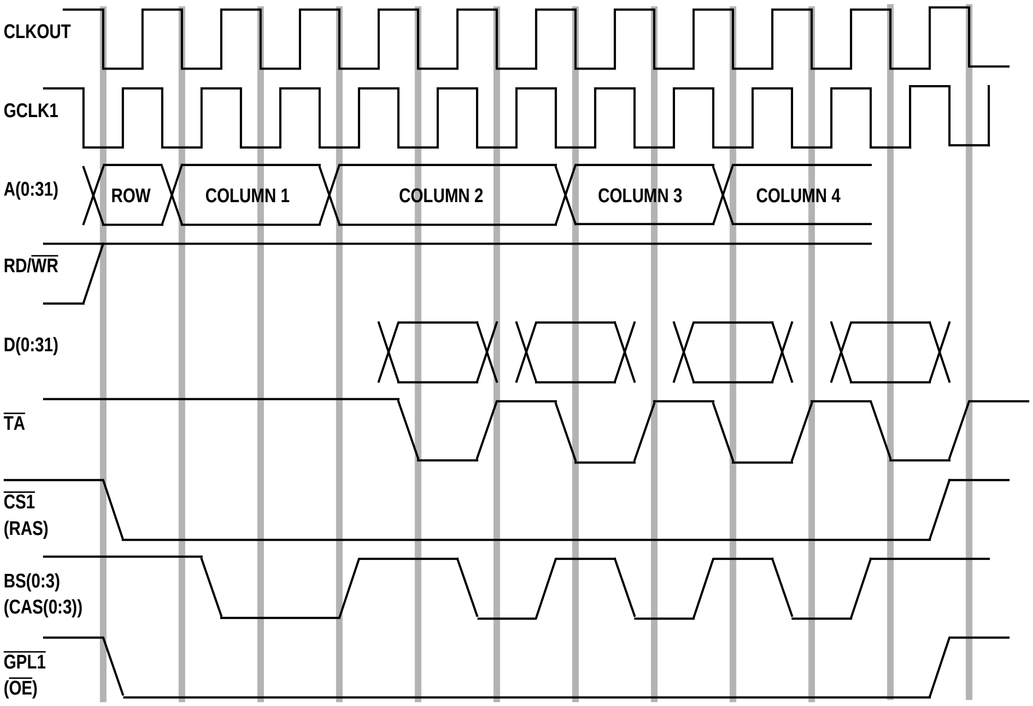
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| cst4 | 0 | 0 | 0 | 0 | 0 | | Bit 0 |
| cst1 | 0 | 0 | 0 | 0 | 0 | | Bit 1 |
| cst2 | 0 | 0 | 0 | 0 | 1 | | Bit 2 |
| cst3 | 0 | 0 | 0 | 0 | 1 | | Bit 3 |
| bst4 | 1 | 1 | 0 | 0 | 0 | | Bit 4 |
| bst1 | 1 | 0 | 0 | 0 | 0 | | Bit 5 |
| bst2 | 1 | 0 | 0 | 0 | 1 | | Bit 6 |
| bst3 | 1 | 0 | 0 | 0 | 1 | | Bit 7 |
| g0l0 | | | | | | | Bit 8 |
| g0l1 | | | | | | | Bit 9 |
| g0h0 | | | | | | | Bit 10 |
| g0h1 | | | | | | | Bit 11 |
| g1t4 | 0 | 0 | 0 | 0 | 0 | | Bit 12 |
| g1t3 | 0 | 0 | 0 | 0 | 1 | | Bit 13 |
| g2t4 | | | | | | | Bit 14 |
| g2t3 | | | | | | | Bit 15 |
| g3t4 | | | | | | | Bit 16 |
| g3t3 | | | | | | | Bit 17 |
| g4t4 | | | | | | | Bit 18 |
| g4t3 | | | | | | | Bit 19 |
| g5t4 | | | | | | | Bit 20 |
| g5t3 | | | | | | | Bit 21 |
| - | | | | | | | Bit 22 |
| - | | | | | | | Bit 23 |
| loop | 0 | 0 | 0 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 0 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 0 | 0 | 0 | 0 | | Bit 28 |
| uta | 1 | 1 | 1 | 0 | 1 | | Bit 29 |
| todt | 0 | 0 | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 0 | 0 | 1 | | Bit 31 |
| | RSS | RSS+1 | RSS+2 | RSS+3 | RSS+4 | | |

**Figure 15-44. Single Beat Read Access To Page Mode DRAM With Extended Data-Out**



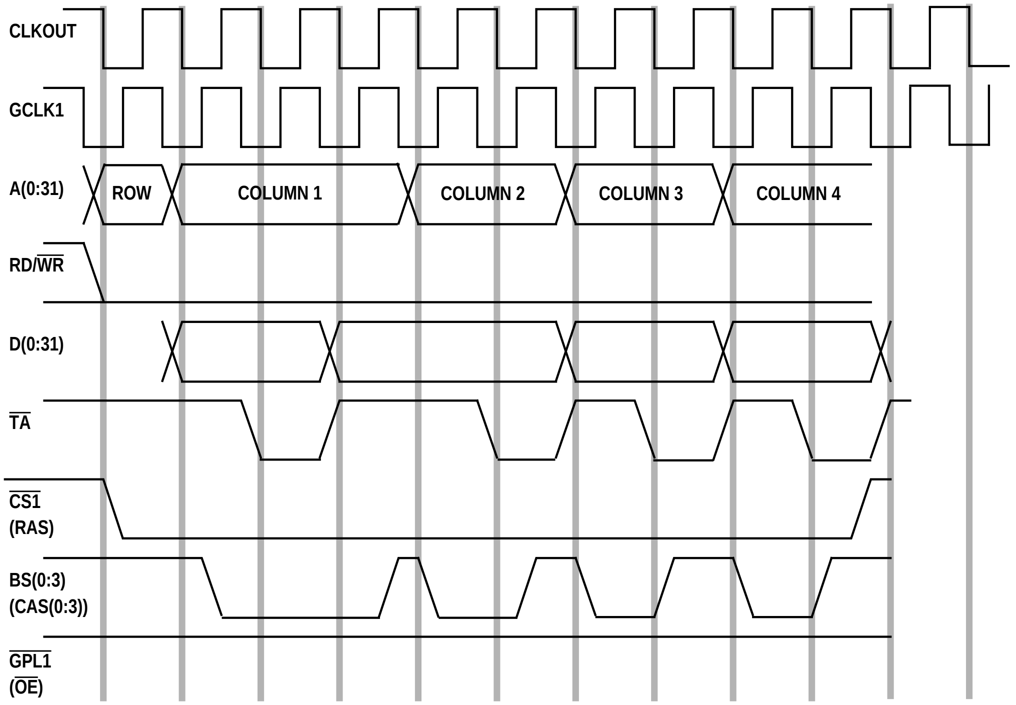
| | | | | | | |
|---|---|---|---|---|---|---|
| cst4<br>cst1<br>cst2<br>cst3 | 0<br>0<br>0<br>0 | 0<br>0<br>0<br>0 | 0<br>0<br>1<br>1 | 1<br>1<br>1<br>1 | | Bit 0<br>Bit 1<br>Bit 2<br>Bit 3 |
| bst4<br>bst1<br>bst2<br>bst3 | 1<br>1<br>1<br>1 | 1<br>0<br>0<br>0 | 0<br>0<br>0<br>0 | 0<br>0<br>0<br>1 | | Bit 4<br>Bit 5<br>Bit 6<br>Bit 7 |
| g0l0<br>g0l1<br>g0h0<br>g0h1 | | | | | | Bit 8<br>Bit 9<br>Bit 10<br>Bit 11 |
| g1t4<br>g1t3 | 1<br>1 | 1<br>1 | 1<br>1 | 1<br>1 | | Bit 12<br>Bit 13 |
| g2t4<br>g2t3 | | | | | | Bit 14<br>Bit 15 |
| g3t4<br>g3t3 | | | | | | Bit 16<br>Bit 17 |
| g4t4<br>g4t3 | | | | | | Bit 18<br>Bit 19 |
| g5t4<br>g5t3 | | | | | | Bit 20<br>Bit 21 |
| -<br>- | | | | | | Bit 22<br>Bit 23 |
| loop<br>exen | 0<br>0 | 0<br>0 | 0<br>0 | 0<br>0 | | Bit 24<br>Bit 25 |
| amx0<br>amx1<br>na<br>uta<br>todt<br>last | 0<br>0<br>0<br>1<br>0<br>0 | 0<br>0<br>0<br>1<br>0<br>0 | 0<br>0<br>0<br>0<br>0<br>0 | 1<br>0<br>0<br>1<br>1<br>1 | | Bit 26<br>Bit 27<br>Bit 28<br>Bit 29<br>Bit 30<br>Bit 31 |
| | WSS | WSS+1 | WSS+2 | WSS+3 | | |

**Figure 15-45. Single Beat Write Access To Page Mode DRAM With Extended Data-Out**



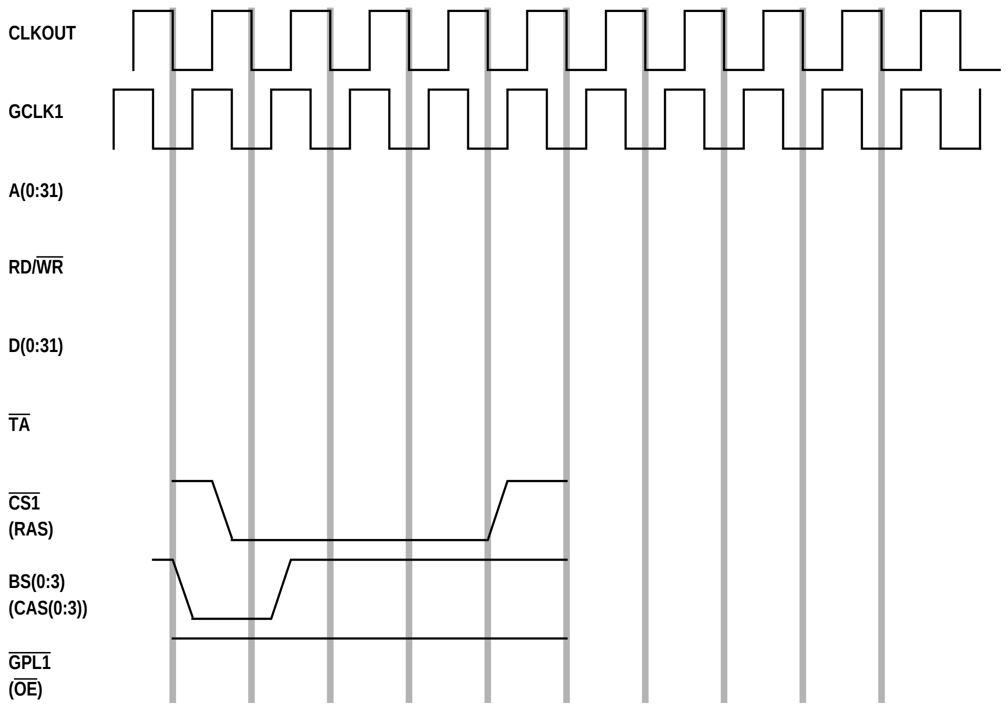
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| cst4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Bit 0 |
| cst1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Bit 1 |
| cst2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Bit 2 |
| cst3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Bit 3 |
| bst4 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | Bit 4 |
| bst1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | Bit 5 |
| bst2 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | Bit 6 |
| bst3 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | Bit 7 |
| g0l0 | | | | | | | | | | | | Bit 8 |
| g0l1 | | | | | | | | | | | | Bit 9 |
| g0h0 | | | | | | | | | | | | Bit 10 |
| g0h1 | | | | | | | | | | | | Bit 11 |
| g1t4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Bit 12 |
| g1t3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Bit 13 |
| g2t4 | | | | | | | | | | | | Bit 14 |
| g2t3 | | | | | | | | | | | | Bit 15 |
| g3t4 | | | | | | | | | | | | Bit 16 |
| g3t3 | | | | | | | | | | | | Bit 17 |
| g4t4 | | | | | | | | | | | | Bit 18 |
| g4t3 | | | | | | | | | | | | Bit 19 |
| g5t4 | | | | | | | | | | | | Bit 20 |
| g5t3 | | | | | | | | | | | | Bit 21 |
| - | | | | | | | | | | | | Bit 22 |
| - | | | | | | | | | | | | Bit 23 |
| loop | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Bit 24 |
| exen | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | Bit 25 |
| amx0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Bit 26 |
| amx1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Bit 27 |
| na | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | Bit 28 |
| uta | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | Bit 29 |
| todt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Bit 30 |
| last | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Bit 31 |
| | **RBS** | **RBS+1** | **RBS+2** | **RBS+3** | **RBS+4** | **RBS+5** | **RBS+6** | **RBS+7** | **RBS+8** | **RBS+9** | **RBS+10** | |

**Figure 15-46. Burst Read Access To Page Mode DRAM With Extended Data-Out**



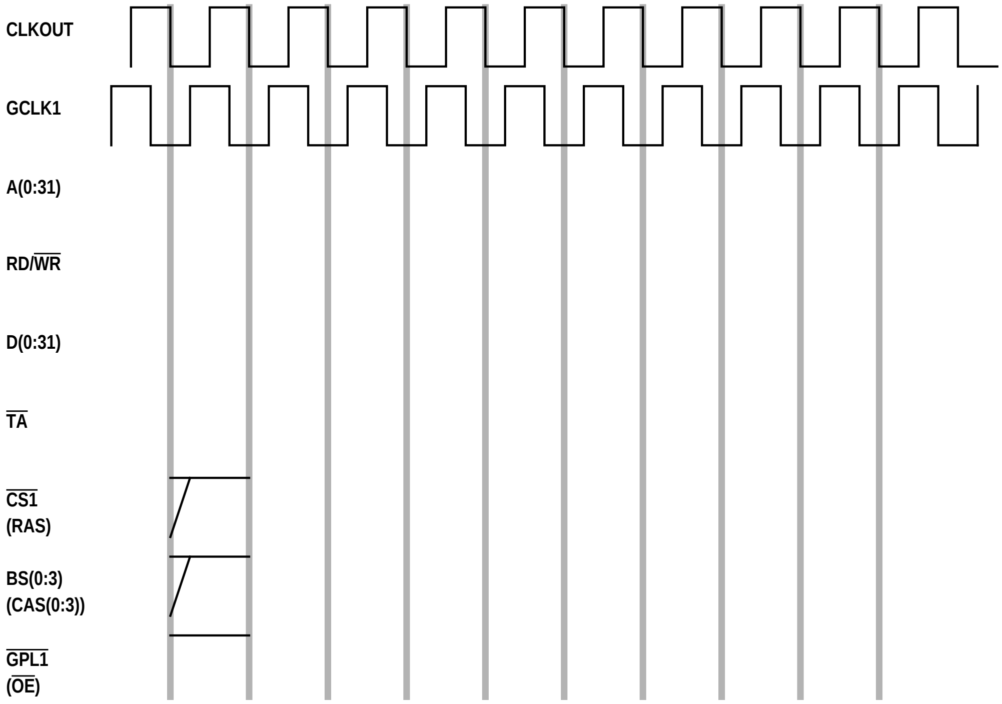
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| cst4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 0 |
| cst1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 1 |
| cst2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 2 |
| cst3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 3 |
| bst4 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | | Bit 4 |
| bst1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 5 |
| bst2 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 6 |
| bst3 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 7 |
| g0l0 | | | | | | | | | | | | Bit 8 |
| g0l1 | | | | | | | | | | | | Bit 9 |
| g0h0 | | | | | | | | | | | | Bit 10 |
| g0h1 | | | | | | | | | | | | Bit 11 |
| g1t4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | Bit 12 |
| g1t3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | Bit 13 |
| g2t4 | | | | | | | | | | | | Bit 14 |
| g2t3 | | | | | | | | | | | | Bit 15 |
| g3t4 | | | | | | | | | | | | Bit 16 |
| g3t3 | | | | | | | | | | | | Bit 17 |
| g4t4 | | | | | | | | | | | | Bit 18 |
| g4t3 | | | | | | | | | | | | Bit 19 |
| g5t4 | | | | | | | | | | | | Bit 20 |
| g5t3 | | | | | | | | | | | | Bit 21 |
| - | | | | | | | | | | | | Bit 22 |
| - | | | | | | | | | | | | Bit 23 |
| loop | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | | Bit 28 |
| uta | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | Bit 29 |
| todt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 31 |
| | **WBS** | **WBS+1** | **WBS+2** | **WBS+3** | **WBS+4** | **WBS+5** | **WBS+6** | **WBS+7** | **WBS+8** | **WBS+9** | | |

**Figure 15-47. Burst Write Access To Page Mode DRAM With Extended Data-Out**



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| cst4 | 1 | 0 | 0 | 0 | 1 | | Bit 0 |
| cst1 | 1 | 0 | 0 | 0 | 1 | | Bit 1 |
| cst2 | 0 | 0 | 0 | 0 | 1 | | Bit 2 |
| cst3 | 0 | 0 | 0 | 0 | 1 | | Bit 3 |
| bst4 | 0 | 0 | 1 | 1 | 1 | | Bit 4 |
| bst1 | 0 | 1 | 1 | 1 | 1 | | Bit 5 |
| bst2 | 0 | 1 | 1 | 1 | 1 | | Bit 6 |
| bst3 | 0 | 1 | 1 | 1 | 1 | | Bit 7 |
| g0l0 | | | | | | | Bit 8 |
| g0l1 | | | | | | | Bit 9 |
| g0h0 | | | | | | | Bit 10 |
| g0h1 | | | | | | | Bit 11 |
| g1t4 | 1 | 1 | 1 | 1 | 1 | | Bit 12 |
| g1t3 | 1 | 1 | 1 | 1 | 1 | | Bit 13 |
| g2t4 | | | | | | | Bit 14 |
| g2t3 | | | | | | | Bit 15 |
| g3t4 | | | | | | | Bit 16 |
| g3t3 | | | | | | | Bit 17 |
| g4t4 | | | | | | | Bit 18 |
| g4t3 | | | | | | | Bit 19 |
| g5t4 | | | | | | | Bit 20 |
| g5t3 | | | | | | | Bit 21 |
| - | | | | | | | Bit 22 |
| - | | | | | | | Bit 23 |
| loop | 0 | 0 | 0 | 0 | 0 | | Bit 24 |
| exen | 0 | 0 | 0 | 0 | 0 | | Bit 25 |
| amx0 | 0 | 0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | 0 | 0 | 0 | 0 | 0 | | Bit 27 |
| na | 0 | 0 | 0 | 0 | 0 | | Bit 28 |
| uta | 1 | 1 | 1 | 1 | 1 | | Bit 29 |
| todt | 0 | 0 | 0 | 0 | 1 | | Bit 30 |
| last | 0 | 0 | 0 | 0 | 1 | | Bit 31 |
| | **PTS** | **PTS+1** | **PTS+2** | **PTS+3** | **PTS+4** | | |

**Figure 15-48. Refresh Cycle (CBR) To Page Mode DRAM With Extended Data-Out**



| | | | |
|---|---|---|---|
| cst4 | 1 | | Bit 0 |
| cst1 | 1 | | Bit 1 |
| cst2 | 1 | | Bit 2 |
| cst3 | 1 | | Bit 3 |
| bst4 | 1 | | Bit 4 |
| bst1 | 1 | | Bit 5 |
| bst2 | 1 | | Bit 6 |
| bst3 | 1 | | Bit 7 |
| g0l0 | | | Bit 8 |
| g0l1 | | | Bit 9 |
| g0h0 | | | Bit 10 |
| g0h1 | | | Bit 11 |
| g1t4 | 1 | | Bit 12 |
| g1t3 | 1 | | Bit 13 |
| g2t4 | | | Bit 14 |
| g2t3 | | | Bit 15 |
| g3t4 | | | Bit 16 |
| g3t3 | | | Bit 17 |
| g4t4 | | | Bit 18 |
| g4t3 | | | Bit 19 |
| g5t4 | | | Bit 20 |
| g5t3 | | | Bit 21 |
| - | | | Bit 22 |
| - | | | Bit 23 |
| loop | 0 | | Bit 24 |
| exen | 0 | | Bit 25 |
| amx0 | 0 | | Bit 26 |
| amx1 | 0 | | Bit 27 |
| na | 0 | | Bit 28 |
| uta | 1 | | Bit 29 |
| todt | 1 | | Bit 30 |
| last | 1 | | Bit 31 |
| | EXS | | |

**Figure 15-49. Exception Cycle For Page Mode DRAM With Extended Data-Out**

## 15.4 MEMORY CONTROLLER EXTERNAL MASTER SUPPORT

The memory controller supports internal bus masters and, if enabled in the SIUMCR register, it will support accesses initiated by external bus masters. Refer to **Section 12.4.1.1 SIU Module Configuration Register** for more information. The external bus masters are classified into two types:

- Synchronous—Bus masters that work with CLKOUT, implementing the MPC821 bus protocol to access a slave device.
- Asynchronous—Bus masters that implement an asynchronous handshake with the slave device to perform a data transfer. The MC68030 and MC68360 are examples of this type of device.

A synchronous master initiates a transfer by asserting $\overline{\text{TS}}$. The address bus A(0:31) must be stable throughout the transaction, starting at the rising edge of CLKOUT in which $\overline{\text{TS}}$ is asserted until the last $\overline{\text{TA}}$ acknowledges the transfer. Since the external master works synchronously with the MPC821, only setup and hold times near the rising edge of CLKOUT are important. Assuming the SEME bit in the SIUMCR is set, once the $\overline{\text{TS}}$ is detected asserted, the memory controller compares the address with each one of it's defined valid banks. If a match is found, control signals to the memory devices is generated and the transfer acknowledge indication ($\overline{\text{TA}}$) is supplied to the master. Refer to Figure 15-50 for details.

An asynchronous master initiates a transfer by driving the address bus and asserting the $\overline{\text{AS}}$ signal. The address lines, together with RD/$\overline{\text{WR}}$ and the TSIZE(0:1) signals must be stable at setup time before the assertion of the $\overline{\text{AS}}$ pin. If the AEME bit in the SIUMCR register is set, the memory controller in the MPC821 synchronizes the $\overline{\text{AS}}$ assertion to its internal clock and generates the control line to the external memory devices. $\overline{\text{TA}}$ is given to the external master to acknowledge the transaction. All the control signals to the memory device and the $\overline{\text{TA}}$ signal are negated with the $\overline{\text{AS}}$ pin.

The MPC821 BADDR(28:30) pins are intended to connect to the memory devices the memory controller handles. They duplicate the value of the address lines A(28:30) when an internal master initiates a transaction on the external bus. When an external master initiates a transaction on the external bus, the BADDR(28:30) lines reflect the value of the A(28:30). Whether it is multiplexed or not depends on which machine controls the transaction and the corresponding attributes on the first memory access clock cycle. Afterwards, they behave as instructed by the memory controller machines.

To connect to external memory devices that require address multiplexing, the MPC821 provides (by means of the $\overline{\text{GPL5\_A}}$ and $\overline{\text{GPL5\_B}}$ signal lines) the ability to control external multiplexing logic. The $\overline{\text{GPL5\_x}}$ line logic value is changed if the user specifies when any of the UPM machines in the memory controller control the slave access. The $\overline{\text{GPL5\_x}}$ line reflects the value of the GL5S bit in the corresponding OR in the first clock cycle of the memory device access. In the following cycles, its value is determined by the bits g5t4 and g5t3 in the UPM RAM. If UPMB controls the slave access, the bit GL5A in the OR indicates that the value of GL5, g5t4 and g5t3 in the UPMB control the logical value of the $\overline{\text{GPL5\_A}}$ line. See Table 15-11 for details.

Notice that GL5S is taken into consideration only for read or write accesses to memory and not for patterns initiated by the UPM as a result of an internal periodic timer request or software request.

### Table 15-11. GPL5 Line Behavior

| MACHINE CONTROLLING MEMORY ACCESS | MEMORY ACCESS CLOCK CYCLE | G5LA | G5LS | g5t4 | g5t3 | GPL5_X |
|---|---|---|---|---|---|---|
| GPCM | x | x | x | x | x | GPL5_A and GPL5_B do not change their value. |
| UPMA | 1st | x | 0 | x | x | GPL5_A is driven low at the falling edge of GCLK1. |
| | | | 1 | | | GPL5_A is driven high at the falling edge of GCLK1. |
| | 2nd, 3rd, etc. | x | x | 0 | x | GPL5_A is driven low at the falling edge of GCLK2 in the current UPM cycle. |
| | | | | 1 | x | GPL5_A is driven high at the falling edge of GCLK2 in the current UPM cycle. |
| | | | | x | 0 | GPL5_A is driven low at the falling edge of GCLK1 in the current UPM cycle. |
| | | | | x | 1 | GPL5_A is driven high at the falling edge of GCLK1 in the current UPM cycle. |
| UPMB | 1st | 0 | 0 | x | x | GPL5_B is driven low at the falling edge of GCLK1. |
| | | | 1 | | | GPL5_B is driven high at the falling edge of GCLK1. |
| | | 1 | 0 | x | x | GPL5_A is driven low at the falling edge of GCLK1. |
| | | | 1 | | | GPL5_A is driven high at the falling edge of GCLK1. |
| | 2nd,3rd,.etc | 0 | 0 | 0 | x | GPL5_B is driven low at the falling edge of GCLK2 in the current UPM cycle. |
| | | | | 1 | x | GPL5_B is driven high at the falling edge of GCLK2 in the current UPM cycle. |
| | | | | x | 0 | GPL5_B is driven low at the falling edge of GCLK1 in the current UPM cycle. |
| | | | | x | 1 | GPL5_B is driven high at the falling edge of GCLK1 in the current UPM cycle. |
| UPMB | 2nd,3rd,.etc | 1 | x | 0 | x | GPL5_A is driven low at the falling edge of GCLK2 in the current UPM cycle. |
| | | | | 1 | x | GPL5_A is driven high at the falling edge of GCLK2 in the current UPM cycle. |
| | | | | x | 0 | GPL5_A is driven low at the falling edge of GCLK1 in the current UPM cycle. |
| | | | | x | 1 | GPL5_A is driven high at the falling edge of GCLK1 in the current UPM cycle. |

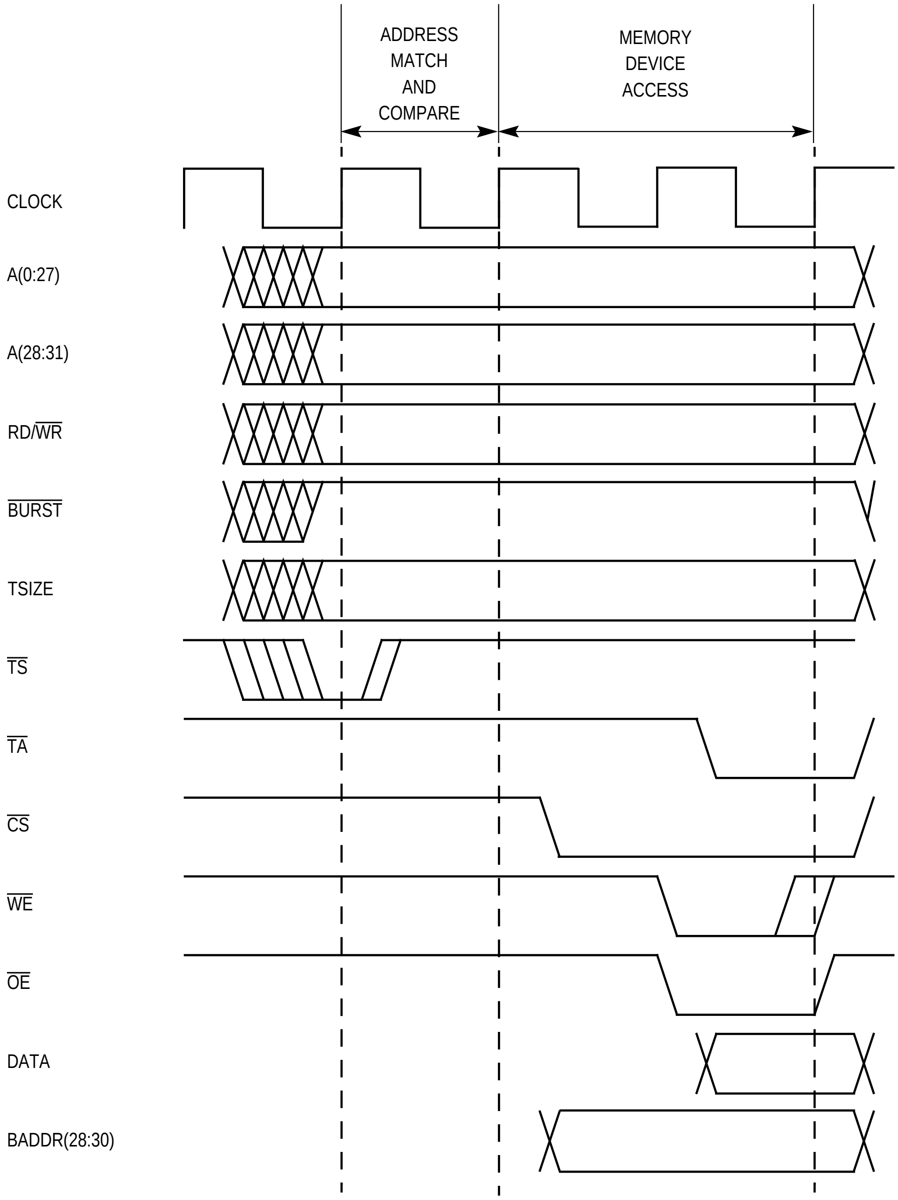


**Figure 15-50. Synchronous External Master Basic Access (GPCM Controlled)**

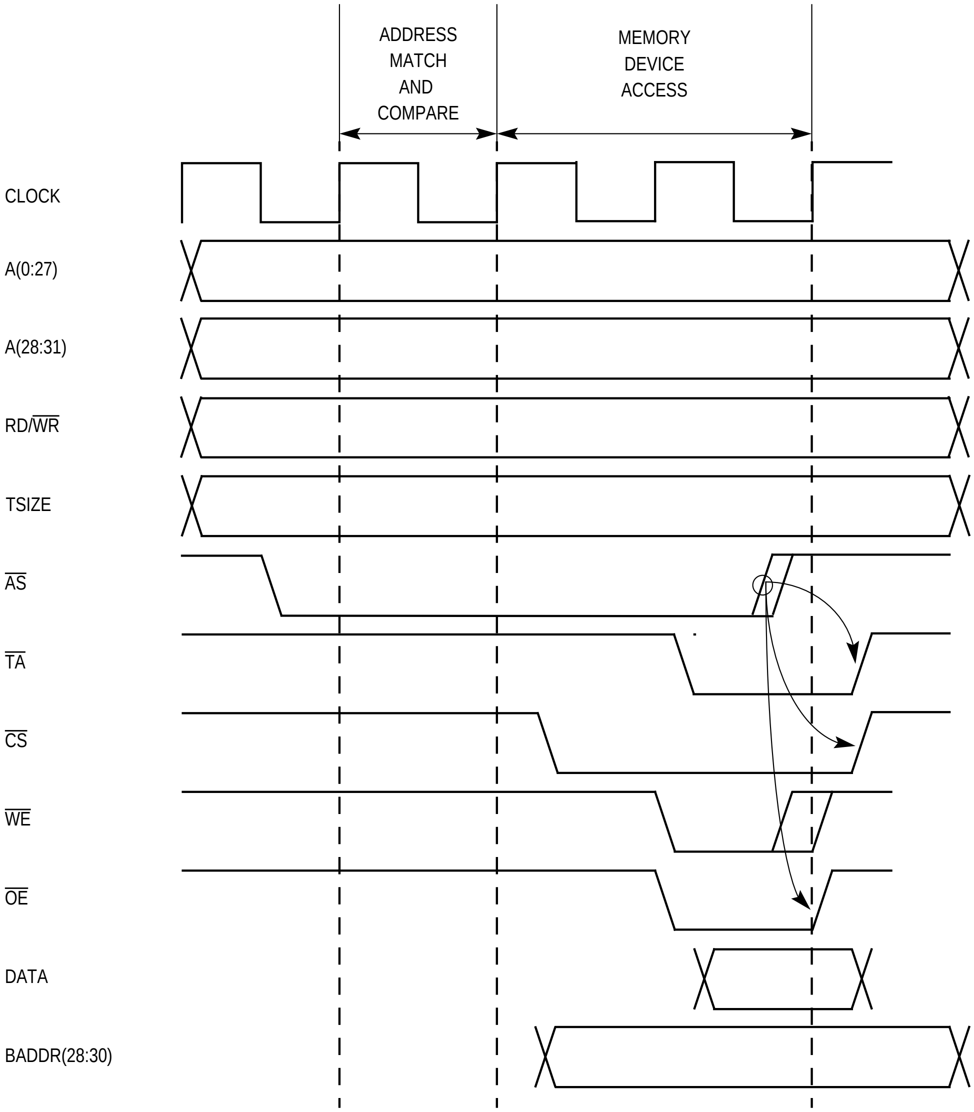


**Figure 15-51. Asynchronous External Master Basic Access (GPCM Controlled)**

Figure 15-52 illustrates a typical system configuration in which the MPC821 and an external master access a DRAM device. Figure 15-53 illustrates the timing behavior of the $\overline{\text{GPL5}}$ pin, BADDR lines, and the other control signals for a burst read access initiated by an external master to a DRAM device. The value of the $\overline{\text{GPL5}}$ pin in the first clock cycle of the memory device access is determined by the value of the GPL5S bit in the corresponding option register (“1”).

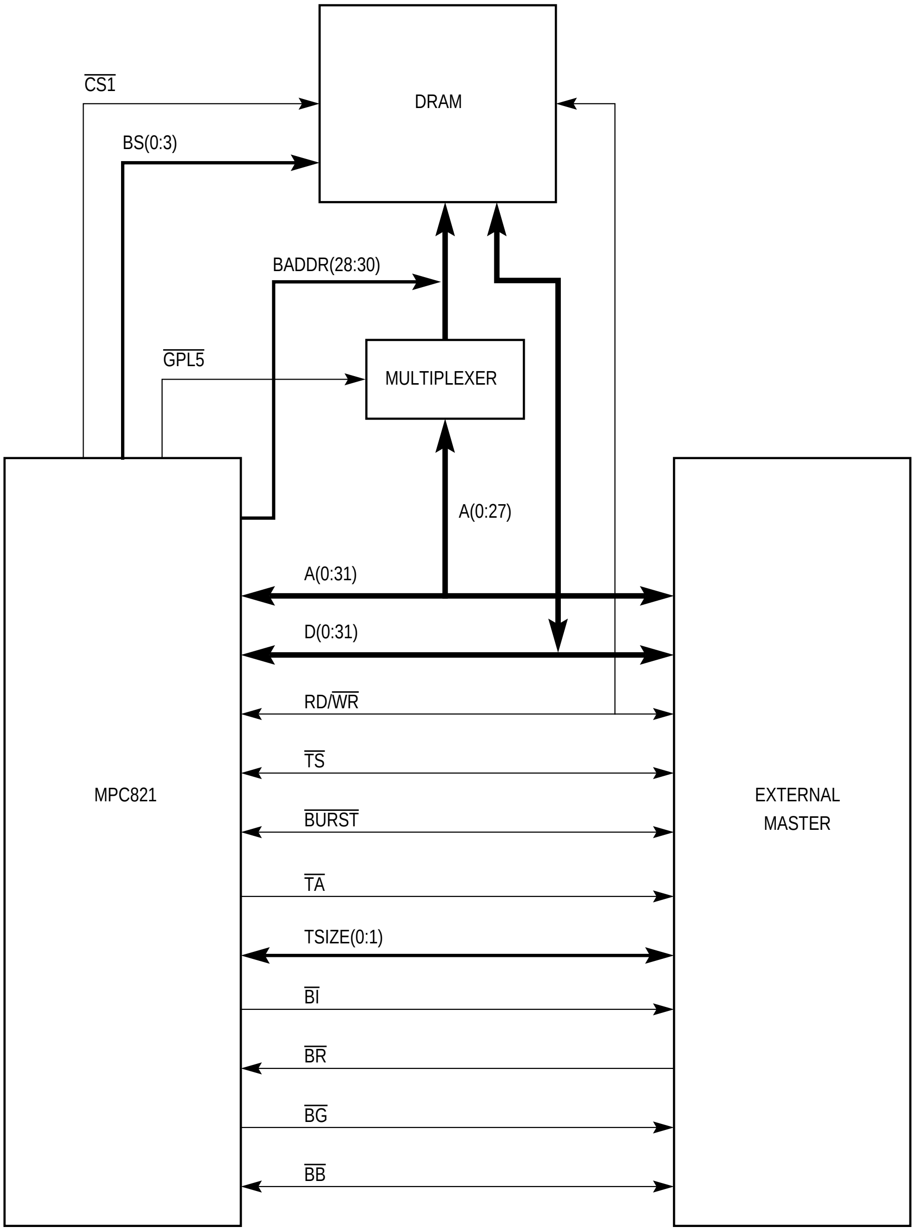
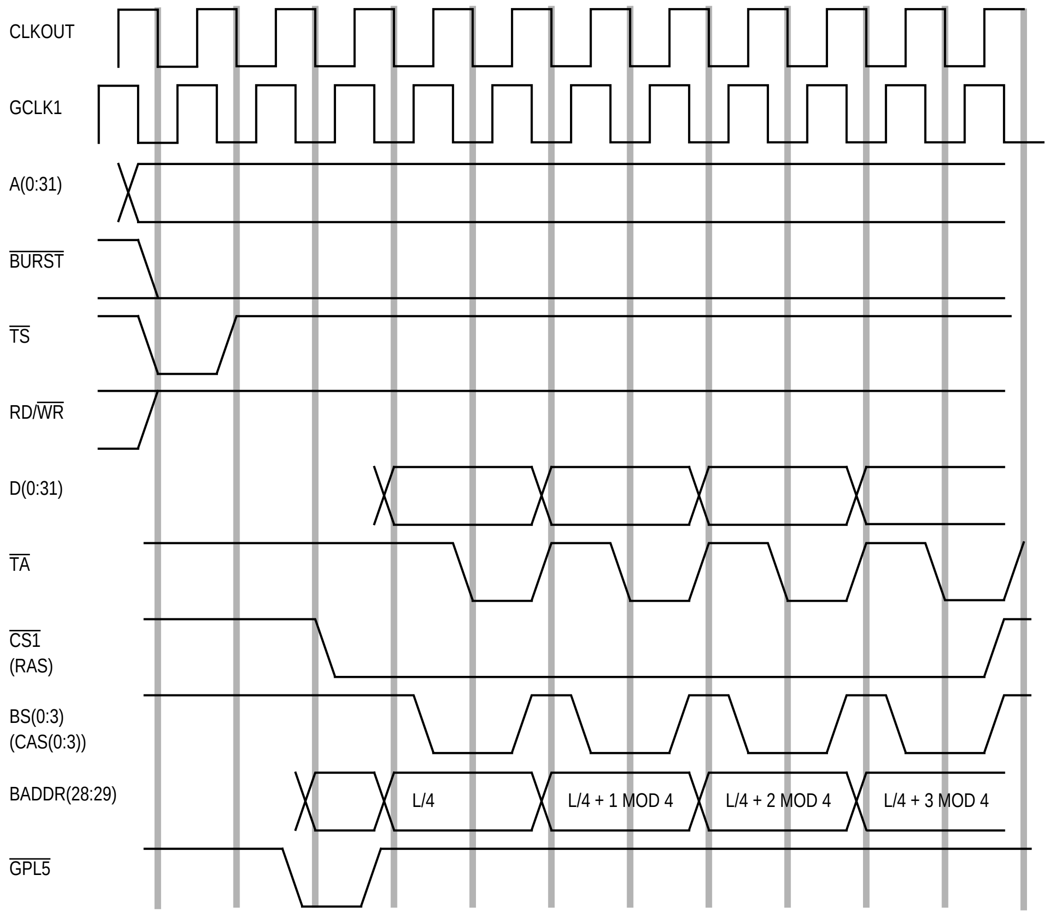


**Figure 15-52. Synchronous External Master–MPC821–DRAM Device Typical Configuration**



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| cst4 | (Bit 0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| cst1 | (Bit 1) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| cst2 | (Bit 2) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| cst3 | (Bit 3) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| bst4 | (Bit 4) | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| bst1 | (Bit 5) | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| bst2 | (Bit 6) | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| bst3 | (Bit 7) | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| g0l0 | (Bit 8) | | | | | | | | | |
| ≈ | | | | | | | | | | |
| g5t4 | (Bit 20) | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| g5t3 | (Bit 21) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| - | (Bit 22) | | | | | | | | | |
| - | (Bit 23) | | | | | | | | | |
| loop | (Bit 24) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| exen | (Bit 25) | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| amx0 | (Bit 26) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| amx1 | (Bit 27) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| na | (Bit 28) | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| uta | (Bit 29) | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| todt | (Bit 30) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| last | (Bit 31) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | **RBS** | **RBS+1** | **RBS+2** | **RBS+3** | **RBS+4** | **RBS+5** | **RBS+6** | **RBS+7** | **RBS+8** |

**Figure 15-53. Synchronous External Master–Burst Read Access To Page Mode DRAM**

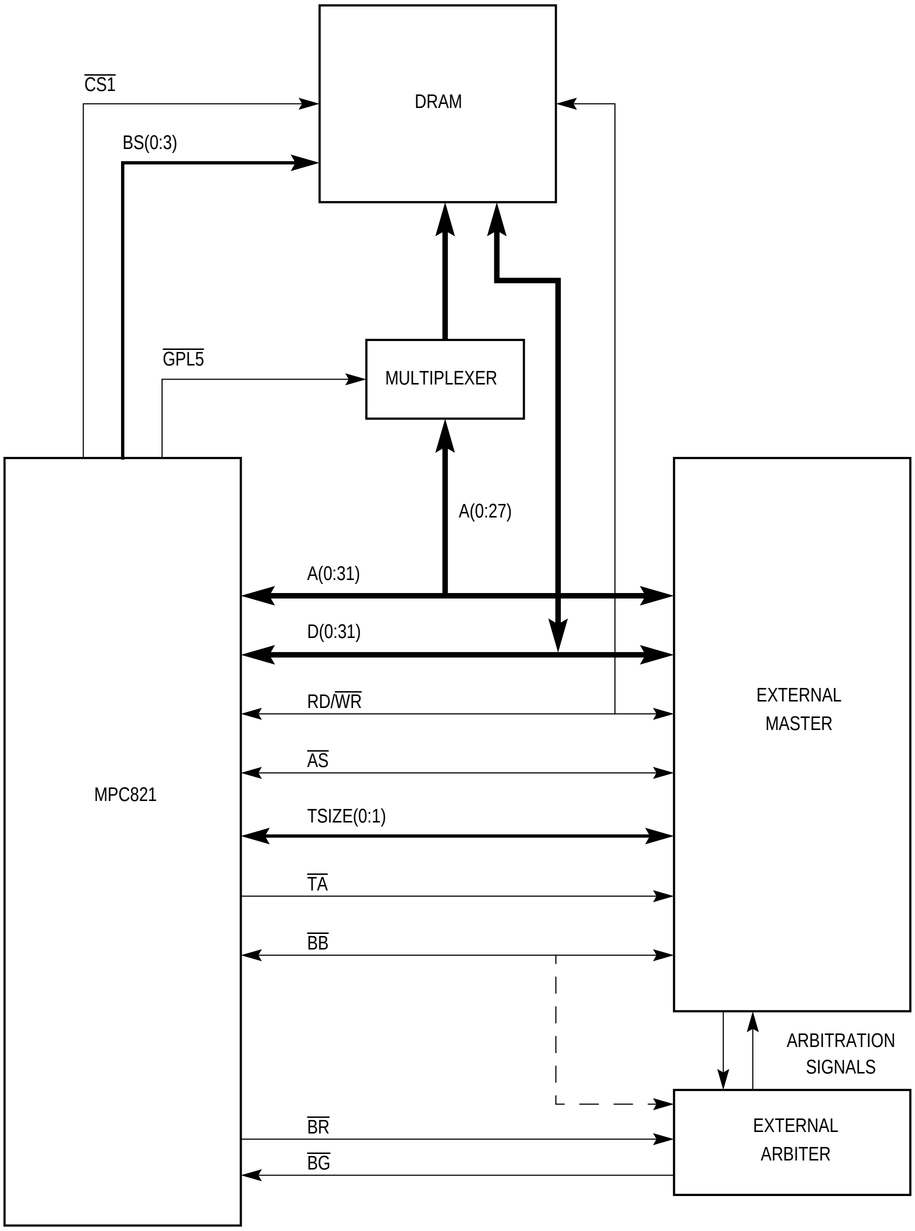
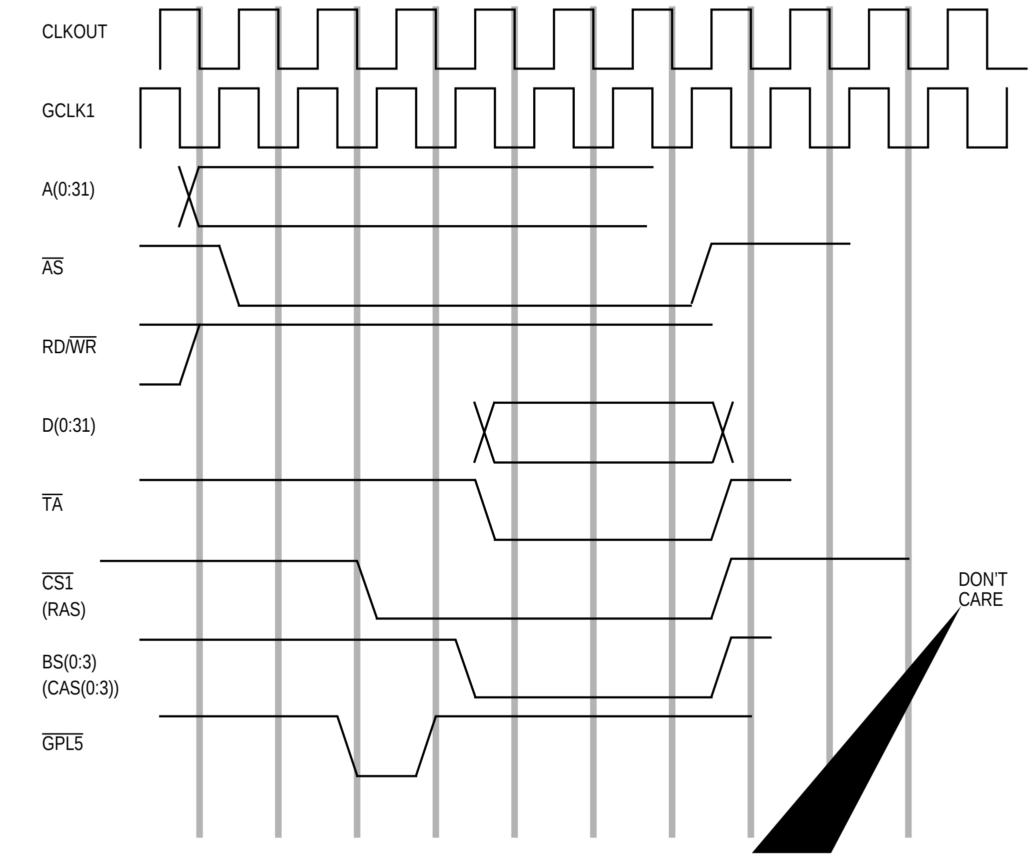


**Figure 15-54. Asynchronous External Master–MPC821–DRAM Device Typical Configuration**



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| cst4 | | | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 0 |
| cst1 | | | 0 | 0 | | | | 0 | | Bit 1 |
| cst2 | | | 0 | 0 | | | | 1 | | Bit 2 |
| cst3 | | | 0 | 0 | | | | 1 | | Bit 3 |
| bst4 | | | 1 | 1 | 0 | 0 | 0 | 0 | | Bit 4 |
| bst1 | | | 1 | 0 | | | | 0 | | Bit 5 |
| bst2 | | | 1 | 0 | | | | 1 | | Bit 6 |
| bst3 | | | 1 | 0 | | | | 1 | | Bit 7 |
| ≈ | | | | | | | | | | |
| g4t4 | | | | | | | | | | Bit 18 |
| g4t3 | | | | | | | | | | Bit 19 |
| g5t4 | | | | | 1 | 1 | 1 | | | Bit 20 |
| g5t3 | | | 0 | 1 | | | | 1 | | Bit 21 |
| - | | | | | | | | | | Bit 22 |
| - | | | | | | | | | | Bit 23 |
| loop | | | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 24 |
| exen | | | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 25 |
| amx0 | | | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 26 |
| amx1 | | | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 27 |
| na | | | 0 | 0 | 0 | 0 | 0 | 0 | | Bit 28 |
| uta | | | 1 | 0 | 0 | 0 | 0 | 1 | | Bit 29 |
| todt | | | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 30 |
| last | | | 0 | 0 | 0 | 0 | 0 | 1 | | Bit 31 |
| | | | **RSS** | **RSS+1** | **WAIT** | **WAIT** | **WAIT** | **RSS+2** | | |

**Figure 15-55. Asynchronous External Master–Read Access To Page Mode DRAM**

## 15.5 PROGRAMMING MODEL

The following registers are used to control the memory controller.

| NAME | DESCRIPTION |
|---|---|
| **MSTAT** | Memory Status Register |
| **BR0** | Base Register Bank 0 |
| **BR1** | Base Register Bank 1 |
| **BR2** | Base Register Bank 2 |
| BR3 | Base Register Bank 3 |
| **BR4** | Base Register Bank 4 |
| **BR5** | Base Register Bank 5 |
| **BR6** | Base Register Bank 6 |
| **BR7** | Base Register Bank 7 |
| **OR0** | Option Register Bank 0 |
| **OR1** | Option Register Bank 1 |
| **OR2** | Option Register Bank 2 |
| **OR3** | Option Register Bank 3 |
| **OR4** | Option Register Bank 4 |
| **OR5** | Option Register Bank 5 |
| **OR6** | Option Register Bank 6 |
| **OR7** | Option Register Bank 7 |
| **MAMR** | Machine A Mode Register |
| **MBMR** | Machine B Mode Register |
| **MPTPR** | Memory Periodic Timer Prescaler |
| **MCR** | Memory Command Register |
| **MDR** | Memory Data Register |
| **MAR** | Memory Address Register |

### Table 15-12. Memory Status Register

| BITS | MNEMONIC | DESCRIPTION / FUNCTION |
|---|---|---|
| 0 | PER0 | **PARITY ERROR BANK 0.**<br>This bit indicates that a parity error is detected when reading from Bank 0. PER0 is cleared by writing a one to this bit or performing a system reset. Writing a zero has no effect on PER0. |
| 1 | PER1 | **PARITY ERROR BANK 1.**<br>This bit indicates that a parity error is detected when reading from Bank 1. PER1 is cleared by writing a one to this bit or performing a system reset. Writing a zero has no effect on PER1. |

### Table 15-12. Memory Status Register (Continued)

| BITS | MNEMONIC | DESCRIPTION / FUNCTION |
|---|---|---|
| 2 | PER2 | **PARITY ERROR BANK 2.**<br>This bit indicates that a parity error is detected when reading from Bank 2. PER2 is cleared by writing a one to this bit or performing a system reset. Writing a zero has no effect on PER2. |
| 3 | PER3 | **PARITY ERROR BANK 3.**<br>This bit indicates that a parity error is detected when reading from Bank 3. PER3 is cleared by writing a one to this bit or performing a system reset. Writing a zero has no effect on PER3. |
| 4 | PER4 | **PARITY ERROR BANK 4.**<br>This bit indicates that a parity error is detected when reading from Bank 4. PER4 is cleared by writing a one to this bit or performing a system reset. Writing a zero has no effect on PER4. |
| 5 | PER5 | **PARITY ERROR BANK 5.**<br>This bit indicates that a parity error is detected when reading from Bank 5. PER5 is cleared by writing a one to this bit or performing a system reset. Writing a zero has no effect on PER5. |
| 6 | PER6 | **PARITY ERROR BANK 6.**<br>This bit indicates that a parity error is detected when reading from Bank 6. PER6 is cleared by writing a one to this bit or performing a system reset. Writing a zero has no effect on PER6. |
| 7 | PER7 | **PARITY ERROR BANK 7.**<br>This bit indicates that a parity error is detected when reading from Bank 7. PER7 is cleared by writing a one to this bit or performing a system reset. Writing a zero has no effect on PER7. |
| 8 | WPER | **WRITE PROTECTION ERROR.**<br>This bit is asserted when a write protect error occurs. A bus monitor ($\overline{\text{TEA}}$ assertion) (if enabled) will prompt the user to read this register if no $\overline{\text{TA}}$ is provided on a write cycle. WPER is cleared by writing a one to this bit or performing a system reset. Writing a zero has no effect on WPER. |
| 9-15 | Reserved | |

### Table 15-13. Memory Periodic Timer Prescaler Register

| BITS | MNEMONIC | DESCRIPTION / FUNCTION | |
|---|---|---|---|
| 0-7 | PTP(0:7) | **PERIODIC TIMERS PRESCALER.**<br>This attribute determines the period of the memory periodic timers input clock.<br>The BRGCLK clock is divided according to the encoding of these bits. | 001x xxxx = divide by 2<br>0001 xxxx = divide by 4<br>0000 1xxx = divide by 8<br>0000 01xx = divide by 16<br>0000 001x = divide by 32<br>0000 0001 = divide by 64 |
| 8-15 | Reserved | | |

### Table 15-14. Base Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-16 | BA(0:16) | **BASE ADDRESS**.<br>The base address field, the upper 17 bits of each base address register, and the address type code field are compared to the address on the address bus to determine if a memory bank controlled by the memory controller is being accessed by an internal bus master. These bits are used in conjunction with the AM(0:16) bits in the OR. | |
| 17-19 | AT(0:2) | **ADDRESS TYPE**.<br>This field can be used to limit accesses to the memory bank to a certain address space type. These bits are used in conjunction with the ATM(0:2) bits in the OR. | |
| 20-21 | PS(0:1) | **PORT SIZE.**<br>This field specifies the port size of this memory region. | 11 = Reserved<br>01 = 8 bits port size<br>10 = 16 bits port size<br>00 = 32 bits port size |
| 22 | PARE | **PARITY ENABLE**.<br>This bit is used to enable of parity checking on this bank. | 0 = Parity checking disable<br>1 = Parity checking enable |
| 23 | WP | **WRITE PROTECT**.<br>This bit can restrict write accesses within the address range of a base register. An attempt to write to the range of addresses specified in a base address register that has this bit set can cause the $\overline{\text{TEA}}$ signal to be asserted by the bus monitor logic (if enabled) causing termination of this cycle. | 0 = Both read and write accesses are allowed.<br>1 = Only read accesses are allowed. The $\overline{\text{CSx}}$ signal, $\overline{\text{TA}}$ is not asserted by the memory controller on write cycles to this memory bank. WPER is set in the MSTAT register if a write to this memory bank is attempted. |
| 24-25 | MS(0:1) | **MACHINE SELECT**.<br>This field specifies the machine selected for the memory operations handling. | 00 = G.P.C.M<br>0X = Reserved<br>10 = U.P.M.A<br>11 = U.P.M.B |
| 26-30 | Reserved | | |
| 31 | V | **VALID BIT**.<br>This bit indicates that the contents of the base register and option register pair are valid. The $\overline{\text{CS}}$ signal does not assert until the V-bit is set.<br>Note: An access to a region does not have the V bit set may cause a bus monitor timeout.<br>Note: Following a system reset, the V bit is set in BR0. | 0 = This bank is invalid.<br>1 = This bank is valid. |

### Table 15-15. Option Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-16 | AM(0:16) | **ADDRESS MASK**.<br>The address mask provides masking on any corresponding bits in the associated base register. By masking the address bits independently, external devices of different size address ranges can be used. Any clear bit masks the corresponding address bit. Any set bit causes the corresponding address bit to be used in address pin comparison. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. This field can be read or written at anytime.<br>Note: Following a system reset, the AM bits are reset in OR0. | |
| 17-19 | ATM(0:2) | **ADDRESS TYPE MASK**.<br>This field can be used to mask certain address type bits, 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.<br>Note: Clear the ATM bits to ignore address type codes as part of the address comparison.<br>Note: Following a system reset, the ATM bits are reset in OR0. | |
| 20 | CSNT/SAM | **CHIP SELECT NEGATION TIME.**<br>This attribute is used to determine when $\overline{\text{CS}}$/$\overline{\text{WE}}$ are negated during an external memory write access handled by the general- purpose chip-select machine. This helps in meeting address/data hold time requirements for slow memories and peripherals.<br>Note: Following a system reset, the CSNT bit is set in OR0. | 0 = $\overline{\text{CS}}$/$\overline{\text{WE}}$ are negated normally.<br>1 = $\overline{\text{CS}}$/$\overline{\text{WE}}$ are negated a quarter of a clock earlier. |
| | | **START ADDRESS MULTIPLEX**.<br>This attribute determines how the address is output on the first cycle of an external memory access read or write when the memory access is handled by UPMA or UPMB. | 0 = Address pins are the address requested by the internal master.<br>1 = Address pins are the address requested by the internal master multiplexed according to:<br>AMA field if UPMA is selected to control the memory access or,<br>AMB field if UPMB is selected to control the memory access. |

## Table 15-15. Option Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 21-22 | ACS(0:1) / G5LA,G5LS | **ADDRESS TO CHIP-SELECT SETUP**.<br>This attribute can be used when the external memory access is handled by the general-purpose chip-select machine. It allows the $\overline{CS}$ assertion to be delayed relative to the address change.<br>Note: Following a system reset, the ACS bits are set in OR0. | 00 = $\overline{CS}$ is output at the same time as the address lines are.<br>01 = Reserved<br>10 = $\overline{CS}$ is output a quarter of a clock later than the address lines.<br>11 = $\overline{CS}$ is output half a clock later than the address lines. |
| | | **GENERAL-PURPOSE LINE 5 A, GENERAL-PURPOSE LINE 5 START**.<br>These attributes determine how the $\overline{GPL5}$ pin is output when the memory access is handled by UPMA or UPMB. | G5LA (valid only if MS = 11 in $\overline{BR}$)<br>0 output $\overline{GPL5}$ on $\overline{GPL5\_B}$ line<br>1 output $\overline{GPL5}$ on $\overline{GPL5\_A}$ line<br><br>G5LS<br>0 The $\overline{GPL5}$ line is driven low on the falling edge of GCLK1 during the first clock cycle of a read or write memory access.<br>1 The $\overline{GPL5}$ line is driven high on the falling edge of GCLK1 during the first clock cycle of a read or write memory access. |
| 23 | $\overline{BI}$ | **BURST INHIBIT**.<br>This attribute determines whether or not this memory bank supports burst accesses. In the nonburst case, the memory controller drives the $\overline{BI}$ signal active when accessing this memory region.<br>If the machine selected to handle this access is GPCM, this bit must be 1.<br>Note: Following a system reset, the $\overline{BI}$ bit is set in OR0. | 0 = Drive $\overline{BI}$ negated. The bank supports burst accesses.<br>1 = Drive $\overline{BI}$ asserted. The bank does not support burst accesses. |
| 24-27 | SCY(0:3) | **CYCLE LENGTH IN CLOCKS**.<br>This attribute determines the number of wait states inserted in the cycle, when the GPCM handles the external memory access and, thus, it is the main parameter for determining the length of the cycle.<br>The total cycle length may vary depending on the settings of other timing attributes.<br>The total memory access length is $(2 + \text{SCY}) \times \text{Clocks}$.<br>If the user has selected an external $\overline{TA}$ response for this memory bank (by setting the SETA bit), then SCY(0:3) are not used.<br>Note: Following a system reset, the SCY bit are set to 1111 in OR0. | 0000 = 0 clock cycle wait state<br>0001 = 1 clock cycle wait state<br>0010 = 2 clock cycles wait states<br>0011 = 3 clock cycles wait states<br>0100 = 4 clock cycles wait states<br>0101 = 5 clock cycles wait states<br>0110 = 6 clock cycles wait states<br>0111 = 7 clock cycles wait states<br>1000 = 8 clock cycles wait states<br>1001 = 9 clock cycles wait states<br>1010 = 10 clock cycles wait states<br>1011 = 11 clock cycles wait states<br>1100 = 12 clock cycles wait states<br>1101 = 13 clock cycles wait states<br>1110 = 14 clock cycles wait states<br>1111 = 15 clock cycles wait states |

## Table 15-15. Option Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 28 | SETA | **EXTERNAL TRANSFER ACKNOWLEDGE**.<br>This bit specifies that the $\overline{TA}$ signal is generated externally when the GPCM is selected to handle the memory access initiated to this memory region.<br>Note: Following a system reset, the SETA bit is reset in OR0. | 0 = $\overline{TA}$ is generated internally by the memory controller. Unless asserted earlier externally.<br>1 = $\overline{TA}$ is generated by external logic. |
| 29 | TRLX | **TIMING RELAXED**.<br>This bit, when asserted, modifies the timing of the signals that control the memory devices when the GPCM is selected to handle the memory access initiated to this memory region.<br>Note: Following a system reset, the TRLX bit is set in OR0. | 0 = Normal timing is generated by the GPCM.<br>1 = Relaxed timing is generated by the GPCM. |
| 30 | EHTR | **EXTENDED HOLD TIME ON READ ACCESSES**.<br>This bit, when asserted inserts an idle clock cycle after a read access from the current bank and any MPC821 write or read access to a different bank.<br>Note: Following a system reset, the EHTR bit is reset in OR0. | 0 = Normal timing is generated by the memory controller.<br>1 = Extended hold time is generated by the memory controller. |
| 31 | Reserved | | |

## Table 15-16. Machine A Mode Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-7 | PTA(0:7) | **PERIODIC TIMER A PERIOD**.<br>This attribute affects the periodic timer A. These bits determine the timer period according to the following equation:<br>$TimerPeriod = \left(\frac{PTA}{F_{MPTC}}\right)$<br>Example:<br>For a 25 MHz BRGCLK and a required service rate of 15.6 microseconds, given PTP = 32, the PTA value should be 12 decimal.<br>12/ (25 MHz / 32) = 15.36 microseconds which is less than the required service period of 15.6 microseconds. | |

### Table 15-16. Machine A Mode Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 8 | PTAE | **PERIODIC TIMER A ENABLE**. This bit allows the periodic timer A to request service.<br>Note: Following a system reset, the PTAE bit is reset. | 0 = Periodic timer A is disabled.<br>1 = Periodic timer A is enabled. |
| 9-11 | AMA(0:2) | **ADDRESS MULTIPLEX SIZE A.** This field determines how the address of the current memory cycle is output on the address pins. The effective control of the address output on the pins is done by the contents of the RAM array in the UPMA. This field is useful when connecting the MPC821 to DRAM devices requiring row and column addresses multiplexed on the same pins. | See Table 15-5 for details. |
| 12 | Reserved | | |
| 13-14 | DSA(0:1) | **DISABLE TIMER PERIOD**. This attribute guarantees a minimum time between accesses to the same memory bank if it is controlled by the UPMA. The disable timer is turned on by the TODT in the RAM array and, when expired, the UPMA will allows the machine access to handle issuing a memory pattern to the same memory region. Accesses to different memory regions can be handled by the same UPMA.<br>Note: To avoid conflicts between successive accesses to different memory regions, the minimum pattern in the RAM array for a request serviced should be equal to or greater than the period established by DSA. | 00 = 1 cycle disable period<br>01 = 2 cycle disable period<br>10 = 3 cycle disable period<br>11 = 4 cycle disable period |
| 15 | Reserved | | |
| 16-18 | G0CLA | **GENERAL LINE 0 CONTROL A**. This field determines which address line can be output to the $\overline{\text{GPL0}}$ pin when the UPMA is selected to control the memory access. | 000 = A12<br>001 = A11<br>010 = A10<br>011 = A9<br>100 = A8<br>101 = A7<br>110 = A6<br>111 = A5 |

### Table 15-16. Machine A Mode Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 19 | GPL_A4DIS | **GPL_A4 OUTPUT LINE DISABLE.** This bit determines if the UPWAITA/ GPL_A4 pin will behave as an output line controlled by the corresponding bits in the UPMA array (GPL4A).<br>Note: Following a system reset, the GPL_A4DIS bit is set. | 0 = UPWAITA/GPL_A4 behaves as GPL_A4 when:<br>The G4T4/DLT3 bit in the UPMA is interpreted as G4T4.<br>The G4T3/WAEN bit in the UPMA is interpreted as G4T3.<br>1 = UPWAITA/GPL_A4 behaves as UPWAITA when:<br>The G4T4/DLT3 bit in the UPMA is interpreted as DLT3.<br>The G4T3/WAEN bit in the UPMA is interpreted as WAEN. |
| 20-23 | RLFA(0:3) | **READ LOOP FIELD A**. This field determines the number of times a loop defined in the UPMA is executed for a burst read or a single beat read pattern. | 0001 = The loop is executed 1 time<br>0010 = The loop is executed 2 times<br>0011 = The loop is executed 3 times<br>0100 = The loop is executed 4 times<br>0101 = The loop is executed 5 times<br>0110 = The loop is executed 6 times<br>0111 = The loop is executed 7 times<br>1000 = The loop is executed 8 times<br>1001 = The loop is executed 9 times<br>1010 = The loop is executed 10 times<br>1011 = The loop is executed 11 times<br>1100 = The loop is executed 12 times<br>1101 = The loop is executed 13 times<br>1110 = The loop is executed 14 times<br>1111 = The loop is executed 15 times<br>0000 = The loop is executed 16 times |
| 24-27 | WLFA(0:3) | **WRITE LOOP FIELD A**. This field determines the number of times a loop defined in the UPMA is executed for a burst write or a single beat write patterns. | 0001 = The loop is executed 1 time<br>0010 = The loop is executed 2 times<br>0011 = The loop is executed 3 times<br>0100 = The loop is executed 4 times<br>0101 = The loop is executed 5 times<br>0110 = The loop is executed 6 times<br>0111 = The loop is executed 7 times<br>1000 = The loop is executed 8 times<br>1001 = The loop is executed 9 times<br>1010 = The loop is executed 10 times<br>1011 = The loop is executed 11 times<br>1100 = The loop is executed 12 times<br>1101 = The loop is executed 13 times<br>1110 = The loop is executed 14 times<br>1111 = The loop is executed 15 times<br>0000 = The loop is executed 16 times |

### Table 15-16. Machine A Mode Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 28-31 | TLFA(0:3) | **TIMER LOOP FIELD A**.<br>This field determines the number of times a loop defined in the UPMA is executed for a periodic timer service pattern. | 0001 = The loop is executed 1 time<br>0010 = The loop is executed 2 times<br>0011 = The loop is executed 3 times<br>0100 = The loop is executed 4 times<br>0101 = The loop is executed 5 times<br>0110 = The loop is executed 6 times<br>0111 = The loop is executed 7 times<br>1000 = The loop is executed 8 times<br>1001 = The loop is executed 9 times<br>1010 = The loop is executed 10 times<br>1011 = The loop is executed 11 times<br>1100 = The loop is executed 12 times<br>1101 = The loop is executed 13 times<br>1110 = The loop is executed 14 times<br>1111 = The loop is executed 15 times<br>0000 = The loop is executed 16 times |

### Table 15-17. Machine B Mode Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-7 | PTB(0:7) | **PERIODIC TIMER B PERIOD**.<br>This attribute affects the periodic timer B. These bits determine the timer period according to the following equation:<br>$\mathrm{TimerPeriod} = \left(\frac{\mathrm{PTB}}{F_{\mathrm{MPTC}}}\right)$<br>Example:<br>For a 25 MHz BRGCLK and a required service rate of 15.6 microseconds, given PTP = 32, the PTB value should be 12 decimal.<br>12/ (25 MHz / 32) = 15.36 microseconds which is less than the required service period of 15.6 microseconds. | |
| 8 | PTBE | **PERIODIC TIMER B ENABLE**.<br>This bit allows the periodic timer B to request service.<br>Note: Following a system reset, the PTBE bit is reset. | 0 = Periodic Timer B is disabled.<br>1 = Periodic Timer B is enabled. |

### Table 15-17. Machine B Mode Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 9-11 | AMB(0:2) | **ADDRESS MULTIPLEX SIZE B.**<br>This field determines how the address of the current memory cycle is output on the address pins. The effective control of the address output on the pins is done by the contents of the RAM array in the UPMB. This field is useful when connecting the MPC821 to DRAM devices requiring row and column addresses multiplexed on the same pins. | See Table 15-5 for details. |
| 12 | Reserved | | |
| 13-14 | DSB(0:1) | **DISABLE TIMER PERIOD**.<br>This attribute guarantees a minimum time between accesses to the same memory bank if it is controlled by the UPMB. The disable timer is turned on by the TODT in the RAM array and, when expired, the UPMB allows the machine access to handle a memory pattern to the same memory region. If an access to a different memory region is handled by the same UPMB, it is allowed.<br>Note: To avoid conflicts between successive accesses to different memory regions, the minimum pattern in the RAM array for a request serviced should be equal to or greater than the period established by DSB. | 00 = 1 cycle disable period<br>01 = 2 cycle disable period<br>10 = 3 cycle disable period<br>11 = 4 cycle disable period |
| 15 | Reserved | | |
| 16-18 | G0CLB | **GENERAL LINE 0 CONTROL B**.<br>This field determines which address line is output to the GPL0 pin when the UPMB is selected to control the memory access. | 000 = A12<br>001 = A11<br>010 = A10<br>011 = A9<br>100 = A8<br>101 = A7<br>110 = A6<br>111 = A5 |
| 19 | GPL_B4DIS | **GPL_B4 OUTPUT LINE DISABLE.**<br>This bit determines if the UPWAITB/ GPL_B4 pin behaves as an output line controlled by the corresponding bits in the UPMB array (GPL4B).<br>Note: Following a system reset, the GPL_B4DIS bit is set. | 0 = UPWAITB/GPL_B4 behaves as GPL_B4 when:<br>The G4T4/DLT3 bit in the UPMB is interpreted as G4T4.<br>The G4T3/WAEN bit in the UPMB is interpreted as G4T3.<br>1 = UPWAITB/GPL_B4 behaves as UPWAITB when:<br>The G4T4/DLT3 bit in the UPMB is interpreted as DLT3.<br>The G4T3/WAEN bit in the UPMB is interpreted as WAEN. |

## Table 15-17. Machine B Mode Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 20-23 | RLFB(0:3) | **READ LOOP FIELD B**.<br>This field determines the number of times a loop defined in the UPMB is executed for a burst read or a single beat read pattern. | 0001 = The loop is executed 1 time<br>0010 = The loop is executed 2 times<br>0011 = The loop is executed 3 times<br>0100 = The loop is executed 4 times<br>0101 = The loop is executed 5 times<br>0110 = The loop is executed 6 times<br>0111 = The loop is executed 7 times<br>1000 = The loop is executed 8 times<br>1001 = The loop is executed 9 times<br>1010 = The loop is executed 10 times<br>1011 = The loop is executed 11 times<br>1100 = The loop is executed 12 times<br>1101 = The loop is executed 13 times<br>1110 = The loop is executed 14 times<br>1111 = The loop is executed 15 times<br>0000 = The loop is executed 16 times |
| 24-27 | WLFB(0:3) | **WRITE LOOP FIELD B**.<br>This field determines the number of times a loop defined in the UPMB is executed for a burst write or a single beat write pattern. | 0001 = The loop is executed 1 time<br>0010 = The loop is executed 2 times<br>0011 = The loop is executed 3 times<br>0100 = The loop is executed 4 times<br>0101 = The loop is executed 5 times<br>0110 = The loop is executed 6 times<br>0111 = The loop is executed 7 times<br>1000 = The loop is executed 8 times<br>1001 = The loop is executed 9 times<br>1010 = The loop is executed 10 times<br>1011 = The loop is executed 11 times<br>1100 = The loop is executed 12 times<br>1101 = The loop is executed 13 times<br>1110 = The loop is executed 14 times<br>1111 = The loop is executed 15 times<br>0000 = The loop is executed 16 times |

## Table 15-17. Machine B Mode Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 28-31 | TLFB(0:3) | **TIMER LOOP FIELD B**.<br>This field determines the number of times a loop defined in the UPMB is executed for a periodic timer service pattern. | 0001 = The loop is executed 1 time<br>0010 = The loop is executed 2 times<br>0011 = The loop is executed 3 times<br>0100 = The loop is executed 4 times<br>0101 = The loop is executed 5 times<br>0110 = The loop is executed 6 times<br>0111 = The loop is executed 7 times<br>1000 = The loop is executed 8 times<br>1001 = The loop is executed 9 times<br>1010 = The loop is executed 10 times<br>1011 = The loop is executed 11 times<br>1100 = The loop is executed 12 times<br>1101 = The loop is executed 13 times<br>1110 = The loop is executed 14 times<br>1111 = The loop is executed 15 times<br>0000 = The loop is executed 16 times |

## Table 15-18. Memory Command Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-1 | OP(0:1) | **COMMAND OPCODE**.<br>This field determines which command is executed by the machine specified in the UM field. | 00 = WRITE. Write the contents of the MDR into the RAM location pointed by MAD in the UPM specified in UM.<br>01 = READ. Read the contents of the RAM location pointed by MAD in the UPM specified in UM into the MDR.<br>10 = RUN. Run the pattern written in the RAM array of the UPM specified in UM servicing the memory bank specified in MB. The pattern run starts at the location pointed by MAD and continues until the LAST bit in the RAM is set.<br>11 = Reserved |
| 2-7 | Reserved | | |
| 8 | UM | **USER MACHINE**.<br>This field indicates which UPM the command should be executed to. | 0 = UPMA<br>1 = UPMB |
| 9-15 | Reserved | | |

### Table 15-18. Memory Command Register (Continued)

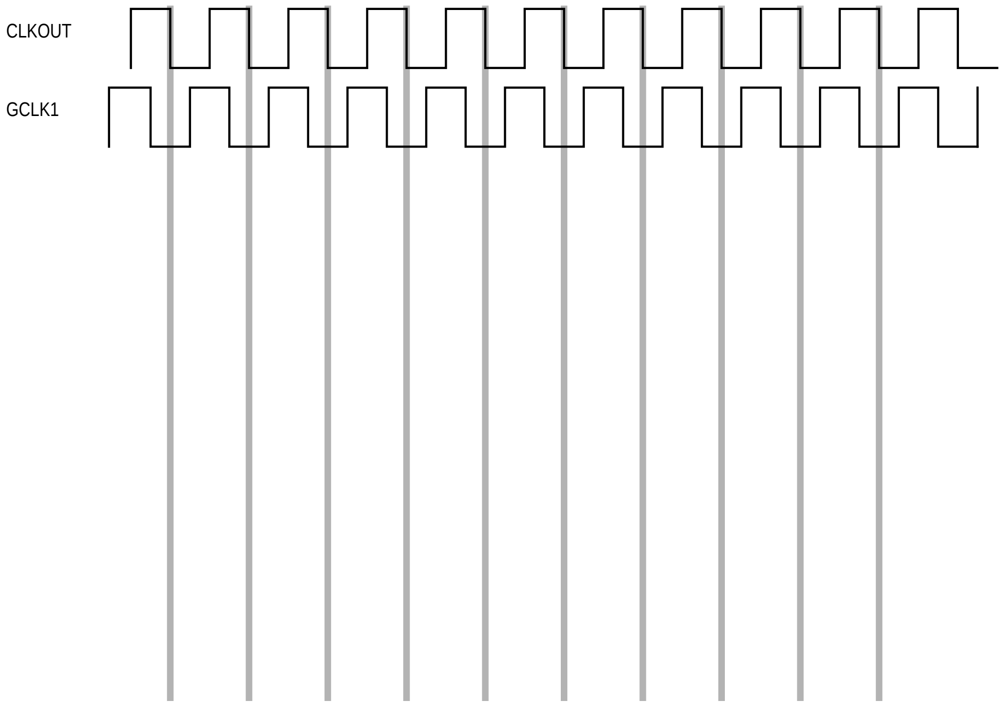
| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 16-18 | MB(0:2) | **MEMORY BANK**.<br>This field indicates which $\overline{CS}$ line is enabled when executing a RUN_PATTERN command. | 000 = $\overline{CS0}$ enabled<br>001 = $\overline{CS1}$ enabled<br>010 = $\overline{CS2}$ enabled<br>011 = $\overline{CS3}$ enabled<br>100 = $\overline{CS4}$ enabled<br>101 = $\overline{CS5}$ enabled<br>110 = $\overline{CS6}$ enabled<br>111 = $\overline{CS7}$ enabled |
| 19 | Reserved | | |
| 20-23 | MCLF(0:3) | **MEMORY COMMAND LOOP FIELD**.<br>This field determines the number of times a loop is executed for a MEMORY COMMAND service pattern. | 0001 = The loop is executed 1 time<br>0010 = The loop is executed 2 times<br>0011 = The loop is executed 3 times<br>0100 = The loop is executed 4 times<br>0101 = The loop is executed 5 times<br>0110 = The loop is executed 6 times<br>0111 = The loop is executed 7 times<br>1000 = The loop is executed 8 times<br>1001 = The loop is executed 9 times<br>1010 = The loop is executed 10 times<br>1011 = The loop is executed 11 times<br>1100 = The loop is executed 12 times<br>1101 = The loop is executed 13 times<br>1110 = The loop is executed 14 times<br>1111 = The loop is executed 15 times<br>0000 = The loop is executed 16 times |
| 24-25 | Reserved | | |
| 26-31 | MAD(0:5) | **MACHINE ADDRESS**.<br>This field is the RAM address pointer of the executed command. | |

### Table 15-19. Memory Data Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-31 | MD(0:31) | **MEMORY DATA**.<br>This is the data to be written into the RAM array when a WRITE command is supplied to the MCR. This is the data read from the array when a READ command is supplied to the MCR. | |

### Table 15-20. Memory Address Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-31 | A(0:31) | **MEMORY ADDRESS**.<br>This is the memory address register that is output to the address lines under control of the AMX bits in the UPM. | |



| | | | | | |
|---|---|---|---|---|---|
| cst4<br>cst1<br>cst2<br>cst3 | | | | | Bit 0<br>Bit 1<br>Bit 2<br>Bit 3 |
| bst4<br>bst1<br>bst2<br>bst3 | | | | | Bit 4<br>Bit 5<br>Bit 6<br>Bit 7 |
| g0l0<br>g0l1<br>g0h0<br>g0h1 | | | | | Bit 8<br>Bit 9<br>Bit 10<br>Bit 11 |
| g1t4<br>g1t3 | | | | | Bit 12<br>Bit 13 |
| g2t4<br>g2t3 | | | | | Bit 14<br>Bit 15 |
| g3t4<br>g3t3 | | | | | Bit 16<br>Bit 17 |
| g4t4<br>g4t3 | | | | | Bit 18<br>Bit 19 |
| g5t4<br>g5t3 | | | | | Bit 20<br>Bit 21 |
| -<br>- | | | | | Bit 22<br>Bit 23 |
| loop<br>exen | | | | | Bit 24<br>Bit 25 |
| amx0<br>amx1<br>na<br>uta<br>todt<br>last | | | | | Bit 26<br>Bit 27<br>Bit 28<br>Bit 29<br>Bit 30<br>Bit 31 |
| | RSS | RSS+1 | RSS+2 | | |

**Figure 11-35. Blank Worksheet for UPM**

# SECTION 16
# COMMUNICATION PROCESSOR MODULE

## 16.1 INTRODUCTION

The MPC821 communication processor module (CPM) is a derivative of the MC68360 QUICC CPM, so the *MC68360 Quad Integrated Communications Controller (QUICC*™ *) User’s Manual* can provide details. Throughout the section below, **bold** text highlights the features of the MPC821 CPM that are above the functions of the MC68360.

## 16.2 FEATURES

The CPM includes various blocks to provide the system with an efficient means of handling data communication tasks: The following is a list of the CPM’s important features.

- RISC controller
  - One instruction per clock
  - Executes code from internal ROM/dual-ported RAM
  - 32-bit architecture
  - Tuned for communication environments: instruction set supports **16-bit multiply accumulate**, CRC computation, and bit manipulation.
  - Internal timer
  - Interfaces with the PowerPC™ CPU through a **5 kbyte** dual-ported RAM and virtual DMA channels for each serial channel
  - Handles serial protocols, **virtual DMA**, and **various DSP operations requiring a multiply and accumulate function**.
- Two full-duplex serial communication controllers support the following protocols:
  - IEEE802.3/Ethernet
  - High level/synchronous data link control
  - LocalTalk (HDLC-based local area network protocol)
  - Universal asynchronous receiver transmitter
  - Infrared protocol
  - Synchronous UART (1x clock mode)
  - Binary synchronous communication
  - Totally transparent operation
- Two full-duplex serial management controllers support the following protocols:
  - GCI (ISDN Interface) monitor and C/I channels
  - UART
  - Transparent operation

- Serial peripheral interface support for master or slave
- **I$^2$C bus controller**
- Time-slot assigner supports multiplexing of data from any of the two serial communication controllers and two serial management controller onto two time-division multiplexed interfaces. The time-slot assigner supports the following time-division multiplexed formats:
  - — T1 / CEPT lines
  - — Pulse code modulation highway interface
  - — ISDN primary rate
  - — Motorola interchip digital link
  - — General circuit interface
  - — User-defined interfaces
- Four independent baud rate generators
- Four general purpose 16-bit timers or two 32-bit timers
- General-purpose parallel ports
  - — Twelve parallel I/O lines with interrupt capability
  - — Parallel interface port controlled by the RISC implements parallel protocols such as Centronics
- CPM interrupt controller

Figure 16-1 illustrates the MPC821 CPM block diagram.

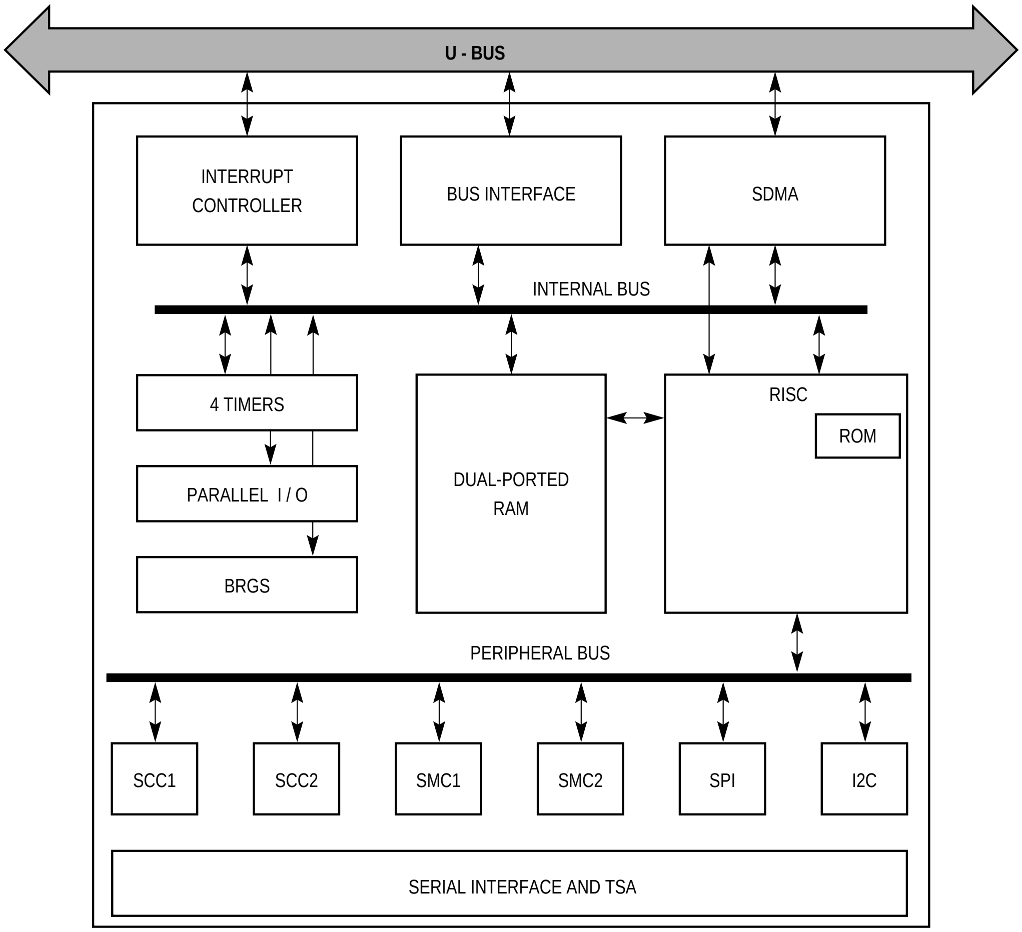


**Figure 16-1. CPM Block Diagram**

## 16.3 SERIAL CONFIGURATIONS

The MPC821 offers an extremely flexible set of communication capabilities. The following figures represent only a subset of all the possible configurations the MPC821 can be used in. Figure 16-2 illustrates a typical configuration for a personal digital assistant (PDA) application that supports various communication links and protocols.

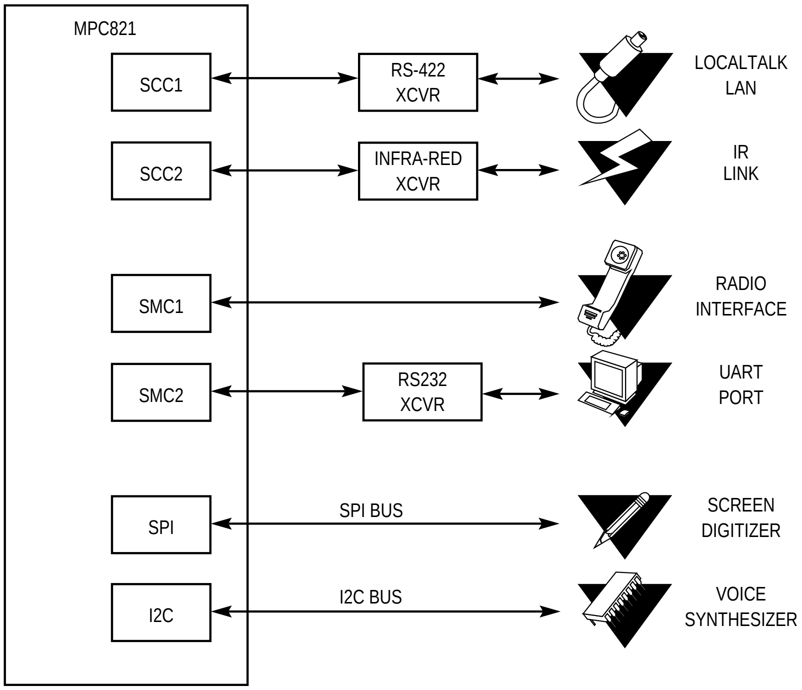


**Figure 16-2. PDA Application Example**

## 16.4 RISC MICROCONTROLLER

### 16.4.1 Overview

The RISC microcontroller is a 32-bit controller for the CPM, residing on a separate bus from the CPU and, therefore, does not impact the performance of the PowerPC CPU. The RISC microcontroller works with the serial channels and parallel port to implement the user-programmable protocols and manage the SDMA channels that transfer data between the I/O channels and memory. The RISC microcontroller architecture and instruction set are optimized and tuned for data communication tasks and data processing functions that are required by many wire-line and wireless communication standards. **A multiply and accumulate (MAC) function composed of a 16-bit x 16-bit multiplier with two 40-bit accumulators enables implementation of many DSP applications.**

The 32-bit RISC microcontroller handles the lower layer tasks and DMA control activities, leaving the PowerPC CPU free to handle higher layer activities. The RISC controller is the 32-bit central controller of the CPM. Since it's execution occurs on a separate bus hidden from the user, it does not impact the CPU core performance. The RISC controller works with the serial channels and parallel interface port to implement the user-chosen protocols and manage the SDMA channels that transfer data between the SCCs and memory.

The RISC controller contains an internal timer that can be used to implement up to 16 additional timers for the user application software. These features are collectively known as the communication processor (CP), which is a subset of the overall CPM. Additionally, the RISC controller manages the operation of the IDMA channels. The 32-bit RISC handles the lower layer tasks and DMA control activities leaving the PowerPC core free to handle higher layer activities.

### 16.4.2 Features

The following is a list of the RISC microcontroller's important features.

- One system clock cycle per instruction
- Fixed-length instruction object code
- Executes code from internal ROM or dual-ported RAM
- 32-bit data path
- Optimized for communication processing
- Digital signal processing capability by means of MAC arithmetic and special addressing modes
- Performs DMA bursting of serial data to external memory

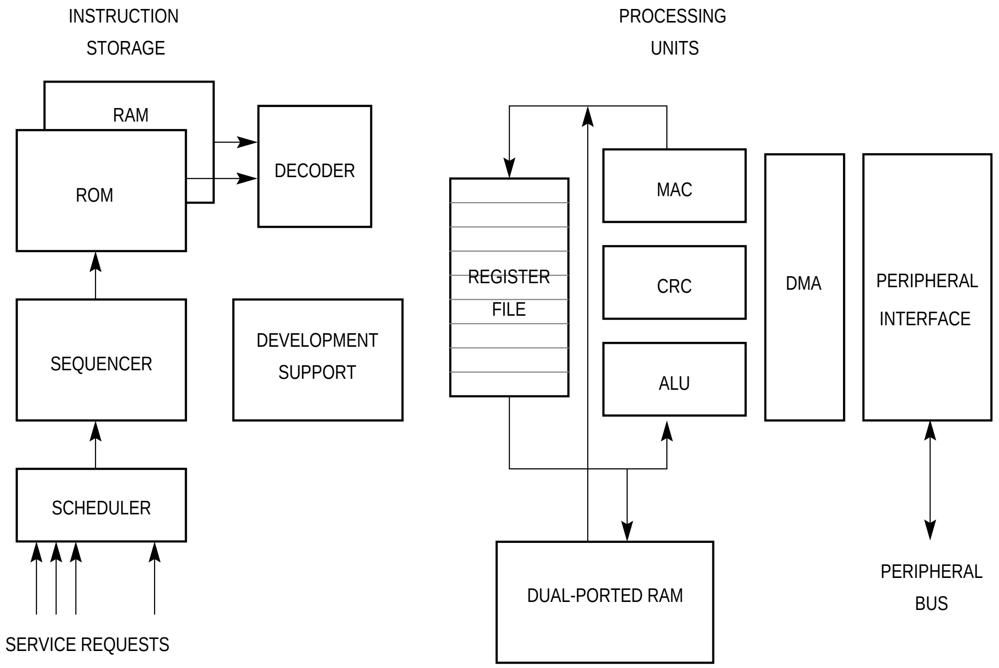


**Figure 16-3. RISC Microcontroller Block Diagram**

### 16.4.3 CPU Interface

The RISC microcontroller communicates with the PowerPC CPU in several ways:

- Many parameters are exchanged through the dual-port RAM. In the case of simultaneous accesses, the microcontroller can be delayed by one clock in its access to the dual-port RAM, but the host is never delayed.
- The RISC microcontroller can execute special commands issued by the host. These commands should only be issued in special situations like exceptions or error recovery.
- The RISC microcontroller generates interrupts through the CPM interrupt controller.
- The RISC CPM status/event registers can read at any time by the PowerPC CPU.

### 16.4.4 Peripheral Interface

The RISC microcontroller uses the peripheral bus to communicate with all of it's peripherals. Each serial communication controller (SCC) has a separate receive and transmit FIFO. The SCC1 FIFOs are 32 bytes each and those remaining are 16-bytes. The SMC, SPI, and I$^2$C FIFO sizes are double-buffered. The parallel interface port (PIP) is a single register interface. The following priority scheme determines the processing priority of the RISC microcontroller.

- Reset in CP command register or system reset
- SDMA bus error
- Commands issued to the command register, including DSP related commands
- IDMA emulation (default setting)
- SCC1 RX
- SCC1 TX
- SCC2 RX
- SCC2 TX
- IDMA emulation (option 2)
- SMC1 RX
- SMC1 TX
- SMC2 RX
- SMC2 TX
- SPI RX
- SPI TX
- I$^2$C RX
- I$^2$C TX
- PIP
- RISC timer tables
- IDMA emulation (option 3)

The RISC microcontroller can control sets of up to 16 timers. These timers are separate and distinct from the four general-purpose timers and baud rate generators in the CPM. They are ideal for protocols that do not require extreme precision, but in those in which it is preferable to free the host CPU from scanning the software's timer tables. These timers are clocked by an internal timer that only the RISC microcontroller uses.

### 16.4.5 Execution From RAM

The RISC microcontroller has an option to execute microcode from a portion of user RAM that is located in the on-chip dual-port RAM. In this mode, either 512 bytes, 1,024 bytes, or 2,048 bytes of the user RAM cannot be accessed by the CPU, but is used exclusively by the RISC microcontroller. In this mode, the RISC microcontroller fetches instructions from both the dual-port RAM and it's own private ROM. This mode allows Motorola to add new protocols or enhancements to the MPC821, in the form of RAM microcodes. If preferred, the user can obtain binary microcode from Motorola and load it into the dual-port RAM.

### 16.4.6 RISC Controller Configuration Register

This 16-bit, memory-mapped, read/write RISC controller configuration register (RCCR) configures the RISC controller to run microcode from ROM or RAM and controls the RISC internal timer. This register is initialized to zero at reset.

RCCR

| BITS | 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 | 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 | 9C4 | | | | | | | | | | | | | | | |

TIME—Timer Enable

This bit enables the RISC controller internal timer that generates a tick to the RISC based on the value programmed into the TIMEP bit. TIME can be modified at any time to start or stop the scanning of the RISC timer tables.

Bit 1—Reserved

TIMEP—Timer Period

This field controls the RISC controller timer tick. The RISC timer tables are scanned on each timer tick and the input to the timer tick generator is the general system clock divided by 1,024. The formula is (TIMEP + 1) × 1,024 = (general system clock period). Thus, a value of 0 stored in these bits gives a timer tick of 1 × (1,024) = 1,024 general system clocks and a value of 63 (decimal) gives a timer tick of 64 × (1,024) = 65,536 general system clocks.

DR1M—IDMA Request 1 Mode

This bit controls the IDMA request 1 sensitivity mode. DREQ1 is used to activate the IDMA channel 1.

0 = IDMA request 1 is edge sensitive.
1 = IDMA request 1 is level sensitive.

DR0M—IDMA Request 0 Mode

This bit controls the IDMA request 0 sensitivity mode. DREQ0 is used to activate the IDMA channel 0.

0 = IDMA request 0 is edge sensitive.
1 = IDMA request 0 is level sensitive.

DRQP—IDMA Request Priority

This bit field controls the priority of the external requests signals relative to the serial channels.

00 = IDMA requests have more priority than the SCCs.
01 = IDMA requests have less priority than the SCCs.
10 = IDMA requests have the lowest priority.
11 = Reserved.

EIE—External Interrupt Enable

When this bit is set, the DREQ0 pin interrupts the RISC controller. The user should configure this bit as instructed during the downloading process of a Motorola-supplied RAM microcode package. Otherwise, it should not be used.

SCD—Scheduler Configuration

The user should configure this bit as instructed during the downloading process of a Motorola-supplied RAM microcode package. Otherwise, it should not be used.

0 = Normal operation.
1 = Alternate configuration of the scheduler.

ERAM—Enable RAM Microcode

The user should configure this bit field as instructed during the downloading process of a Motorola-supplied RAM microcode package. Otherwise, it should not be used.

00 = Disable microcode program execution from the dual ported RAM.
01 = Microcode is executed from the first 512 bytes of the dual-ported RAM.
10 = Microcode is executed from the first 1,024 bytes of the dual-ported RAM.
11 = Microcode is executed from the first 2,048 bytes of the dual-ported RAM.

### 16.4.7 RISC Microcode Revision Number

The RISC controller writes a revision number stored in it's ROM to a dual-port RAM location called REV_num that resides in the miscellaneous parameter RAM. The other locations are reserved for future use.

**Table 16-1. RISC Microcode Revision Number**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| Misc Base + 00 | REV_num | Half-word | Microcode Revision Number |
| Misc Base + 02 | RES | Half-word | Reserved |
| Misc Base + 04 | RES | Word | Reserved |
| Misc Base + 08 | RES | Word | Reserved |

## 16.5 COMMAND SET

The PowerPC CPU issues commands to the RISC by writing to the CPM command register (CPCR). The CPCR rarely needs to be accessed. For example, to terminate the transmission of an SCC's frame without waiting until the end, a STOP TX command must be issued through the command register.

### 16.5.1 CPM Command Register

The host should set the FLG bit in the CPM command register (CPCR) when it issues a command and the CP clears FLG after completing the command, thus indicating to the host that it is ready for the next command. Subsequent commands to the CPCR can only be given after FLG is clear. However, the software reset command issued by setting the RST bit doesn't depend on the state of FLG, but the host should still set FLG when setting RST. The CPCR, a 16-bit, memory-mapped, read/write register, is cleared by reset.

CPCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RST | — | | | OPCODE | | | | CH_NUM | | | | — | | | 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 | | | | | | | | | | | | | | | |

RST—Software Reset Command

This bit is set by the host and cleared by the CP and when this command is executed, the RST and FLG bit are cleared within two general system clocks. The RISC reset routine is approximately 60 clocks long, but the user can begin initialization of the CP immediately after this command is issued. RST is useful when the host wants to reset the registers and parameters for all the channels (SCCs, SMCs, SPI, I$^2$C, and PIP) as well as the RISC processor and RISC timer tables. However, this command does not affect the serial interface (SI) or parallel I/O registers.

Bits 1–3—Reserved

OPCODE—Operation Code
The operation code bits are listed in Table 16-2 below.

**Table 16-2. Opcodes**

| OPCODE | CHANNEL | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | SCC | SMC (UART/TRANS) | SMC (GCI) | SPI | $I^2C$ | IDMA | DSP | TIMER |
| 0000 | INIT RX and TX PARAMS | INIT RX and TX PARAMS | INIT RX and TX PARAMS | INIT RX and TX PARAMS | INIT RX and TX PARAMS | — | — | — |
| 0001 | INIT RX PARAMS | INIT RX PARAMS | — | INIT RX PARAMS | INIT RX PARAMS | — | — | — |
| 0010 | INIT TX PARAMS | INIT TX PARAMS | — | INIT TX PARAMS | INIT TX PARAMS | — | — | — |
| 0011 | ENTER HUNT MODE | ENTER HUNT MODE | — | — | — | — | — | — |
| 0100 | STOP TX | STOP TX | — | — | — | — | — | — |
| 0101 | GRACEFUL STOP TX | — | — | — | — | INIT IDMA | — | — |
| 0110 | RESTART TX | RESTART TX | — | — | — | — | — | — |
| 0111 | CLOSE RX BD | CLOSE RX BD | — | CLOSE RX BD | CLOSE RX BD | — | — | — |
| 1000 | SET GROUP ADDRESS | — | — | — | — | — | — | SET TIMER |
| 1001 | — | — | GCI TIMEOUT | — | — | — | — | — |
| 1010 | RESET BCS | — | GCI ABORT REQUEST | — | — | — | — | — |
| 1011 | — | — | — | — | — | STOP IDMA | — | — |
| 1100 | — | — | — | — | — | — | START DSP | — |
| 1101 | — | — | — | — | — | — | INIT DSP | — |
| 1110 | U | U | U | U | U | U | U | U |
| 1111 | U | U | U | U | U | U | U | U |

INIT TX and RX PARAMS—Initialize Transmit and Receive Parameters

This command initializes the transmit and receive parameters in the parameter RAM to the values that they had after the last reset of the CP. This command is especially useful when switching protocols on a given serial channel.

INIT RX PARAMS—Initialize Receive Parameters

This command initializes the receive parameters of the serial channel.

INIT TX PARAMS—Initialize Transmit Parameters

This command initializes the transmit parameters of the serial channel.

ENTER HUNT MODE—Enter Hunt Mode

This command causes the receiver to stop receiving and begin looking for a new frame. The exact operation of this command may vary depending on the protocol used.

STOP TX—Stop Transmission

This command aborts the transmission from this channel as soon as the transmit FIFO has been emptied. It should be used in cases where transmission needs to be stopped as quickly as possible. Transmission proceeds when the RESTART command is issued.

GRACEFUL STOP TX—Graceful Stop Transmission

This command stops the transmission from this channel as soon as the current frame has been fully transmitted from the transmit FIFO. Transmission proceeds when the RESTART command is issued and the R-bit is set in the next transmit buffer descriptor.

RESTART TX—Restart Transmission

Once the STOP TX command has been issued, this command is used to restart transmission at the current buffer descriptor.

CLOSE RX BD—Close Receive Buffer Descriptor

This command causes the receiver to close the current receive buffer descriptor, making the receive buffer immediately available for manipulation by the user. Reception continues using the next available buffer descriptor. This command can be used to access the data buffer without waiting until the data buffer is completely filled by the SCC.

INIT IDMA—Initialize IDMA

This command initializes the specified IDMA internal RISC state to the value it had at system reset. It is only required when the IDMA autobuffer or buffer chaining modes are used.

SET TIMER—Set Timer

This command activates, deactivates, or reconfigures one of the 16 timers in the RISC timer table.

SET GROUP ADDRESS—Set Group Address

This command sets a bit in the hash table for the Ethernet logical group address recognition function.

GCI ABORT REQUEST—GCI Abort Request

The GCI receiver sends an abort request on the E-bit.

GCI TIMEOUT—GCI Time-Out

The GCI performs the timeout function.

RESET BCS—Reset Block Check Sequence

This command is used in BISYNC mode to reset the block check sequence calculation.

U—Undefined

Reserved for use by Motorola-supplied RAM microcodes.

CH_NUM—Channel Number

These bits are set by the host to define 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$^2$C / IDMA1 |
| 0010 | |
| 0011 | |
| 0100 | SCC2 |
| 0101 | SPI / IDMA2 / RISC Timers |
| 0110 | |
| 0111 | |
| 1000 | |
| 1001 | SMC1 / DSP1 (R) |
| 1010 | |
| 1011 | |
| 1100 | |
| 1101 | SMC2 / DSP2 (T) |
| 1110 | |
| 1111 | |

Bits12–14—Reserved

FLG—Command Semaphore Flag

The bit is set by the host and cleared by the CP.

0 = The CP is ready to receive a new command.
1 = The CPCR contains a command that the CP is currently processing. The CP clears this bit at the end of the command execution or after reset.

### 16.5.2 Command Register Examples

To perform a complete reset of the CP, the value $8001 should be written to the CPCR. Following this command, the CPCR returns the value $0000 after two clocks. To execute an ENTER HUNT MODE command to the SCC2, the value $0341 should be written to the CPCR. While the command is executing the CPCR returns the value $0341, and once it is finished executing it returns the value $0340.

### 16.5.3 Command Execution Latency

The worst-case command execution latency is 120 clocks and the typical command execution latency is about 40 clocks.

## 16.6 DUAL-PORT RAM

The CPM has 5,120 bytes of static RAM configured as dual-port memory. A block diagram of the dual-port RAM is illustrated in Figure 16-4 and its memory map is in Figure 16-5.

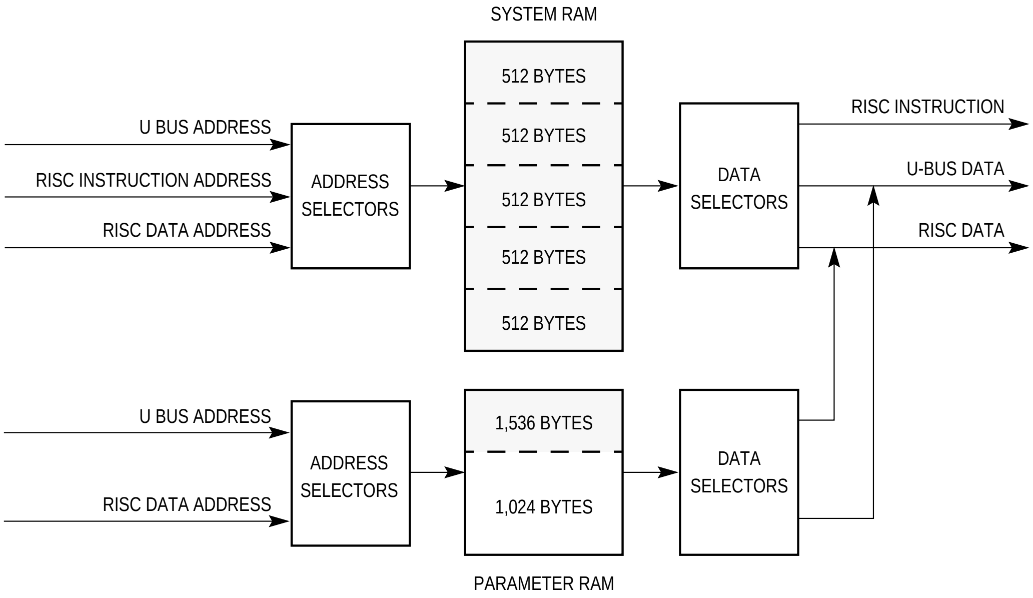


**Figure 16-4. Dual-Port RAM Block Diagram**

The dual-port RAM can be accessed by the RISC microcontroller or one of two bus masters—the PowerPC CPU or SDMA channel. When the dual-port RAM is accessed by the CPU or SDMA channel, it is accessed in two clocks and when it is accessed by the RISC, it is accessed in one clock. In the case of simultaneous access (with at least one write operation), the RISC is delayed by one clock.

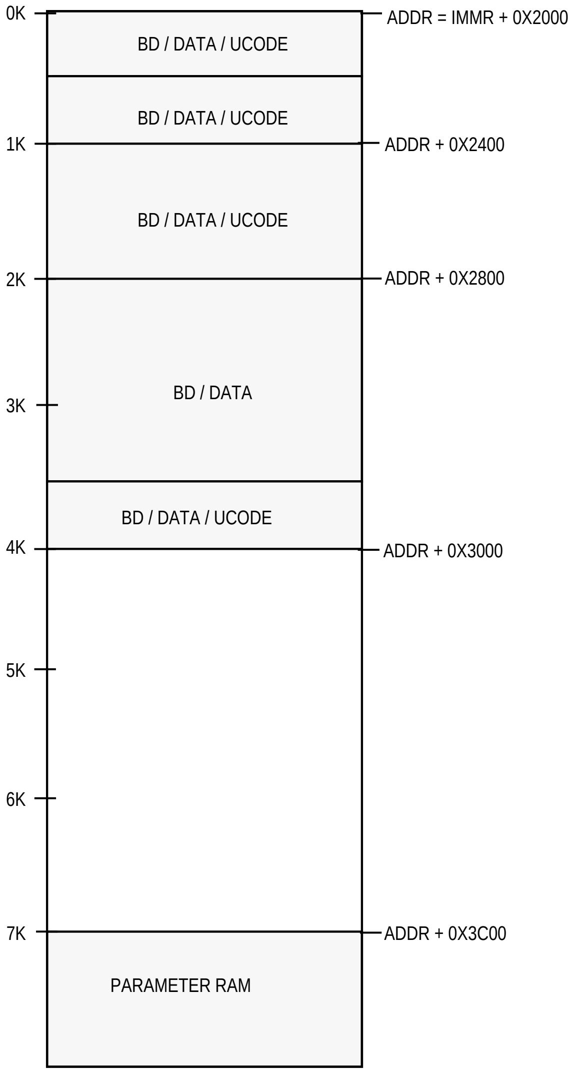


**Figure 16-5. Dual-Port RAM Memory Map**

When the dual-port RAM is accessed by the CPU or SDMA channel, the data and address are taken from the U-Bus and presented on the U-Bus data bus. The RISC has access to the entire dual-port RAM for data fetches and portions of the system RAM for microcode instruction fetches.

The dual-port RAM is used for five possible tasks, of which any two can occur simultaneously.

- To store parameters associated with the SCCs, SMCs, SPI, I$^2$C, and IDMAs in the 1,024-byte parameter RAM.
- To store the buffer descriptors that describe where data is to be received and transmitted from.
- To store data from the serial channels (optional because data can also be stored externally in the system memory).
- To store RAM microcode for the RISC processor. This feature allows Motorola to add protocols in the future.
- For additional scratchpad RAM space for the user program.

Only the parameters in the parameter RAM and the microcode RAM option require fixed addresses to be used. The buffer descriptors, buffer data, and scratchpad RAM can be located in the internal system RAM or in any unused parameter RAM. For instance, in the available area when a serial channel or sub-block is not being used. When a microcode from RAM is executed, certain portions of the system RAM are no longer available. There are three possible configurations for microcode area sizes— first 512-byte block, first two 512-byte blocks, or first four 512-byte blocks. When the first or second 512-byte blocks are used for microcode, the last 256-byte block is also used. When the third or fourth 512-byte blocks are used for microcode the last 512-byte block is used. The 1,536-byte block is always available as system RAM.

## 16.6.1 Buffer Descriptors

The SCCs, SMCs, SPI, and I$^2$C always use buffer descriptors for controlling data buffers and their buffer descriptor formats are all the same (as shown in the chart below).

| | 0 | 15 |
|---|---|---|
| OFFSET + 0 | STATUS AND CONTROL | |
| OFFSET + 2 | DATA LENGTH | |
| OFFSET + 4 | HIGH-ORDER DATA BUFFER POINTER | |
| OFFSET + 6 | LOW-ORDER DATA BUFFER POINTER | |

If the IDMA is used in the buffer chaining or auto-buffer mode, the IDMA channel also uses buffer descriptors and they are described in more detail in **Section 13.11 IDMA Emulation**.

## 16.6.2 Parameter RAM

The CPM maintains a section of dual-port RAM called the parameter RAM which contains many parameters for the operation of the SCCs, SMCs, SPI, I$^2$C, and IDMA channels. An overview of the parameter RAM structure is shown in Table 16-3.

The exact definition of the parameter RAM is contained in each protocol subsection describing a device that uses a parameter RAM. For example, the Ethernet parameter RAM is defined differently in some locations from the HDLC specific parameter RAM.

**Table 16-3. Parameter RAM Overview**

|  | PAGE | ADDRESSES | PERIPHERAL |
|---|---|---|---|
| IMMR + 0x3C00 | 1 | DPRAM_Base+ $1c00 | SCC1 |
|  |  | DPRAM_Base+ $1c7f |  |
|  |  | DPRAM_Base+ $1c80 | I$^2$C |
|  |  | DPRAM_Base+ $1caf |  |
|  |  | DPRAM_Base+ $1cb0 | MISC |
|  |  | DPRAM_Base+ $1cbf |  |
|  |  | DPRAM_Base+ $1cc0 | IDMA1 |
|  |  | DPRAM_Base+ $1cff |  |
| IMMR + 0x3D00 | 2 | DPRAM_Base+ $1d00 | SCC2 |
|  |  | DPRAM_Base+ $1d7f |  |
|  |  | DPRAM_Base+ $1d80 | SPI |
|  |  | DPRAM_Base+ $1daf |  |
|  |  | DPRAM_Base+ $1db0 | Timers |
|  |  | DPRAM_Base+ $1dbf |  |
|  |  | DPRAM_Base+ $1dc0 | IDMA2 |
|  |  | DPRAM_Base+ $1dff |  |
| IMMR + 0x3E00 | 3 | DPRAM_Base+ $1e00 | SCC3 Reserved |
|  |  | DPRAM_Base+ $1e7f |  |
|  |  | DPRAM_Base+ $1e80 | SMC1 |
|  |  | DPRAM_Base+ $1ebf |  |
|  |  | DPRAM_Base+ $1ec0 | DSP1 |
|  |  | DPRAM_Base+ $1eff |  |
| IMMR + 0x3F00 | 4 | DPRAM_Base+ $1f00 | SCC4 Reserved |
|  |  | DPRAM_Base+ $1f7f |  |
|  |  | DPRAM_Base+ $1f80 | SMC2 |
|  |  | DPRAM_Base+ $1fbf |  |
|  |  | DPRAM_Base+ $1fc0 | DSP2 |
|  |  | DPRAM_Base+ $1fff |  |

## 16.7 RISC TIMER TABLES

The RISC controller has the ability to control up to 16 timers that are separate from the four general-purpose timers and baud rate generators in the CPM. These timers are best used in protocols that do not require extreme precision, but in which it is preferable to free the host CPU from scanning the software's timer tables. These timers are clocked from an internal timer that only the RISC microcontroller uses. Each pair of timers can be configured as pulse width modulation (PWM) channels. The output of the channel is driven on one of port B pins and up to eight PWM channels are supported. The following is a list of the RISC timer tables' important features.

- 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 resolution as low as 41 microseconds at 25 MHz
- Maximum timeout period of 172 seconds at 25 MHz
- Continuously updated reference counter

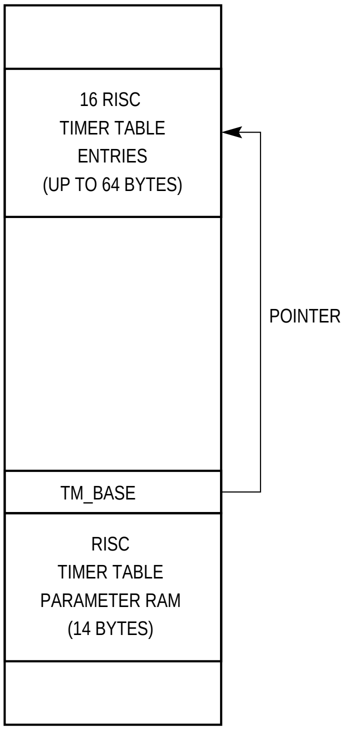
All operations on the RISC timer tables are based on a fundamental "tick" of the RISC internal timer that is programmed in the RCCR. The tick is a multiple of 1,024 general system clocks. Refer to **Section 16.4 RISC Microcontroller** for more details. The RISC timer tables have the lowest priority of all RISC operations. Therefore, if the RISC is so busy with other tasks that it does not have time to service the timer during a tick interval, one or more of the timers may not be updated during a tick. This behavior can be used to estimate the worst-case loading of the RISC processor. See Table 16-4 for more details. The timer tables are configured in the RCCR, the timer table parameter RAM, and by the SET TIMER command issued to the CPCR, timer event register, and timer mask register.

### 16.7.1 RISC Timer Table Parameter RAM

Two areas of internal RAM are used for the RISC timer tables:

- The RISC timer table parameter RAM
- The RISC timer table entries

The RISC timer table parameter RAM area begins at the RISC timer base address and is used for the general timer parameters. See Table 16-4 for details.



### Figure 16-6. RISC Timer Table RAM Usage

### Table 16-4. RISC Timer Table Parameter RAM

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| Timer Base + 00 | **TM_BASE** | Half-word | RISC Timer Table Base Address |
| Timer Base + 02 | TM_ptr | Half-word | RISC Timer Table Pointer |
| Timer Base + 04 | R_TMR | Half-word | RISC Timer Mode Register |
| Timer Base + 06 | R_TMV | Half-word | RISC Timer Valid Register |
| Timer Base + 08 | **TM_cmd** | Word | RISC Timer Command Register |
| Timer Base + 0C | **TM_cnt** | Word | RISC Timer Internal Count |

NOTE: Items in bold must be initialized by the user.

### TM_BASE—RISC Timer Table Base Address

The actual RISC timers are recognized by the user as a small block of memory in the dual-port RAM and TM_BASE is the offset from the beginning of the dual-port RAM where that block resides. The user should allocate 4 bytes at TM_BASE for each timer used (64 bytes at TM_BASE if all 16 timers are used). If less than 16 timers are used, the timers should always be allocated in ascending order to save space. For example, if the user only needs two timers, then 8 bytes are required at location TM_BASE as long as the user only enables RISC timers 0 and 1.

**NOTE**

The TM_BASE should always be aligned to a word boundary (evenly divisible by four).

### TM_ptr—RISC Timer Table Pointer

This value is used exclusively by the RISC to point to the next timer accessed in the timer table. It should not be modified by the user.

### R_TMR—RISC Timer Mode Register

This value is used exclusively by the RISC to store the mode of the timer—one-shot (bit is 0) or restart (bit is 1). R_TMR should not be modified by the user. The SET TIMER command should be used instead.

### R_TMV—RISC Timer Valid Register

This value is used exclusively by the RISC to determine if a timer is currently enabled. If the corresponding timer is enabled, a bit is 1. R_TMV should not be modified by the user. The SET TIMER command should be used instead.

### TM_cmd—RISC Timer Command Register

This value is used as a parameter location when the SET TIMER command is issued. The user should write this location prior to issuing the SET TIMER command. This parameter is defined as follows:

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | V | R | PWM | RESERVED | | | | | | | | | TIMER NUMBER | | | |

| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | TIMER PERIOD | | | | | | | | | | | | | | | |

#### V—Valid

This bit should be set to enable the timer and cleared to disable it.

R—Restart

This bit should be set for an automatic restart or cleared for a one-shot operation of the timer.

PWM—Pulse Width Modulation Mode

This bit should be set to 1 for PWM operation. Refer to **Section 16.7.3 PWM Mode** for details.

Bits 3–11—Reserved

These bits should be written with zeros.

Bits 12–15—Timer Number

The timer number is a value from zero to 15 that signifies timer configuration.

Bits 16–31—Timer Period

The timer period is the 16-bit timeout value of the timer. The maximum value is 65,536 and is programmed by writing $0000 to the timer period.

TM_cnt—RISC Timer Internal Count

This value is a tick counter that the RISC updates after each tick. The update occurs after the RISC scans the timer table. It is updated if the RISC 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 RISC receives and responds to.

### 16.7.2 RISC Timer Table Entries

The 16 timers are located in the block of memory following the TM_BASE location and each timer occupies 4 bytes. The first half-word forms the initial value of the timer written during the execution of the SET TIMER command and the next half-word is the current value of the timer that is decremented until it reaches zero. These locations should not be modified by the user. They are documented only as a debugging aid for user code.

### 16.7.3 PWM Mode

Each pair of timers can be used to generate a PWM waveform on one of port B pins and a maximum of eight channels is supported. The first timer (even numbered) is used to control the duty-cycle time of the waveform. The TIMER PERIOD should be set to the HIGH period of the waveform and the PWM and V bit should be set to 1. The second timer (odd numbered) is used to control the cycle time. The TIMER PERIOD should be set to the preferred cycle time, the PWM bit should be set to zero, and the R and V bits should be set to 1. Table 16-5 shows port B pin assignments for the PWM mode. The respective port B pins should be configured as a general-purpose outputs.

Table 16-5. PWM Channels Pin Assignments

| TIMERS | 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] |
| TIMER 12, 13 | PORT B[17] |
| TIMER 14, 15 | PORT B[16] |

### 16.7.4 RISC Timer Event Register

This 16-bit register is used to report events recognized by the 16 timers and to generate interrupts. An interrupt is only generated if the RISC timer table bit is set in the CPM interrupt mask register. The RISC timer event register (RTER) can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. This register is cleared at reset.

RTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | TMR 15 | TMR 14 | TMR 13 | TMR 12 | TMR 11 | TMR 10 | TMR 9 | TMR 8 | TMR 7 | TMR 6 | TMR 5 | TMR 4 | TMR 3 | TMR 2 | TMR 1 | TMR 0 |
| 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 | 9D6 | | | | | | | | | | | | | | | |

### 16.7.5 RISC Timer Mask Register

This 16-bit register is used to enable interrupts that can be generated in the RTER. If a bit is set, it enables the corresponding interrupt in the RTER. If a bit is cleared, the RISC timer mask register (RTMR) masks the corresponding interrupt in the RTER. An interrupt is only generated if the RISC timer table bit is set in the CPM interrupt mask register. This read/write register is cleared at reset.

### 16.7.6 SET TIMER Command

This command is used to enable, disable, and configure the 16 timers in the RISC timer table and it is issued to the CPCR. This means the value $0851 should be written to CPCR. However, before writing this value, the TM_cmd value should be set up by the user. Refer to **Section 16.7.1 RISC Timer Table Parameter RAM** for details.

### 16.7.7 RISC Timer Initialization Sequence

The following sequence of steps initializes the RISC timers:

1. Configure the RCCR to determine the preferred tick interval that will be used for the entire timer table. The TIME bit is normally turned on at this time. However, it can be turned on later if all RISC timers need to be synchronized.
2. Determine the maximum number of timers to be located in the timer table. Configure the TM_BASE in the RISC timer table parameter RAM to point to a location in the dual-port RAM with 4 × N bytes available, where N is the number of timers. If N is less than 16, use timer 0 through timer N–1 to save space.
3. Clear the TM_cnt field in the RISC timer table parameter RAM to show how many ticks have elapsed since the RISC internal timer was enabled. This step is optional.
4. Clear the RISC timer event register, if it is not already cleared. Ones are written to clear this register.
5. Configure the RTMR to enable those timers that should generate interrupts. Ones enable interrupts.
6. Set the RISC timer table bit in the CPM interrupt mask register to generate interrupts to the system. The CPM interrupt controller may require other initialization not mentioned here.
7. Configure the TM_cmd field of the RISC timer table parameter RAM. At this point, determine whether a timer is to be enabled or disabled, one-shot or restart, and what its timeout period should be. If the timer is being disabled, the parameters (other than the timer number) are ignored.
8. Issue the SET TIMER command by writing $0851 to the CPCR.
9. Repeat the preceding two steps for each timer to be enabled or disabled.

### 16.7.8 RISC Timer Initialization Example

The following sequence initializes RISC timer 0 to generate an interrupt approximately every second using a 25-MHz general system clock:

1. Write the TIMEP bits of the RCCR with 111111 to generate the slowest clock. This value generates a tick every 65,536 clocks, which is every 2.6 milliseconds at 25 MHz.
2. Configure the TM_BASE in the RISC timer table parameter RAM to point to a location in the dual-port RAM with 4 bytes available. Assuming the beginning of dual-port RAM is available, write $0000 to TM_BASE.
3. Write $0000 to the TM_cnt field in the RISC timer table parameter RAM to see how many ticks have elapsed since the RISC internal timer was enabled. This step is optional.
4. Write $FFFF to the RTER to clear any previous events.
5. Write $0001 to the RTMR to enable RISC timer 0 to generate an interrupt.
6. Write $00020000 to the CPM interrupt mask register to allow the RISC timers to generate a system interrupt. Initialize the CPM interrupt configuration register.

7. Write $C0000EE6 to the TM_cmd field of the RISC timer table parameter RAM. This enables RISC timer 0 to timeout after 3,814 (decimal) ticks of the timer. The timer automatically restarts after it times out.
8. Write $0851 to the CPCR to issue the SET TIMER command.
9. Set the TIME bit in the RCCR to enable the RISC timer to begin operation.

### 16.7.9 RISC Timer Interrupt Handling

The following sequence describes what would normally occur within an interrupt handler for the RISC timer tables:

1. Once an interrupt occurs, read the RTER to see which timer(s) have caused interrupts. The RISC timer event bits are usually cleared by this time.
2. Issue additional SET TIMER commands at this time or later, as preferred. Nothing needs to be done if the timer is being automatically restarted for a repetitive interrupt.
3. Clear the R-TT bit in the CPM interrupt status register.
4. Execute the RTE instruction.

### 16.7.10 RISC Timer Table Algorithm

The RISC scans the timer table once every tick. For each valid timer in the table, the RISC decrements the count and checks for a timeout. If no timeout occurs, it moves to the next timer and if a timeout does occur, the RISC sets the corresponding event bit in the RISC timer event register. Then it checks to see if the timer is to be restarted and if it does, it leaves the timer valid bit set in the R_TMV location and resets the current count to the initial count. Otherwise, it clears the R_TMV bit. Once the timer table is scanned, the RISC updates the TM_cnt value in the RISC timer table parameter RAM and stops working on the timer tables until the next tick.

If a SET TIMER command is issued, the RISC controller makes the appropriate modifications to the timer table and parameter RAM, but does not scan the timer table until the next tick of the internal timer. It is important to use the SET TIMER command to properly synchronize the timer table alterations to the execution of the RISC.

### 16.7.11 RISC Timer Table Application: Track the RISC Loading

The RISC timers can be used to track RISC controller loading. The following sequence provides a method for using the 16 RISC timers to determine if the RISC controller ever exceeds the 96% utilization level during any tick interval. Removing the timers then adds a 4% margin to the RISC utilization level, but the aggressive user can use this technique to push the RISC performance to it’s limit. The user should use the standard initialization sequence and incorporate the following differences:

1. Program the tick of the RISC timers to be 1,024 x 16 = 16,384.
2. Disable RISC timer interrupts, if preferred.
3. Using the SET TIMER command, initialize all 16 RISC timers to have a timer period of $0000, which equates 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 should 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 and if it is more than two ticks different from the general-purpose timer, the RISC controller has, during some tick interval, exceeded the 96% utilization level.

**NOTE**

> The general-purpose timers are up-counters, but the RISC timers are down-counters. The user should take this under consideration when comparing timer counts.

## 16.8 DIGITAL SIGNAL PROCESSING CAPABILITIES

Many embedded control applications require implementation of DSP-style algorithms, such as finite impulse response (FIR) filters with or without adaptive equalization, data compression, and scrambling. Usually, to support such applications, a separate DSP processor must be included in the system and this complicates it and causes the cost and power consumption to increase.

The CPM of the MPC821 provides the additional horsepower needed for those applications and thus, eliminates the need for an additional processor. The RISC microcontroller instruction set supports high-performance multiply and accumulate (MAC) operation as well as special addressing modes that are important for efficient implementation of the DSP algorithm. The RISC microcontroller runs concurrently with the PowerPC CPU, and increases the CPU bandwidth left for other system tasks. The system can take advantage of this increased CPU bandwidth by lowering the system clock frequency (and voltage), thus decreasing power consumption.

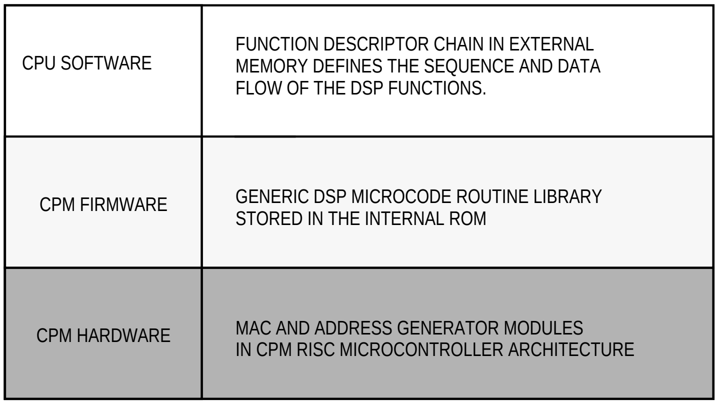
### 16.8.1 Features

The following is a list of the important features of DSP.

- 16-bit x 16-bit multiply and accumulate
  - — Two 40-bit accumulators with overflow saturation logic
  - — Two 32-bit input registers
  - — 1 MAC operation per clock (2 clocks latency, 1 clock blockage)
  - — A single instruction triggers a sequence of one, two, or four MACs
  - — Concurrent operation with other instructions
  - — Complex (16-bit real, 16-bit image) FIR loop: 4 clocks per 4 multiplies
- Load/store with automatic post increment/decrement
  - — Post increment/decrement by 0, 1, 2, 4
  - — Modulo and modifier for cyclic buffer support
- DSP routines library provides 11 basic building blocks for implementation of V.32bis and V.34

## 16.8.2 DSP Functionality Overview

Three layers implement the DSP functionality—hardware (H/W), firmware (F/W), and software (S/W). The user only needs to construct the software layer to generate the application.



| | |
|---|---|
| CPU SOFTWARE | FUNCTION DESCRIPTOR CHAIN IN EXTERNAL MEMORY DEFINES THE SEQUENCE AND DATA FLOW OF THE DSP FUNCTIONS. |
| CPM FIRMWARE | GENERIC DSP MICROCODE ROUTINE LIBRARY STORED IN THE INTERNAL ROM |
| CPM HARDWARE | MAC AND ADDRESS GENERATOR MODULES IN CPM RISC MICROCONTROLLER ARCHITECTURE |

**Figure 16-7. DSP Functionality Implementation**

**16.8.2.1 THE HARDWARE.** The CPM microcontroller H/W contains special DSP processing units such as a multiplier and accumulator (a MAC device) that is capable of handling real/complex numbers and an address generator for accessing cyclic buffer structures in the dual-ported RAM.

**16.8.2.2 THE FIRMWARE.** A set of DSP functions has been compiled to form a library of basic building blocks and each function is implemented by a microcode routine stored in the internal ROM. A software interface is defined to enable passing of parameters between the CPU and the CPM (pointer to filter coefficients, pointer to data buffer, and result buffer pointer). Several functions can be chained to reduce software intervention and interrupts rate. The assumption is that all data structures reside in the dual-ported RAM (data samples and coefficients). Table 16-6 lists the DSP functions included in the library.

Table 16-6. DSP Functions

| FUNCTION | INPUT | COEFFICIENT | OUTPUT | APPLICATION |
|---|---|---|---|---|
| FIR1 | Real | Real | Real | Decimation, RX Interpolation |
| FIR2 | Complex | Real | Complex | TX Filter, RX Filter |
| FIR3 | Complex | Complex | Real/Complex | EC Computation, Equalizer |
| FIR5 | Complex | Complex | Real/Complex | Fractionally Spaced Equalizer |
| FIR6 | Real | Complex | Complex | |
| IIR | Real | Real | Real | Biquad Filter |
| MOD | Complex | Complex | Real/Complex | TX Modulation |
| DEMOD | Real | Complex | Complex | RX Demodulation |
| LMS1 | — | — | — | EC Update, Equalizer Update (T/2, T/3) |
| LMS2 | — | — | — | Equalizer Update (2T/3) |
| WADD | Real | — | Real | Interpolation |

### 16.8.3 Programming Model

A function descriptor (FD), similar to the SCC buffer descriptor, is used to specify the DSP function and for passing the parameters associated with the function. A table of such descriptors forms a circular queue with a programmable length. The descriptors are stored in the external memory. There are two FD tables (chains)—one for the transmitter and one for the receiver. The CPU prepares a chain of FDs in the system memory. A special host command directs the CPM microcontroller to execute the chain and a maskable interrupt is then generated after the chain is completed.

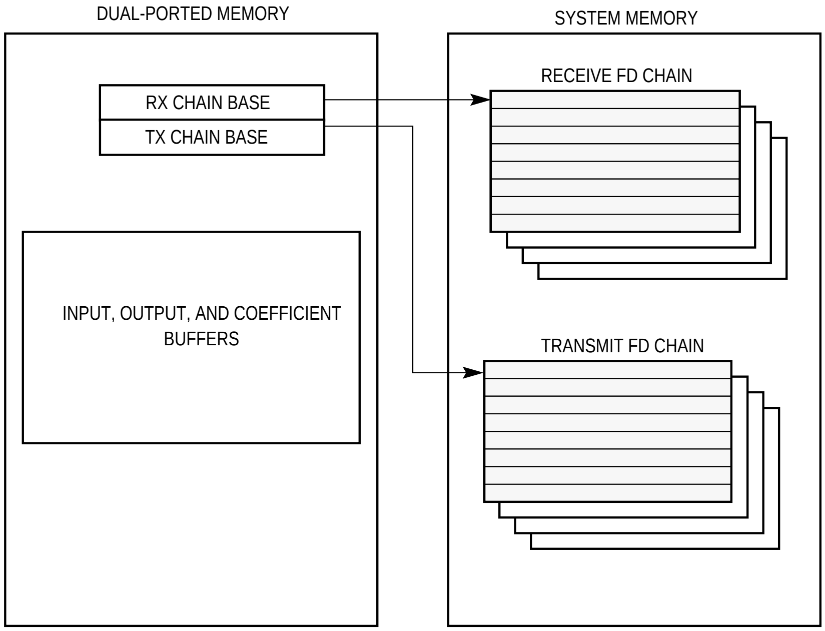


**Figure 16-8. DSP Function Descriptors**

**16.8.3.1 DATA REPRESENTATION.** The inputs, coefficients, and outputs are represented by 16-bit, fixed-point, 2s complement numbers. A real number is represented by a single 16-bit word, as shown in Figure 16-9 below. A complex number is represented by a pair of 16-bit words—one word for the imaginary component and one for the real component. See Figure 16-10 below for details.

| **BITS** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **FIELD** | S | REAL FRACTION | | | | | | | | | | | | | | |

**Figure 16-9. Real Number**

| **BITS** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **FIELD** | S | IMAGINARY FRACTION | | | | | | | | | | | | | | |
| **BITS** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **FIELD** | S | REAL FRACTION | | | | | | | | | | | | | | |

**Figure 16-10. Complex Number**

**16.8.3.2 MODULO ADDRESSING.** The input and output buffers are circular with a programmable size. The lower boundary (base address) of a circular buffer containing M bytes must have zeros in the k LSBs, where $2^k \geq M$, and therefore must be a multiple of $2^k$. The upper boundary is the lower boundary, plus the size minus one (base address + M-1). Once M is chosen, a sequential series of memory blocks (each of length $2^k$) is created where these circular buffer can be located. If $M < 2^k$, there is a $2^k$-M space between the sequential M-sized circular buffers and M should be a multiple of four. See Figure 16-11 below for details.

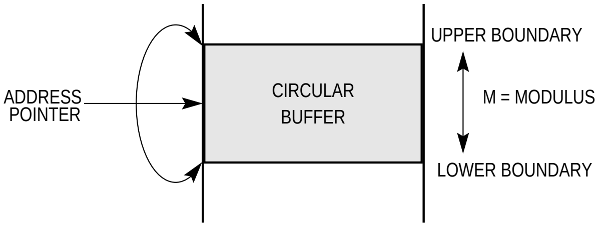


**Figure 16-11. Circular Buffer**

**16.8.3.3 DSP FUNCTION DESCRIPTOR.** Each function descriptor is composed of eight 16-bit words. The first word contains the function opcode as well as status and control bits. The following seven words contain a parameter packet for the function.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | S | — | W | I | — | — | — | — | — | — | — | OPCODE | | | | |
| **OFFSET + 2** | PARAMETER 1 | | | | | | | | | | | | | | | |
| • • • | • • • | | | | | | | | | | | | | | | |
| • • • | • • • | | | | | | | | | | | | | | | |
| • • • | • • • | | | | | | | | | | | | | | | |
| **OFFSET + 14** | PARAMETER 7 | | | | | | | | | | | | | | | |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FD s in this table is programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

Opcode—Function Operation Code

This bit field specifies the function that should be executed (FIR, modulation). Some opcodes are reserved for user expansion by tapping into RAM to execute a user routine.

**16.8.3.4 DSP PARAMETER RAM.** A section in the dual-port RAM is associated with each DSP chain and each section is used for parameter storage or as a scratchpad. See Table 16-7 for DSP parameter RAM memory map details. The FDBASE entry defines the starting location in the system memory for the FD chain and the start address should be 16-byte aligned. The FDBASE should be initialized before issuing the INIT_DSP command.

**Table 16-7. DSP Parameter RAM**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| DSP Base + $00 | **FDBASE** | Word | FD Table Base Address |
| DSP Base + $04 | FD_ptr | Word | FD Pointer |
| DSP Base + $08 | DSTATE | Word | DSP State |
| DSP Base + $10 | DSTATUS | Half-word | Current FD Status |
| DSP Base + $12 | I | Half-word | Current FD Number of Iterations |
| DSP Base + $14 | TAP | Half-word | Current FD Number of TAPs |
| DSP Base + $16 | CBASE | Half-word | Current FD Cbase |
| DSP Base + $18 | | Half-word | Current FD Sample Buffer Size-1 |
| DSP Base + $1A | XPTR | Half-word | Current FD Pointer to Sample Pointer |
| DSP Base + $1C | | Half-word | Current FD Output Buffer Size-1 |
| DSP Base + $1E | YPTR | Half-word | Current FD Pointer to Output Buffer Pointer |
| DSP Base + $20 | M | Half-word | Current FD Sample Buffer Size-1 |
| DSP Base + $22 | | Half-word | Current FD Sample Buffer Pointer |
| DSP Base + $24 | N | Half-word | Current FD Output Buffer Size-1 |
| DSP Base + $26 | | Half-word | Current FD Output Buffer Pointer |
| DSP Base + $28 | K | Half-word | Current FD Coefficient Buffer Size-1 |
| DSP Base + 2A | | Half-word | Current FD Coefficient Buffer Pointer |

NOTE: Items in bold must be initialized by the user.
DSP0 base = IMMR + 0x3EC0 and DSP1 base = IMMR + 0x3FC0.

**16.8.3.5 DSP COMMAND SET.** The following commands are issued to the DSP command register.

INIT DSP CHAIN

This command initializes the corresponding chain to be inactive. The FD pointer is initialized to the starting address of the FD table.

START DSP CHAIN

This command activates the corresponding chain.

**16.8.3.6 DSP EVENT REGISTER.** The DSP functions use the SDMA status register (SDSR) to generate maskable interrupts to the CPU core. An interrupt is set at the completion of the function execution if the I bit is set in the function descriptor. There are two interrupt events, DSP1 and DSP2, and they are each associated with a corresponding chain. The SDSR is a memory-mapped register that can be read at any time. A bit is reset by writing a 1 and is left unchanged by writing a zero and more than one bit can be reset at a time. The register is cleared by reset.

SDSR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | SBER | RINT | — | | | | DSP2 | DSP1 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ADDR | 908 | | | | | | | |

SBER—SDMA Channel Bus Error

This bit indicates that the SDMA channel terminated with an error during a read or write cycle. The SDMA bus error address can be read from the SDAR. SBER is cleared by writing a 1 and writing a zero has no effect.

RINT—Reserved Interrupt

This status bit is reserved for factory testing. RINT is cleared by writing a 1 and writing a zero has no effect.

Bits 2–5—Reserved

DSP1—DSP Chain1 (Receiver) Interrupt

This bit is set on the completion of chain1 function execution, if the I bit is set in the function descriptor.

DSP2—DSP Chain2 (Transmitter) Interrupt

This bit is set on the completion of chain2 function execution, if the I bit is set in the function descriptor.

**16.8.3.7 DSP MASK REGISTER.** The SDMA mask register (SDMR) is used to mask the DSP interrupts and is an 8-bit read/write register with the same bit format as the SDSR. If a bit in the SDMR is a 1, the corresponding interrupt in the status register is enabled and if it is zero, the corresponding interrupt in the status register is masked. This register is cleared on reset.

### 16.8.4 Example of DSP Implementation

Figure 16-12 illustrates two ways to implement a typical DSP task—by running C code on the CPU or by using the CPM functions. The figure below uses a section of the Tx data pump flow of a V.32 modem as an example. The TX filter is composed of three subfilters.

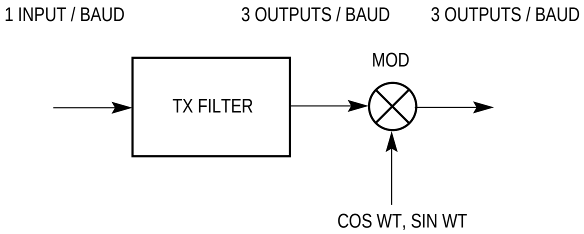


**Figure 16-12. Typical DSP Task Implementation Example**

In the first implementation, it takes 476 CPU instructions (371 for the filter and 105 for the modulation) to execute the code. Repeating that 2,400 times per second (the transmission symbol rate) yields 1.14 MIPS. In the second implementation, the S/W builds a static FD structure composed of two chained functions (a FIR and a MOD). The CPU activates the CPM microcontroller to execute those functions by a single write to the command register. Using an interrupt, the CPM signifies completion of the process. The CPM executes the functions two times more efficiently than the CPU, which results in 0.55 CPM MIPS and very few CPU cycles.

The TX Filter is implemented by executing three subfilters each time a new sample is received. This is accomplished by invoking FIR2 with iteration count set to specify three iterations and auto-increment of the input sample pointer on completion of the function. FIR2 writes the three results into the output buffer, which is also the modulation input buffer. The modulation is accomplished by invoking the MOD function with the iteration count set to specify three iterations. The input pointer is auto-incremented with each iteration.

### 16.8.4.1 CPU ONLY IMPLEMENTATION.

```
void tx_filter ()
{
   S16 *coefr
   S16 *samplr, *sampli
   S16 *coefend;
   S32 filtoutr, filtouti;
   U8 subcount, sampleindex;
   extern S16 mult(S16 p1, S16 p2);     /* in-line invocation */

   coefr=txfiltcoef_str;
   coefend=txfiltcoef_end;
   samplr=&txfiltdly[REAL][txfiltptr];
   sampli=&txfiltdly[IMAG][txfiltptr];
   sampleindex=0;
   while (coefr<coefend) {
      filtoutr=filtouti=0;
      subcount=0;
      while (subcount<TXSUBFILTLEN) {
         filtoutr+=mult(*coefr, *samplr--);
         filtouti+=mult(*coefr++, *sampli--);
      }
      samplr=&txfiltdly[REAL][txfiltptr];
      sampli=&txfiltdly[IMAG][txfiltptr];
      modbuff[REAL][sampleindex]= filtoutr ;
      modbuff[IMAGE][sampleindex++]= filtouti;
   }
}

void modulator ()
{

   U8 i;
   S32 termrnd;
   extern S16 mult(S16 p1, S16 p2);     /* in-line invocation */

   i=0;
   while (i<SAMPLE_PER_T) {
      sigout[i]= mult(sn1800[REAL][cosindx], modbuf[REAL][i]) -
               mult(sn1800[IMAG][cosindx], modbuf[IMAG][i]);
      cosindx++;
      if (cosindx==SIN1800TBL_LEN)cosindx=0;
      i++;
   }

void main ()
{
        *
        *
   tx_filter();
   modulator();
        *
        *
}
```

### 16.8.4.2 CPU+CPM IMPLEMENTATION.

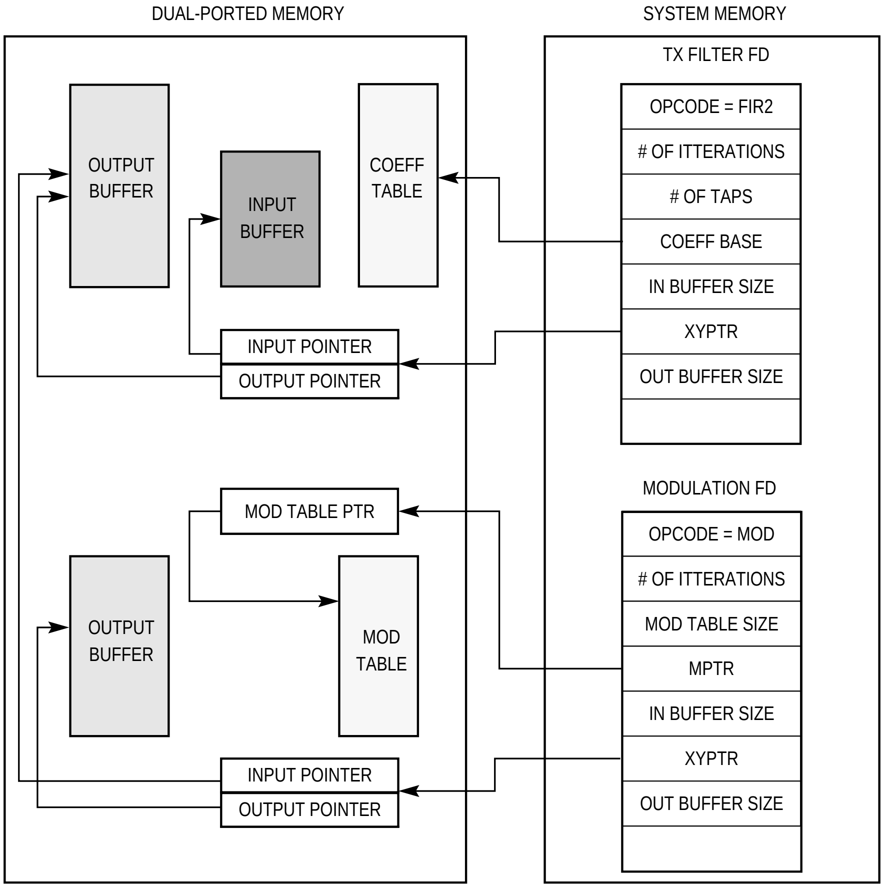


**Figure 16-13. CPU+CPM Implementation**

```
/* Buffer Descriptors */
typedef struct dsp_fd {
        unsigned short  status;
        unsigned short  parameter[7];
} DSP_FD;

#define WRAP    0x2000         /* wrap bit */
#define INTR    0x1000         /* interrupt on completion */


/* define for function opcodes */
#define FIR_2   0x0102  /* FIR2  filter */
#define MOD     0x0008  /* Modulation function opcode */

/* Initialize a static fd table for 2 functions */
DSP_FD  filters[2]= {
```

```
                 { FIR_2,P11,P12, , P17}
                ,{(WRAP | INTR | MOD),P21,P22, , P27}
};

void main()
{
        *
        *
        *
        /* issue command to CP to start processing the fd chain */
        issue_command( START_FD );
        *
        *
        *
}
```

## 16.8.5 FIR1–Real C, Real X, and Real Y

**16.8.5.1 DESCRIPTION.** The FIR1 implements a basic FIR filter with k real coefficients, real input samples, and real output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.

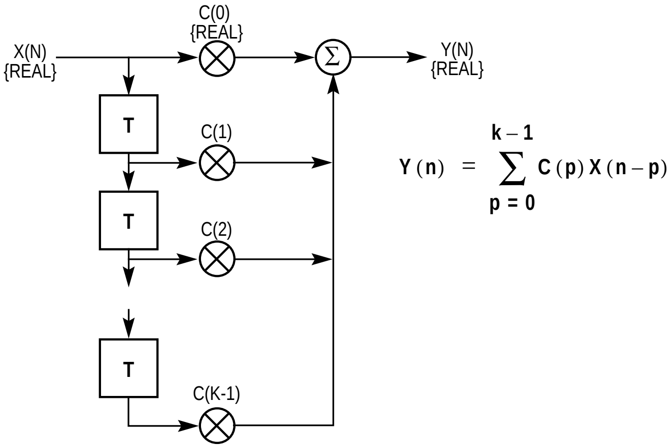


**Figure 16-14. FIR1 Implementation Example**

**16.8.5.2 COEFFICIENTS AND SAMPLE DATA BUFFERS.** The coefficients vector occupies k 16-bit words in memory and C(0) is stored in the first location. The samples input buffer is a cyclic buffer containing M+1 bytes. Each sample is a 16-bit word and the new sample is stored in the address following the previous sample. The output buffer is a cyclic buffer containing N+1 bytes. Each output is a 16-bit word and the new output is stored in the address following the previous output.

| COEFFICIENTS | INPUT SAMPLES | OUTPUT |
|---|---|---|
| C(0) | * | * |
| C(1) | * | * |
| C(2) | x(n-k+1) | * |
|  | * | * |
|  | * | Y(n-k+1) |
| C(k-1) | x(n-2) | * |
|  | x(n-1) | * |
|  | x(n) | Y(n-2) |
|  |  | Y(n-1) |
|  |  | Y(n) |

**Figure 16-15. FIR1 Coefficients and Sample Data Buffers**

**16.8.5.3 FIR1 FUNCTION DESCRIPTOR.** The FIR1 FD bit table is described below.

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | — | IALL | INDEX |  | PC | — | — | 00001 |  |  |  |  |
| OFFSET + 2 | I |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 4 | K |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 6 | CBASE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 8 | M |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + A | XYPTR |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + C | N |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + E | RESERVED |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function has been processed.
1 = A maskable interrupt is generated after this function has been processed.

IALL— Auto Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field only after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The Coefficients pointer is not preset after each iteration.
1 = The Coefficients pointer is preset after each iteration to CBASE.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.5.4 FIR1 PARAMETER PACKET.** The FIR1 parameter packet is composed of seven 16-bit words and is described in the following table.

**Table 16-8. FIR1 Parameter Packet**

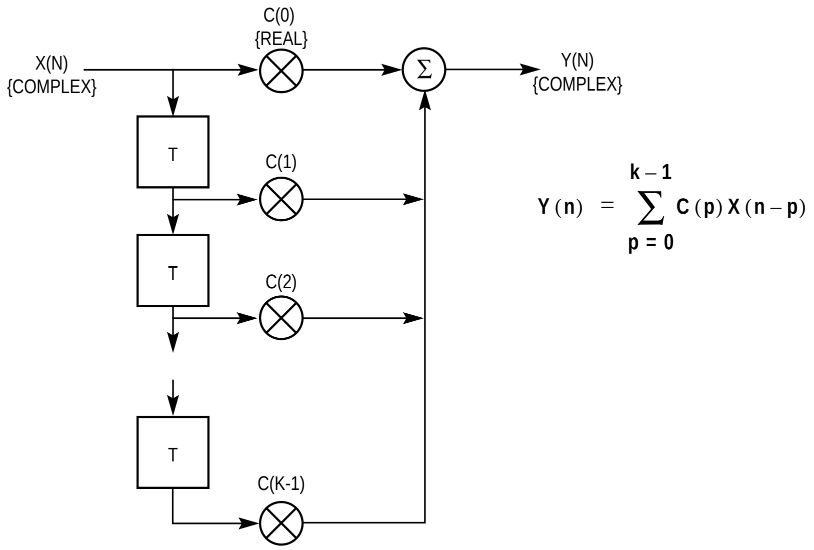
| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | I | Number of Iterations. |
| Word 2 | K | Number of TAPs-1. The number of taps should be a multiple of 4. |
| Word 3 | CBASE | Filter Coefficients Vector Base Address. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 8 (4 samples). |
| Word 5 | XYPTR | Pointer to a structure composed of the input sample data pointer and the output buffer pointer. |
| Word 6 | N | Output Buffer Size-1. The minimum output buffer size is 4 (2 outputs). |
| Word 7 | RES | Reserved |

**16.8.5.5 APPLICATIONS.** The FIR1 is used in the decimation and RX interpolation. For example, the following FD structure can be used to implement a 2:1 decimation:

| | S | — | W | I | — | IALL | INDX | PC | — | — | OPCODE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | S | 0 | W | I | 0 | 1 | 10 | 1 | 0 | 0 | 00001 |
| **OFFSET + 2** | I=3 (THREE ITERATIONS) | | | | | | | | | | |

## 16.8.6 FIR2–Real C, Complex X, and Complex Y

**16.8.6.1 DESCRIPTION.** The FIR2 implements a basic FIR filter with k real coefficients, complex input samples, and complex output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.



$$Y(n) = \sum_{p=0}^{k-1} C(p)X(n-p)$$

**Figure 16-16. FIR2 Implementation Example**

**16.8.6.2 COEFFICIENTS AND SAMPLE DATA BUFFERS.** The coefficients vector occupies k 16-bit words in memory and C(0) is stored in the first location. The samples input buffer is a cyclic buffer containing M+1 bytes and each input sample is two 16-bit words (real and imaginary components). The new sample is stored in the address following the previous sample. The output buffer is a cyclic buffer containing N+1 bytes. Each output is two 16-bit words (real and imaginary components). The new output is stored in the address following the previous output.

| COEFFICIENTS |
|---|
| C(0) |
| C(1) |
| C(2) |
| |
| |
| C(k-1) |

| INPUT SAMPLES |
|---|
| * |
| * |
| image {x(n-k+1)} |
| real {x(n-k+1)} |
| * |
| image {x(n-2)} |
| real{x(n-2)} |
| image{x(n-1)} |
| real{x(n-1)} |
| image{x(n)} |
| real{x(n)} |

| OUTPUT |
|---|
| * |
| * |
| * |
| image{Y(n-k+1)} |
| real{Y(n-k+1)} |
| * |
| * |
| image{Y(n-2)} |
| real{Y(n-2)} |
| image{Y(n-1)} |
| real{Y(n-1)} |
| image{Y(n)} |
| real{Y(n)} |

**Figure 16-17. FIR2 Coefficients and Sample Data Buffers**

**16.8.6.3 FIR2 FUNCTION DESCRIPTOR.** The FIR2 FD bit table is described below.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | — | IALL | INDEX | | PC | — | — | 00010 | | | | |
| OFFSET + 2 | I | | | | | | | | | | | | | | | |
| OFFSET + 4 | K | | | | | | | | | | | | | | | |
| OFFSET + 6 | CBASE | | | | | | | | | | | | | | | |
| OFFSET + 8 | M | | | | | | | | | | | | | | | |
| OFFSET + A | XYPTR | | | | | | | | | | | | | | | |
| OFFSET + C | N | | | | | | | | | | | | | | | |
| OFFSET + E | RESERVED | | | | | | | | | | | | | | | |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function has been processed.
1 = A maskable interrupt is generated after this function has been processed.

IALL— Auto Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field only after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The Coefficients pointer is not preset after each iteration.
1 = The Coefficients pointer is preset after each iteration to CBASE.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.6.4 FIR2 PARAMETER PACKET.** The FIR2 parameter packet is composed of seven 16-bit words and is described in the table below.

**Table 16-9. FIR2 Parameter Packet**

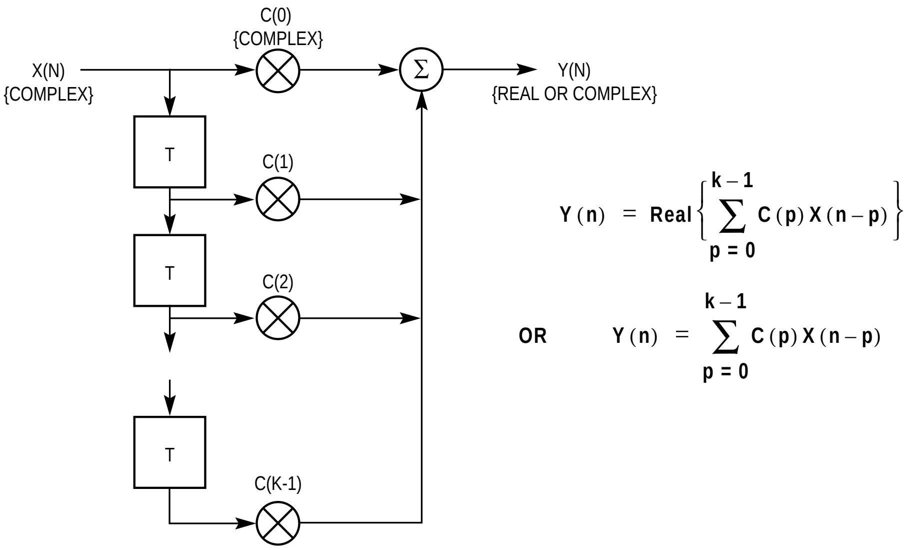
| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | I | Number of Iterations. |
| Word 2 | K | Number of TAPs-1. |
| Word 3 | CBASE | Filter Coefficients Vector Base Address. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 8 (4 samples). |
| Word 5 | XYPTR | Pointer to a structure composed of the input sample data pointer and the output buffer pointer. |
| Word 6 | N | Output Buffer Size-1. The minimum output buffer size is 8 (2 outputs). |
| Word 7 | RES | Reserved |

**16.8.6.5 APPLICATIONS.** The FIR2 is used in the TX and RX filters. For example, the following FD structure can be used to implement the TX filter:

| | S | — | W | I | — | IALL | INDX | PC | — | — | OPCODE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | 0 | W | I | 0 | 0 | 01 | 0 | 0 | 0 | 00010 |
| OFFSET + 2 | I=3 (THREE ITERATIONS) | | | | | | | | | | |

## 16.8.7 FIR3–Complex C, Complex X, and Real/Complex Y

**16.8.7.1 DESCRIPTION.** The FIR3 implements a basic FIR filter with k complex coefficients, complex input samples, and real or complex output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.



$$Y(n) = \text{Real}\left\{ \sum_{p=0}^{k-1} C(p)X(n-p) \right\}$$

$$Y(n) = \sum_{p=0}^{k-1} C(p)X(n-p)$$

**Figure 16-18. FIR2 Implementation Example**

**16.8.7.2 COEFFICIENTS AND SAMPLE DATA BUFFERS.** The coefficients vector occupies k pairs of 16-bit words (real and imaginary components) in memory and C(0) is stored in the first location. The samples input buffer is a cyclic buffer containing M+1 bytes and each input sample is two 16-bit words (real and imaginary components). The new sample is stored in the address following the previous sample. The output buffer is a cyclic buffer containing N+1 bytes and each output is two 16-bit words (real and imaginary components). The new output is stored in the address following the previous output.

| COEFFICIENTS | INPUT SAMPLES | REAL OUTPUT (X=0) | COMPLEX OUTPUT (X=1) |
|---|---|---|---|
| image{C(0)} | * | * | * |
| real{C(0)} | * | * | * |
| image{C(1)} | image {x(n-k+1)} | * | image{Y(n-k+1)} |
| real{C(1)} | real {x(n-k+1)} | * | real{Y(n-k+1)} |
| * | * | Y(n-K+1) | * |
| * | image {x(n-2)} | * | * |
| image{C(k-1)} | real{x(n-2)} | * | image{Y(n-2)} |
| real{C(k-1)} | image{x(n-1)} | Y(n-2) | real{Y(n-2)} |
|  | real{x(n-1)} | Y(n-1) | image{Y(n-1)} |
|  | image{x(n)} | Y(n) | real{Y(n-1)} |
|  | real{x(n)} | * | image{Y(n)} |
|  |  | * | real{Y(n)} |
|  |  | * | * |

**Figure 16-19. FIR3 Coefficients and Sample Data Buffers**

**16.8.7.3 FIR3 FUNCTION DESCRIPTOR.** The FIR3 FD bit table is described below.

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | X | IALL | INDEX |  | PC | — | — | 00011 |  |  |  |  |
| OFFSET + 2 | I |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 4 | K |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 6 | CBASE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 8 | M |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + A | XYPTR |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + C | N |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + E | RESERVED |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

X— Complex Output

0 = Only the real component of the result is written to the output buffer.
1 = The real and the imaginary parts of the result is written to the output buffer.

IALL— Auto Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field only after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The Coefficients pointer is not preset after each iteration.
1 = The Coefficients pointer is preset after each iteration to CBASE.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.7.4 FIR3 PARAMETER PACKET.** The FIR3 parameter packet is composed of seven 16-bit words and is described in the table below.

**Table 16-10. FIR3 Parameter Packet**

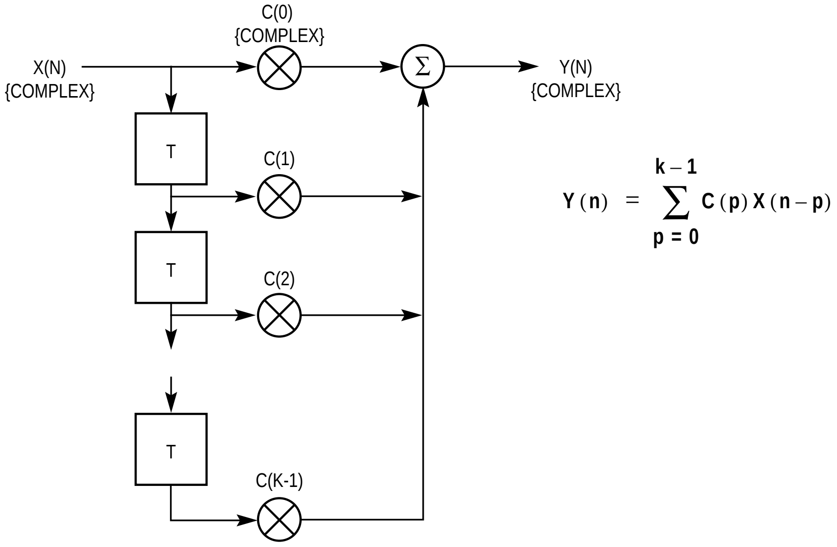
| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | I | Number of Iterations. |
| Word 2 | K | Number of TAPs-1. |
| Word 3 | CBASE | Filter Coefficients Vector Base Address. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples). |
| Word 5 | XYPTR | Pointer to a structure composed of the input sample data pointer and the output buffer pointer. |
| Word 6 | N | Output Buffer Size-1. The minimum output buffer size for x=1 is 8 (2 outputs). The minimum output buffer size for x=0 is 4 (2 outputs). |
| Word 7 | RES | Reserved |

**16.8.7.5 APPLICATIONS.** The FIR3 with the real output is used in the EC computation and the one with the complex output is used in the equalizer.

|  | S | — | W | I | X | IALL | INDX | PC | — | — | OPCODE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | 0 | W | I | 0 | 0 | 01 | 0 | 0 | 0 | 00011 |
| OFFSET + 2 | I=3 (THREE ITERATIONS) | | | | | | | | | | |

## 16.8.8 FIR5–Complex C, Complex X, and Complex Y

**16.8.8.1 DESCRIPTION.** The FIR5 implements a basic FIR filter with k complex coefficients, complex input samples, and complex output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1. The FIR5 only uses other input data samples to implement a fractionally spaced equalizer.



$$Y(n) = \sum_{p=0}^{k-1} C(p)X(n-p)$$

**Figure 16-20. FIR5 Implementation Example**

**16.8.8.2 COEFFICIENTS AND SAMPLE DATA BUFFERS.** The coefficients vector occupies k pairs of 16-bit words (real and imaginary components) in memory and C(0) is stored in the first location. The samples input buffer is a cyclic buffer containing M+1 bytes. Each input sample is two 16-bit words (real and imaginary components) and the new sample is stored in the address following the previous sample. The output buffer is a cyclic buffer containing N+1 bytes and the new output is stored in the address following the previous output.

| COEFFICIENTS | INPUT SAMPLES | REAL OUTPUT<br>(X=0) | COMPLEX OUTPUT<br>(X=1) |
|---|---|---|---|
| image{C(0)} | * | * | * |
| real{C(0)} | * | * | * |
| image{C(1)} | image {x(n-k+1)} | * | image{Y(n-k+1)} |
| real{C(1)} | real {x(n-k+1)} | * | real{Y(n-k+1)} |
| * | * | Y(n-K+1) | * |
| * | image {x(n-2)} | * | * |
| image{C(k-1)} | real{x(n-2)} | * | image{Y(n-2)} |
| real{C(k-1)} | image{x(n-1)} | Y(n-2) | real{Y(n-2)} |
|  | real{x(n-1)} | Y(n-1) | image{Y(n-1)} |
|  | image{x(n)} | Y(n) | real{Y(n-1)} |
|  | real{x(n)} | * | image{Y(n)} |
|  |  | * | real{Y(n)} |
|  |  | * | * |

**Figure 16-21. FIR5 Coefficients and Sample Data Buffers**

**16.8.8.3 FIR5 FUNCTION DESCRIPTOR.** The FIR5 FD bit table is described below.

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | X | IALL | INDEX |  | PC | — | — | 00011 |  |  |  |  |
| OFFSET + 2 | I |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 4 | K |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 6 | CBASE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 8 | M |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + A | XYPTR |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + C | N |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + E | RESERVED |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

X— Complex Output

0 = Only the real component of the result is written to the output buffer.
1 = The real and the imaginary parts of the result is written to the output buffer.

IALL— Auto Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field only after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The Coefficients pointer is not preset after each iteration.
1 = The Coefficients pointer is preset after each iteration to CBASE.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.8.4 FIR5 PARAMETER PACKET.** The FIR5 parameter packet is composed of seven 16-bit words and is described in the table below.

**Table 16-11. FIR5 Parameter Packet**

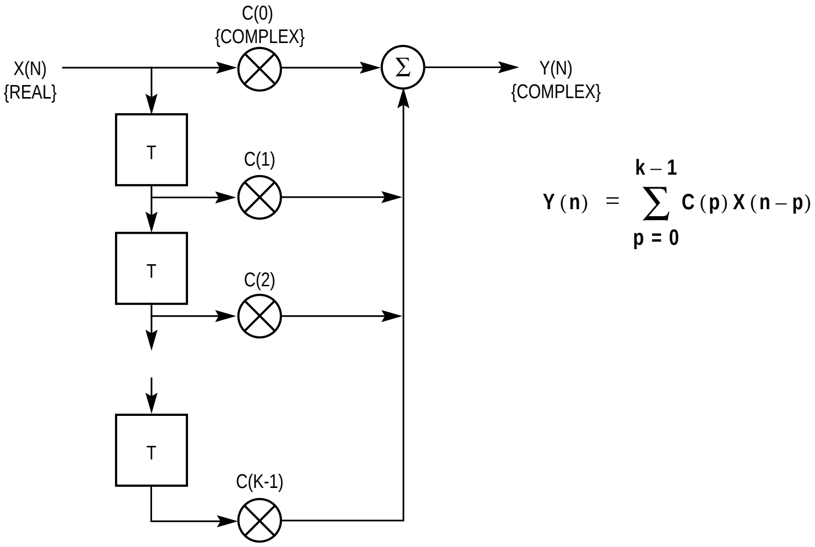
| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | I | Number of Iterations. |
| Word 2 | K | Number of TAPs-1. |
| Word 3 | CBASE | Filter Coefficients Vector Base Address. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples). |
| Word 5 | XYPTR | Pointer to a structure composed of the input sample data pointer and the output buffer pointer. |
| Word 6 | N | Output Buffer Size-1. The minimum output buffer size for x=1 is 8 (2 outputs). The minimum output buffer size for x=0 is 4 (2 outputs). |
| Word 7 | RES | Reserved |

**16.8.8.5 APPLICATIONS.** The FIR5 is used in the fractionally spaced equalizer. The following example demonstrates how the FD structure can be used to implement a fractionally spaced equalizer.

| | S | — | W | I | X | IALL | INDX | PC | — | — | OPCODE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | S | 0 | W | I | 1 | 0 | 11 | 0 | 0 | 0 | 00101 |
| **OFFSET + 2** | I=1 (ONE ITERATION) | | | | | | | | | | |

## 16.8.9 FIR6–Complex C, Real X, and Complex Y

**16.8.9.1 DESCRIPTION.** The FIR6 implements a basic FIR filter with k complex coefficients, real input samples, and complex output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.



$$Y(n) = \sum_{p=0}^{k-1} C(p)X(n-p)$$

Figure 16-22. FIR6 Implementation Example

**16.8.9.2 COEFFICIENTS AND SAMPLE DATA BUFFERS.** The coefficients vector occupies k pairs of 16-bit words (real and imaginary components) in memory and C(0) is stored in the first location. The samples input buffer is a cyclic buffer containing M+1 bytes and each sample is a 16-bit word. The new sample is stored in the address following the previous sample. The output buffer is a cyclic buffer containing N+1 bytes and the new output is stored in the address following the previous output.

| COEFFICIENTS |
|---|
| image{C(0)} |
| real{C(0)} |
| image{C(1)} |
| real{C(1)} |
| * |
| * |
| image{C(k-1)} |
| real{C(k-1)} |

| INPUT SAMPLES |
|---|
| * |
| * |
| x(n-k+1) |
| * |
| * |
| x(n-2) |
| x(n-1) |
| x(n) |
| * |
| * |
| * |

| OUTPUT |
|---|
| * |
| * |
| * |
| image{Y(n-k+1)} |
| real{Y(n-k+1)} |
| * |
| * |
| image{Y(n-2)} |
| real{Y(n-2)} |
| image{Y(n-1)} |
| real{Y(n-1)} |
| image{Y(n)} |
| real{Y(n)} |

**Figure 16-23. FIR6 Coefficients and Sample Data Buffers**

**16.8.9.3 FIR6 FUNCTION DESCRIPTOR.** The FIR6 FD bit table is described below.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | S | — | W | I | — | IALL | INDEX | | PC | — | — | 00110 | | | | |
| **OFFSET + 2** | I | | | | | | | | | | | | | | | |
| **OFFSET + 4** | K | | | | | | | | | | | | | | | |
| **OFFSET + 6** | CBASE | | | | | | | | | | | | | | | |
| **OFFSET + 8** | M | | | | | | | | | | | | | | | |
| **OFFSET + A** | XYPTR | | | | | | | | | | | | | | | |
| **OFFSET + C** | N | | | | | | | | | | | | | | | |
| **OFFSET + E** | RESERVED | | | | | | | | | | | | | | | |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

X— Complex Output

0 = Only the real component of the result is written to the output buffer.
1 = The real and the imaginary parts of the result is written to the output buffer.

IALL— Auto Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field only after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The Coefficients pointer is not preset after each iteration.
1 = The Coefficients pointer is preset after each iteration to CBASE.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.9.4 FIR6 PARAMETER PACKET.** The FIR6 parameter packet is composed of seven 16-bit words and is described in the table below.

**Table 16-12. FIR6 Parameter Packet**

| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | I | Number of Iterations-1 (0 = one iteration). |
| Word 2 | K | Number of TAPs-1. The number of taps should be a multiple of 2. |
| Word 3 | CBASE | Filter Coefficients Vector Base Address. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 4(2 samples). |
| Word 5 | XYPTR | Pointer to a structure composed of the input sample data pointer and the output buffer pointer. |
| Word 6 | N | Output Buffer Size-1. The minimum output buffer size is 8 (2 outputs). |
| Word 7 | RES | Reserved |

## 16.8.10 IIR–Real C, Real X, Real Y

**16.8.10.1 DESCRIPTION.** The IIR implements a basic BIQUAD IIR filter with six real coefficients, real input samples, and real output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1. Several stages of the BIQUAD filter can be cascaded by specifying iteration count greater than one and concatenating the filter coefficients into one vector.

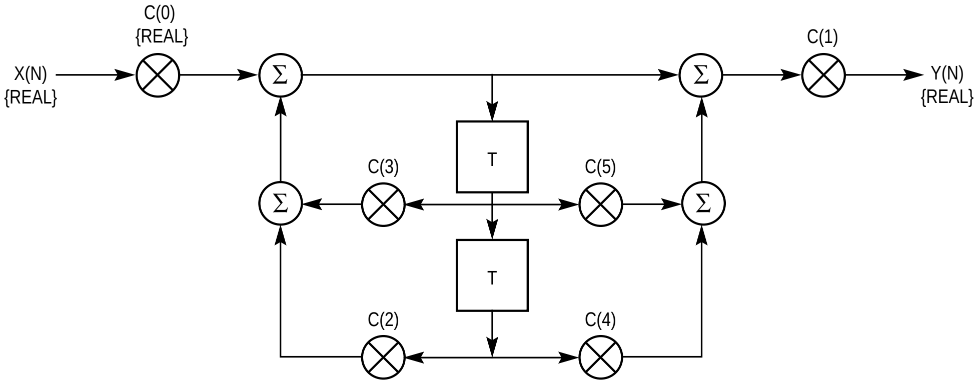


**Figure 16-24. IIR Implementation Example**

**16.8.10.2 COEFFICIENTS AND SAMPLE DATA BUFFERS.** The coefficients vector occupies six 16-bit words in memory and C(0) is stored in the first location. C(1) is only used in the last stage of a cascaded IIR filter. The samples input buffer is a cyclic buffer containing M+1 bytes. Each sample is a 16-bit word and the new sample is stored in the address following the previous sample. The output buffer is a cyclic buffer containing N+1 bytes and the new output is stored in the address following the previous one.

| COEFFICIENTS | INPUT SAMPLES | OUTPUT |
|---|---|---|
| C(0) | * | * |
| C(1) | * | * |
| C(2) | x(n-k+1) | * |
| C(3) | * | * |
| C(4) | * | Y(n-k+1) |
| C(5) | x(n-2) | * |
| | x(n-1) | * |
| | x(n) | Y(n-2) |
| | | Y(n-1) |
| | | Y(n) |
| | | |
| | | |

**Figure 16-25. IIR Coefficients and Sample Data Buffers**

**16.8.10.3 IIR FUNCTION DESCRIPTOR.** The IIR FD bit table is described below.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | — | — | INDEX | | — | — | — | 00111 | | | | |
| OFFSET + 2 | I | | | | | | | | | | | | | | | |
| OFFSET + 4 | TPTR | | | | | | | | | | | | | | | |
| OFFSET + 6 | CBASE | | | | | | | | | | | | | | | |
| OFFSET + 8 | M | | | | | | | | | | | | | | | |
| OFFSET + A | XYPTR | | | | | | | | | | | | | | | |
| OFFSET + C | N | | | | | | | | | | | | | | | |
| OFFSET + E | RESERVED | | | | | | | | | | | | | | | |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

INDEX— Auto Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

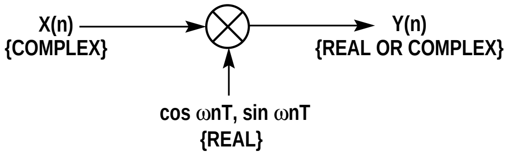
**Table 16-13. IIR Parameter Packet**

| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | I | Number of Iterations (= cascaded stages). |
| Word 2 | TPTR | Pointer to temp delay line(s) pointer. |
| Word 3 | CBASE | Filter Coefficients Vector Base Address. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 4 (2 samples). |
| Word 5 | XYPTR | Pointer to a structure composed of the input sample data pointer and the output buffer pointer. |
| Word 6 | N | Output Buffer Size-1. The minimum output buffer size is 4 (2 outputs). |
| Word 7 | RES | Reserved |

**16.8.10.4 APPLICATIONS.** Among other things, the IIR is used in timing recovery and interpolating filter.

## 16.8.11 MOD–Real Sin, Real Cos, Complex X, and Real/Complex Y

**16.8.11.1 DESCRIPTION.** The MOD implements a basic modulator function with a modulation table composed of {cos ωnT, sin ωnT} pairs, complex input samples, and real output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.



$$\mathrm{REAL}\{Y(n)\} = \mathrm{REAL}\{X(n)\} \times \cos\omega nT - \mathrm{IMAGE}\{X(n)\} \times \sin\omega nT$$

$$\mathrm{IMAGE}\{Y(n)\} = \mathrm{REAL}\{X(n)\} \times \sin\omega nT + \mathrm{IMAGE}\{X(n)\} \times \cos\omega nT$$

**Figure 16-26. MOD Implementation Example**

**16.8.11.2 MODULATION TABLE AND SAMPLE DATA BUFFERS.** The modulation table is composed of 16-bit cosine and sine pairs that occupy K+1 bytes in memory. The samples input buffer is a cyclic buffer containing M+1 bytes. Each sample is a pair of 16-bit words (real and imaginary components) and the new sample is stored in the address following the previous sample. The output buffer is a cyclic buffer containing N+1 bytes and the new output is stored in the address following the previous output. The output buffer can be real or complex, depending on the X bit in the function descriptor.

| MODULATION TABLE |
|---|
| sin $q_1$ |
| cos $q_1$ |
| sin $q_2$ |
| cos $q_2$ |
| * |
| * |
| sin $q_n$ |
| cos $q_n$ |

| INPUT SAMPLES |
|---|
| * |
| * |
| image {x(n-k+1)} |
| real {x(n-k+1)} |
| * |
| image {x(n-2)} |
| real{x(n-2)} |
| image{x(n-1)} |
| real{x(n-1)} |
| image{x(n)} |
| real{x(n)} |

| OUTPUT (REAL) | OUTPUT (COMPLEX) |
|---|---|
| * | * |
| * | * |
| * | * |
| real{Y(n-K+1)} | image{Y(n-k+1)} |
| * | real{Y(n-k+1)} |
| * | * |
| real{Y(n-2)} | * |
| real{Y(n-1) | image{Y(n-2)} |
| real{Y(n)} | real{Y(n-2)} |
| * | image{Y(n-1)} |
| * | real{Y(n-1)} |
| * | image{Y(n)} |
| * | real{Y(n)} |

**Figure 16-27. MOD Table and Sample Data Buffers**

**16.8.11.3 MOD FUNCTION DESCRIPTOR.** The MOD FD bit table is described below.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | X | — | — | — | — | — | — | 01000 | | | | |
| OFFSET + 2 | I | | | | | | | | | | | | | | | |
| OFFSET + 4 | K | | | | | | | | | | | | | | | |
| OFFSET + 6 | MPTR | | | | | | | | | | | | | | | |
| OFFSET + 8 | M | | | | | | | | | | | | | | | |
| OFFSET + A | XYPTR | | | | | | | | | | | | | | | |
| OFFSET + C | N | | | | | | | | | | | | | | | |
| OFFSET + E | RESERVED | | | | | | | | | | | | | | | |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

X— Complex Output

0 = Only the real component of the result is written to the output buffer.
1 = The real and imaginary parts of the result is written to the output buffer.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.11.4 MOD PARAMETER PACKET.** The MOD parameter packet is composed of seven 16-bit words and is described in the table below.

**Table 16-14. MOD Parameter Packet**

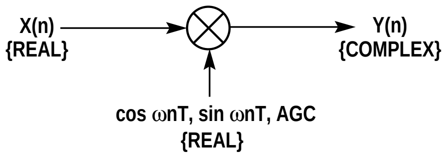
| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | I | Number of Iterations. |
| Word 2 | K | Modulation Table Size-1. The minimum modulation table size is 8 (2 sin/cos pairs). |
| Word 3 | MPTR | Pointer to Modulation Table Pointer. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples). |
| Word 5 | XYPTR | Pointer to a structure composed of the input sample data pointer and the output buffer pointer. |
| Word 6 | N | Output Buffer Size-1. The minimum output buffer size for x=1 is 8 (2 outputs). The minimum output buffer size for x=0 is 4 (2 samples). |
| Word 7 | RES | Reserved |

**16.8.11.5 APPLICATIONS.** The MOD is used in the modulator. The following example demonstrates how the FD structure can be used to implement the MOD function:

| | S | — | W | I | X | — | — | — | — | — | — | OPCODE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | 0 | W | I | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 01000 |
| OFFSET + 2 | I=3 (THREE ITERATIONS) | | | | | | | | | | | |

## 16.8.12 DEMOD–Real Sin; Real Cos, Real X, Complex Y

**16.8.12.1 DESCRIPTION.** The DEMOD implements a basic demodulator function with a modulation table composed of (cos ωnT, sin ωnT) pairs, real input samples, and complex output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1. The AGC parameter controls the demodulator gain.



$$\mathrm{REAL}\{Y(n)\} = (1 + \mathrm{AGC}) \times X(n) \times \cos\omega nT$$

$$\mathrm{IMAGE}\{Y(n)\} = (1 + \mathrm{AGC}) \times X(n) \times (-\sin\omega nT)$$

**Figure 16-28. DEMOD Implementation Example**

### 16.8.12.2 MODULATION TABLE, SAMPLE DATA BUFFERS, AND AGC CONSTANT.

The modulation table is composed of 16-bit cosine and sine pairs that occupy k+1 bytes in memory. The samples input buffer is a cyclic buffer containing M+1 bytes. Each sample is a 16-bit word and the new sample is stored in the next address following the previous sample. The output buffer is a cyclic buffer containing N+1 bytes and the new output is stored in the address following the previous output. The AGC constant is in the range -1≤ AGC≤1.

| MODULATION TABLE |
|---|
| sin $q_1$ |
| cos $q_1$ |
| sin $q_2$ |
| cos $q_2$ |
| * |
| * |
| sin $q_n$ |
| cos $q_n$ |

| INPUT SAMPLES |
|---|
| * |
| * |
| * |
| * |
| x(n-k+1) |
| * |
| * |
| x(n-2) |
| x(n-1) |
| x(n) |
|  |

| OUTPUT (COMPLEX) |
|---|
| * |
| * |
| * |
| image{Y(n-k+1)} |
| real{Y(n-k+1)} |
| * |
| * |
| image{Y(n-2)} |
| real{Y(n-2)} |
| image{Y(n-1)} |
| real{Y(n-1)} |
| image{Y(n)} |
| real{Y(n)} |

**Figure 16-29. DEMOD Modulation Table and Sample Data Buffers**

### 16.8.12.3 DEMOD FUNCTION DESCRIPTOR.

The DEMOD FD bit table is described below.

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | — | — | — | — | — | — | — | 01001 |  |  |  |  |
| OFFSET + 2 | I |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 4 | K |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 6 | DPTR |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 8 | M |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + A | XYPTR |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + C | N |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + E | RESERVED |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.12.4 DEMOD PARAMETER PACKET.** The DEMOD parameter packet is composed of seven 16-bit words and is described in the table below.

**Table 16-15. DEMOD Parameter Packet**

| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | I | Number of Iterations. |
| Word 2 | K | Modulation Table Size-1. The minimum modulation table size is 8 (2 sin/cos pairs). |
| Word 3 | DPTR | Pointer to Modulation Table Pointer and AGC constant. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples). |
| Word 5 | XYPTR | Pointer to a structure composed of the input sample data pointer and the output buffer pointer. |
| Word 6 | N | Output Buffer Size-1. The minimum output buffer size is 8 (2 outputs). |
| Word 7 | RES | Reserved |

**16.8.12.5 APPLICATIONS.** The DEMOD is used in the modulator. The following example demonstrates how the FD structure can be used to implement the MOD function:

| | S | — | W | I | X | — | — | — | — | — | — | OPCODE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | 0 | W | I | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 01001 |
| OFFSET + 2 | I=3 (THREE ITERATIONS) | | | | | | | | | | | |

## 16.8.13 LMS1–Complex Coefficients, Complex Samples, and Real/Complex Scalar

**16.8.13.1 DESCRIPTION.** The LMS1 implements a basic FIR filter coefficients update. The coefficients and input samples are complex numbers, but the scalar is a real or complex number.

$$C^{n+1}{}_{i} = C^{n}{}_{i} + E \times X_{n-i}$$

**Figure 16-30. LMS1 Implementation Example**

**16.8.13.2 COEFFICIENTS AND SAMPLE DATA BUFFERS.** The coefficients vector occupies k pairs of 16-bit words (real and imaginary components) in memory and C(0) is stored in the first location. The samples input buffer is a cyclic buffer containing M+1 bytes. Each sample is a pair of 16-bit words (real and imaginary components) and the new sample is stored in the address following the previous sample.

| COEFFICIENTS | INPUT SAMPLES |
|---|---|
| image{C(0)} | * |
| real{C(0)} | * |
| image{C(1)} | * |
| real{C(1)} | image{X(n-k+1)} |
| * | real{X(n-k+1)} |
| * | * |
| image{C(k-1)} | * |
| real{C(k-1)} | image{X(n-2)} |
|  | real{X(n-2)} |
|  | image{X(n-1)} |
|  | real{X(n-1)} |
|  | image{X(n)} |
|  | real{X(n)} |

**Figure 16-31. LMS1 Coefficients and Sample Data Buffers**

**16.8.13.3 LMS1 FUNCTION DESCRIPTOR.** The LMS1 FD bit table is described below.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | X | — | INDEX | | — | — | — | 01010 | | | | |
| OFFSET + 2 | RESERVED | | | | | | | | | | | | | | | |
| OFFSET + 4 | K | | | | | | | | | | | | | | | |
| OFFSET + 6 | CBASE | | | | | | | | | | | | | | | |
| OFFSET + 8 | M | | | | | | | | | | | | | | | |
| OFFSET + A | XYPTR | | | | | | | | | | | | | | | |
| OFFSET + C | EPTR | | | | | | | | | | | | | | | |
| OFFSET + E | RESERVED | | | | | | | | | | | | | | | |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

X— Complex Scalar

0 = The scalar (E) is a real number.
1 = The scalar (E) is a complex number.

INDEX— Auto Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.13.4 LMS1 PARAMETER PACKET.** The LMS1 parameter packet is composed of seven 16-bit words and is described in the table below.

**Table 16-16. DEMOD Parameter Packet**

| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | RES | Reserved |
| Word 2 | K | Number of Taps-1. |
| Word 3 | CBASE | Filter Coefficients Vector Base Address. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples). |
| Word 5 | XYPTR | Pointer to New Sample Data Pointer. |
| Word 6 | EPTR | Pointer to Scalar. |
| Word 7 | RES | Reserved |

**16.8.13.5 APPLICATIONS.** The LMS1 is used in the EC update.

## 16.8.14 LMS2–Complex Coefficients, Complex Samples, and Real/Complex Scalar

**16.8.14.1 DESCRIPTION.** The LMS2 implements a basic FIR filter coefficients update and the sample pointer is incremented by two (required for fractionally spaced equalizer updates). The coefficients and input samples are complex numbers, but the scalar is a real or complex number.

$$C^{n+1}{}_i = C^{n}{}_i + E \times X_{n-i}$$

**Figure 16-32. LMS2 Implementation Example**

**16.8.14.2 COEFFICIENTS AND SAMPLE DATA BUFFERS.** The coefficients vector occupies k pairs of 16-bit words (real and imaginary components) in memory and C(0) is stored in the first location. The samples input buffer is a cyclic buffer containing M+1 bytes. Each sample is a pair of 16-bit words (real and imaginary components) and the new sample is stored in the address following the previous sample.

| COEFFICIENTS | INPUT SAMPLES |
|---|---|
| image{C(0)} | * |
| real{C(0)} | * |
| image{C(1)} | * |
| real{C(1)} | image{X(n-k+1)} |
| * | real{X(n-k+1)} |
| * | * |
| image{C(k-1)} | * |
| real{C(k-1)} | image{X(n-2)} |
|  | real{X(n-2)} |
|  | image{X(n-1)} |
|  | real{X(n-1)} |
|  | image{X(n)} |
|  | real{X(n)} |

**Figure 16-33. LMS2 Coefficients and Sample Data Buffers**

**16.8.14.3 LMS2 FUNCTION DESCRIPTOR.** The LMS2 FD bit table is described below.

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | X | — | INDEX |  | — | — | — | 01011 |  |  |  |  |
| OFFSET + 2 | RESERVED |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 4 | K |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 6 | CBASE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + 8 | M |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + A | XPTR |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + C | EPTR |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OFFSET + E | RESERVED |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

X— Complex Scalar

0 = The scalar (E) is a real number.
1 = The scalar (E) is a complex number.

INDEX— Auto Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.14.4 LMS2 PARAMETER PACKET.** The LMS2 parameter packet is composed of seven 16-bit words and is described in the table below.

**Table 16-17. LMS2 Parameter Packet**

| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | RES | Reserved |
| Word 2 | K | Number of Taps-1. |
| Word 3 | CBASE | Filter Coefficients Vector Base Address. |
| Word 4 | M | Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples). |
| Word 5 | XPTR | Pointer to New Sample Data Pointer. |
| Word 6 | EPTR | Pointer to Scalar. |
| Word 7 | RES | Reserved |

**16.8.14.5 APPLICATIONS.** The LMS2 is used in the fractionally spaced equalizer coefficient update.

## 16.8.15 Weighted Vector Addition (WADD)–Real X, Real Y

**16.8.15.1 DESCRIPTION.** The weighted vector addition function receives two real vectors and two real coefficients $\alpha$ and $\beta$ as inputs. The function generates an output vector that is the linear combination between the two input vectors according to $\alpha$ and $\beta$. It is a special case when $\beta = 1 - \alpha$ and $(0 \le \alpha \le 1)$ generates a linear interpolation between the two input vectors.

$$Y(n) = \alpha X_1(n) + \beta X_2(n)$$

**Figure 16-34. WADD Implementation Example**

**16.8.15.2 COEFFICIENTS AND SAMPLE DATA BUFFERS.** Each input vector is stored in a cyclic buffer containing M+1 bytes. Each sample is a 16-bit word and the newest sample is stored in the address following the previous sample. The output buffer is a cyclic buffer containing N+1 bytes. Each output is a 16-bit word and the newest output is stored in the address following the previous one.

| $X_1$ INPUT SAMPLES | $X_2$ INPUT SAMPLES | OUTPUT |
|---|---|---|
| * | * | * |
| $x_1$(n-k+1) | * | * |
| * | $x_2$(n-k+1) | * |
| * | * | * |
| $x_1$(n-2) | * | Y(n-k+1) |
| $x_1$(n-1) | $x_2$(n-2) | * |
| $x_1$(n) | $x_2$(n-1) | * |
| * | $x_2$(n) | Y(n-2) |
| * | * | Y(n-1) |
| | | Y(n) |
| | | |
| | | |

**Figure 16-35. WADD Modulation Table and Sample Data Buffers**

**16.8.15.3 WADD FUNCTION DESCRIPTOR.** The WADD FD bit table is described below.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | S | — | W | I | — | — | — | — | — | — | — | 01100 | | | | |
| OFFSET + 2 | I | | | | | | | | | | | | | | | |
| OFFSET + 4 | α | | | | | | | | | | | | | | | |
| OFFSET + 6 | β | | | | | | | | | | | | | | | |
| OFFSET + 8 | M | | | | | | | | | | | | | | | |
| OFFSET + A | XYPTR | | | | | | | | | | | | | | | |
| OFFSET + C | N | | | | | | | | | | | | | | | |
| OFFSET + E | RESERVED | | | | | | | | | | | | | | | |

S—STOP

0 = Do not stop after execution of this FD.
1 = Stop after execution of this FD.

W—Wrap (Final FD in Table)

0 = This is not the last FD in the FD table.
1 = This is the last FD in the FD table. After this buffer has been used, the CP processes the first FD that FDBASE points to in the table. The number of FDs in this table are programmable and determined only by the W-bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

Opcode—Function Operation Code

This bit field specifies the function to be executed.

**16.8.15.4 WADD PARAMETER PACKET.** The WADD parameter packet is composed of seven 16-bit words and is described in the table below.

**Table 16-18. WADD Parameter Packet**

| ADDRESS | NAME | DESCRIPTION |
|---|---|---|
| Word 1 | I | Number of Iterations. |
| Word 2 | α | $X_1$ Weight Coefficient. |
| Word 3 | β | $X_2$ Weight Coefficient. |
| Word 4 | M | Samples Buffer Size-1. |
| Word 5 | XYPTR | Pointer to a structure composed of $X_1$ input sample data pointer, output buffer pointer, and the $X_2$ input sample data pointer. |
| Word 6 | EPTR | Output Buffer Size-1. |
| Word 7 | RES | Reserved |

**16.8.15.5 APPLICATIONS.** By specifying different α and β values, several functions can be realized.

**Table 16-19. WADD Applications**

| α | β | FUNCTION |
|---|---|---|
| $0 \leq \alpha \leq 1$ | 1- α | Linear Interpolation. |
| α | 0 | $y(n)=\alpha x(n)$ Scalar Multiply. |
| 1 | -1 | $y(n)=x_1(n)-x_2(n)$ Vector Subtract. |

### 16.8.16 DSP Execution Times

The execution time of a given function is a linear function of the number of taps and iterations specified for that function. It contains an overview for context switch, handling the FD, and initialization. Table 16-20 below lists the execution time for each of the DSP functions.

Table 16-20. DSP Functions Execution Times

| FUNCTION | EXECUTION TIME<br>(I = NUMBER OF ITERATIONS,<br>K+1 = NUMBER OF TAPS) |
|---|---|
| FIR1 | 53 + 20 · (i-1) + 1.25 · i · (k+1) |
| FIR2 | 47 + 17 · (i-1) + 3 · i · (k+1) |
| FIR3 | 44 + 14 · (i-1) + 4 · i · (k+1) |
| FIR5 | 44 + 14 · (i-1) + 5 · i · (k+1) |
| FIR6 | 50 + 20 · (i-1) + 3 · i · (k+1) |
| IIR | 44 + 11 · i |
| MOD | 44 + 7 · i |
| DEMOD | 47 + 14 · i |
| LMS1 | 42 + 7 · (k+1) |
| LMS2 | 42 + 7 · (k+1) |
| WADD | 46 + 7 · i |
| NOTE: Add 1 clock for Wrap, add 5 clocks for Stop, and add 4 clocks for Interrupt. | |

## 16.9 TIMERS

The CPM includes four identical 16-bit general-purpose timers or two 32-bit timers. Each general-purpose timer consists of a timer mode register, a timer capture register, a timer counter, a timer reference register, a timer event register, and a timer global configuration register. The timer mode register contains the prescaler value programmed by the user. The timer block diagram is illustrated in Figure 16-36.

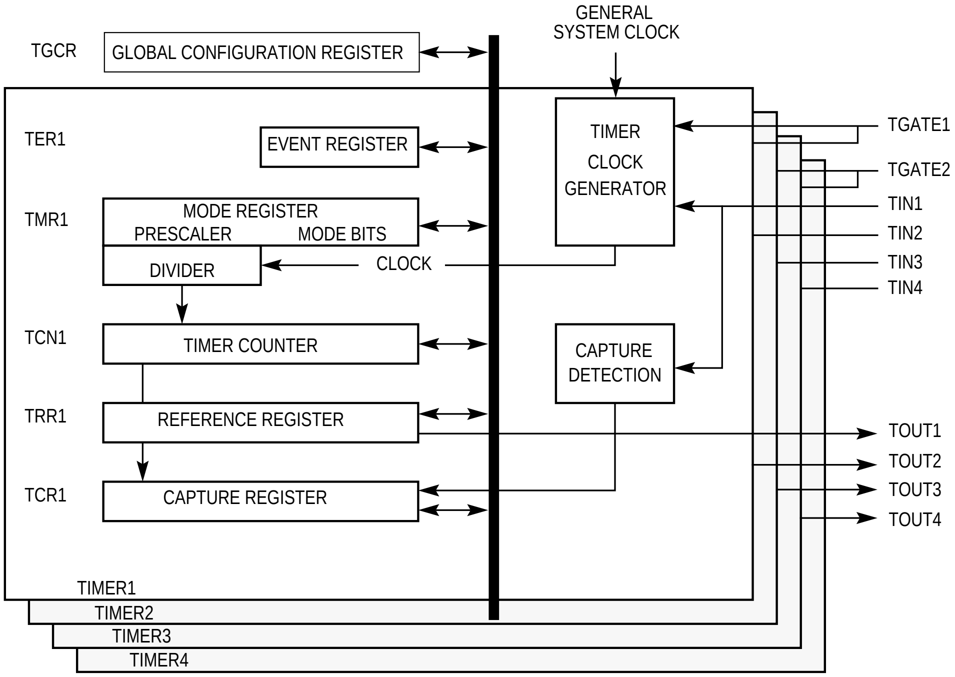


**Figure 16-36. Timer Block Diagram**

## 16.9.1 Features

The following is a list of the timer’s important features:

- Maximum period of 10.7 seconds (at 25 MHz)
- 40-nanosecond resolution (at 25 MHz)
- Programmable sources for the clock input
- Input capture capability
- Output compare with programmable mode for the output pin
- Two timers internally or externally cascadable to form a 32-bit timer
- Free run and restart modes
- Functional compatibility with timers on the MC68360

### 16.9.2 General-Purpose Timer Units

The clock input to the prescaler can be selected from three sources:

- The general system clock
- The general system clock divided by 16
- The corresponding TINx pin

The general system clock is generated in the clock synthesizer and defaults to the system frequency (25 MHz). However, the general system clock has the option to be divided before it leaves the clock synthesizer. This mode, called slow go, is used to save power. Whatever the resulting frequency of the general system clock, the user can either choose that frequency or the frequency divided by 16 as the input to the prescaler of each timer. Alternatively, the user may prefer the TINx pin to be the clock source. TINx is internally synchronized to the internal clock. If the user has chosen to internally cascade two 16-bit timers to a 32-bit timer, then a timer can use the clock generated by the output of another timer.

The clock input source is selected by the ICLK bits of the corresponding TMR. The prescaler is programmed to divide the clock input by values from 1 to 256 and the output of the prescaler is used as an input to the 16-bit counter. The best resolution of the timer is one clock cycle (40 nanoseconds at 25 MHz). The maximum period (when the reference value is all ones) is 268,435,456 cycles (10.7 seconds at 25 MHz). Both values assume that the general system clock is the full 25 MHz.

Each timer can be configured to count until a reference is reached and then either begin a new time count immediately or continue to run. The FRR bit of the corresponding TMR selects each mode. Upon reaching the reference value, the corresponding TER bit is set and an interrupt is issued if the ORI bit in the TMR is set. The timers can output a signal on the timer output pin $\overline{\text{TOUT1}}$, $\overline{\text{TOUT2}}$, $\overline{\text{TOUT3}}$, or $\overline{\text{TOUT4}}$) when the reference value is reached (selected by the OM bit of the corresponding TMR.). This signal can be an active-low pulse or a toggle of the current output. The output can also be internally connected to the input of another timer, resulting in a 32-bit timer.

In addition, each timer has a 16-bit TCR that is used to latch the value of the counter when a defined transition of TIN1, TIN2, TIN3, or TIN4 is sensed by the corresponding input capture edge detector. The type of transition triggering the capture is selected by the CE bits in the corresponding TMR. Upon a capture or reference event, the corresponding TER bit is set and a maskable interrupt request is issued to the CPM interrupt controller. The timers may be gated/restarted by an external gate signal. There are two gate pins—$\overline{\text{TGATE1}}$ controls timer 1 and/or 2 and $\overline{\text{TGATE2}}$ controls timer 3 and/or 4. Normal gate mode enables the count on a falling edge of the $\overline{\text{TGATEx}}$ pin and disables the count on the rising edge of the $\overline{\text{TGATEx}}$ pin. This mode allows the timer to count conditionally, based on the state of the $\overline{\text{TGATEx}}$ pin.

Restart gate mode performs the same function as normal mode, except it also resets the counter on the falling edge of the TGATEx pin. This mode has applications in pulse interval measurement and bus monitoring:

- **Pulse Measurement**—The restart gate mode can measure a low pulse on the TGATEx pin. The rising edge of the TGATEx pin completes the measurement and if TGATEx is externally connected to TINx, it causes the timer to capture the count value and generate a rising-edge interrupt.
- **Bus Monitoring—**The restart gate mode can detect a signal that is abnormally stuck low. The bus signal should be connected to the TGATEx pin. The timer count is reset on the falling edge of the bus signal and if the bus signal does not go high again within the number of user-defined clocks, an interrupt can be generated.

The gate function is enabled in the timer mode register and the gate operating mode is selected in the timer global configuration register.

**NOTE**

> TGATE is internally synchronized to the system clock. If TGATE meets the asynchronous input setup time, then when working with the internal clock the counter begins counting after one system clock.

**16.9.2.1 CASCADED MODE.** In this mode, two 16-bit timers can be internally cascaded to form a 32-bit counter. Timer 1 may be internally cascaded to timer 2 and 3 can be internally cascaded to timer 4. Since the decision to cascade timers is made independently, the user has the option of selecting two 16-bit timers or one 32-bit timer. The timer global configuration register (TGCR) is used to put the timers into cascaded mode. Refer to Figure 16-37 below for details.

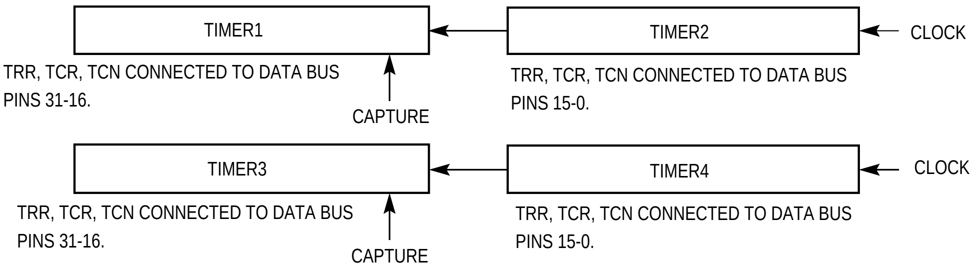

TIMER1
TIMER2
CLOCK
TRR, TCR, TCN CONNECTED TO DATA BUS PINS 31-16.
CAPTURE
TRR, TCR, TCN CONNECTED TO DATA BUS PINS 15-0.
TIMER3
TIMER4
CLOCK
TRR, TCR, TCN CONNECTED TO DATA BUS PINS 31-16.
CAPTURE
TRR, TCR, TCN CONNECTED TO DATA BUS PINS 15-0.

**Figure 16-37. Timer Cascaded Mode Block Diagram**

If the CAS bit is set in the timer global configuration register, the two timers function as a 32-bit timer with a 32-bit timer reference register, timer capture register, and timer counter. In this case, TMR1 and/or TMR3 are ignored, and the modes are defined using TMR2 and/or TMR4. The capture is controlled from TIN2 or TIN4 and the interrupts are generated from TER2 or TER4. When working in the cascaded mode, the cascaded timer reference register, timer capture register, and timer counter should always be referenced with 32-bit bus cycles.

**16.9.2.2 TIMER GLOBAL CONFIGURATION REGISTER.** The timer global configuration register (TGCR) is a 16-bit, memory-mapped, read/write register that contains configuration parameters used by all four timers. It allows simultaneous starting and stopping of any number of timers if one bus cycle is used to access TGCR which is cleared by reset.

TGCR

| BITS | 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 | 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 | 980 | | | | | | | | | | | | | | | |

CAS4—Cascade Timers

0 = Normal operation.
1 = Timers 3 and 4 are cascaded to form a 32-bit timer.

CAS2—Cascade Timers

0 = Normal operation.
1 = Timers 1 and 2 are cascaded to form a 32-bit timer.

FRZ—Freeze

0 = The corresponding timer ignores the FREEZE pin.
1 = Halts the corresponding timer if the FREEZE is asserted. FREEZE is asserted in by the CPU in breakpoint.

STP —Stop Timer

0 = Normal operation.
1 = Reduce power consumption of the timer. This bit stops all clocks to the timer, except the clock from the U-Bus interface, which allows the user to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.

RST—Reset Timer

0 = Reset the corresponding timer (a software reset is identical to an external reset).
1 = Enable the corresponding timer if the STP bit is cleared.

GM2—Gate Mode for Pin 2

This bit is only valid if the gate function is enabled in TMR3 or TMR4.

0 = Restart gate mode. The $\overline{\text{TGATE2}}$ pin is used to enable/disable the count. The falling edge of $\overline{\text{TGATE2}}$ enables and restarts the count and the 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 TCN.

GM1—Gate Mode for Pin 1

This bit is only valid if the gate function is enabled in TMR1 or TMR2.

0 = Restart gate mode. The TGATE1 pin is used to enable/disable count. A falling $\overline{\text{TGATE1}}$ pin enables and restarts the count and a rising edge of $\overline{\text{TGATE1}}$ disables the count.
1 = Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE1}}$ does not restart the count value in TCN.

**16.9.2.3 TIMER MODE REGISTER.** RTMR1 through TMR4 are identical 16-bit, memory-mapped, read/write registers. These registers are cleared by reset.

**NOTE**

The TGCR should be initialized prior to the TMRs or erratic behavior can occur. The only exception is the RST bit in the TGCR, which can be modified at any time.

TMR1-TMR4S

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | PS | | | | | | | | CE | | OM | ORI | FRR | ICLK | | GE |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | 990 (TMR1), 992 (TMR2), 9A0 (TMR3), 9A2 (TMR4) | | | | | | | | | | | | | | | |

PS—Prescaler Value

The prescaler is programmed to divide the clock input by values from 1 to 256. The value 00000000 divides the clock by 1 and 11111111 divides the clock by 256.

CE—Capture Edge and Enable Interrupt

00 = Disable interrupt on capture event; capture function is disabled.
01 = Capture on rising TINx edge only and enable interrupt on capture event.
10 = Capture on falling TINx edge only and enable interrupt on capture event.
11 = Capture on any TINx edge and enable interrupt on capture event.

OM—Output Mode

0 = Active-low pulse on $\overline{\text{TOUTx}}$ for one timer input clock cycle as defined by the ICLK bits. Thus, $\overline{\text{TOUTx}}$ may be low for one general system clock period, one general system clock/16 period, or one TINx pin clock cycle period. $\overline{\text{TOUTx}}$ changes occur on the rising edge of the system clock.
1 = Toggle the $\overline{\text{TOUTx}}$ pin. $\overline{\text{TOUTx}}$ changes occur on the rising edge of the system clock.

ORI—Output Reference Interrupt Enable

0 = Disable interrupt for reference reached (does not affect interrupt on capture function).
1 = Enable interrupt upon reaching the reference value.

FRR—Free Run/Restart

0 = Free run. The timer count continues to increment after the reference value is reached.
1 = Restart. The timer count is reset immediately after the reference value is reached.

ICLK—Input Clock Source for the Timer

00 = Internally cascaded input.
For TMR1, the timer 1 input is the output of timer 2.
For TMR3, the timer 3 input is the output of timer 4.
For TMR2 and TMR4, this selection means no input clock is provided to the timer.
01 = Internal general system clock.
10 = Internal general system clock divided by 16.
11 = Corresponding TIN pin: TIN1, TIN2, TIN3, or TIN4 (falling edge).

GE—Gate Enable

0 = The $\overline{\text{TGATE}}$ signal is ignored.
1 = The $\overline{\text{TGATE}}$ signal is used to control the timer.

**16.9.2.4 TIMER REFERENCE REGISTERS.** Each timer reference register (TRR) is a 16-bit, memory-mapped, read/write register containing the reference value for the timeout. TRR1 through TRR4 are set to all ones by reset. The reference value is not reached until TCN increments to equal TRR.

**16.9.2.5 TIMER CAPTURE REGISTERS.** Each timer capture register (TCR) is a 16-bit register used to latch the value of the counter. TCR1 through TCR4 appear as memory-mapped, read-only registers to the user and are cleared at reset.

**16.9.2.6 TIMER COUNTER.** Each timer counter (TCN) is a 16-bit, memory-mapped, read/write up-counter. A read cycle to TCN1 through TCN4 yields the current value of the timer, but does not affect the counting operation. A write cycle to TCN1 through TCN4 sets the register to the written value, causing its corresponding prescaler to be reset.

**NOTE**

Write operation to this register while the timer is not running may not update the register correctly. The user should always use the timer reference to define the preferred count value.

**16.9.2.7 TIMER EVENT REGISTERS.** Each timer event register (TER) is a 16-bit register used to report events recognized by any of the timers. On recognition of an output reference event, the timer sets the REF bit in the TER, regardless of the corresponding ORI bit in the TMR. The capture event is only set if it is enabled by the CE bits in the TMR. TER1 through TER4, which appear to the user as memory-mapped registers, can be read at any time.

A bit is reset by writing a 1 to that bit (writing a zero does not affect a bit value). More than one bit can be reset at a time. Both bits must be reset before the timer negates the interrupt to the CPM interrupt controller. This register is cleared by reset.

TER1-TER4

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | | | | | | | | | | | | | | REF | CAP |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ADDR | 9B0 (TER1), 9B2 (TER2), 9B4 (TER3), 9B6 (TER4) | | | | | | | | | | | | | | | |

Bits 0–13—Reserved

REF—Output Reference Event
The counter has reached the TRR value. The ORI bit in the TMR is used to enable the interrupt request caused by this event.

CAP—Capture Event
The counter value has been latched into the TCR. The CE bits in the TMR are used to enable generation of this event.

## 16.9.3 Timer Examples

The following is an example of the required initialization sequence of timer 2 to generate an interrupt every 10 microseconds, assuming a general system clock of 25 MHz. This means that an interrupt should be generated every 250 system clocks.

1. TGCR = $0000. Put timer 2 into the reset state. Do not use cascaded mode.
2. TMR2 = $001A. Enable the prescaler of the timer to divide-by-1 and the clock source to general system clock. Enable an interrupt when the reference value is reached, and restart the timer to repeatedly generate 10 microseconds interrupts.
3. TCN2 = $0000. Initialize the timer 2 count to zero (default state of this register).
4. TRR2 = $00FA. Initialize the timer 2 reference value to 250 (decimal).

5. TER2 = $FFFF. Clear TER2 of any bits that might have been set.
6. CIMR = $00040000. Enable the timer 2 interrupt in the CPM interrupt controller and initialize the CICR.
7. TGCR = $0010. Enable timer 2 to begin counting.

To implement the same function with a 32-bit timer using timers 1 and 2, the following sequence can be used:

1. TGCR = $0080. Cascade timers 1 and 2 and put them in the reset state.
2. TMR2 = $001A. Enable the prescaler of timer 2 to divide-by-1 and the clock source to general system clock, enable an interrupt when the reference value is reached, and restart the timer to repeatedly generate 10-microsecond interrupts.
3. TMR1 = $0000. Enable timer 1 to use the output of timer 2 as its input (the default state of this register).
4. TCN1 = $0000, TCN2 = $0000. Initialize the combined timers 1 and 2 count to zero (the default state of this register). This can be accomplished with one 32-bit data move to TCN1.
5. TRR1 = $0000, TRR2 = $00FA. Initialize the combined timers 1 and 2 reference value to 250 (decimal). This can be accomplished with one 32-bit data move to TRR1.
6. TER2 = $FFFF. Clear TER2 of any bits that might have been set.
7. CIMR = $00040000. Enable the timer 2 interrupt in the CPM interrupt controller and initialize the CICR.
8. TGCR = $0091. Enable timers 1 and 2 to begin counting, but leave them in cascaded mode.

## 16.10 SDMA CHANNELS

Two physical serial DMA (SDMA) channels are present on the MPC821. One is controlled by the RISC microcontroller and the other by the LCD controller. The RISC microcontroller implements twelve virtual SDMA channels and each one is associated with a serial channel transmitter or receiver. Four are associated with the two full-duplex SCCs. The other eight are assigned to the service of the SPI, $I^2C$, and the two SMCs. Each channel is permanently assigned to service either the receive or transmit operation of an SCC, SMC, SPI, or $I^2C$. Figure 16-38 illustrates the paths of the data flow. Data from the SCCs, SMCs, SPI, and $I^2C$ may be routed to the external RAM (path 1) or the internal dual-port RAM (path 2). In both cases, however, the U-Bus is used for data transfer.

On a path 1 access, the U-Bus and the external system bus must be acquired by the SDMA channel. On a path 2 access, only the U-Bus needs to be acquired and the access is not seen on the external system bus, unless the MPC821 is configured into the “show cycles” mode of the SIU. Thus, the transfer on the U-Bus occurs at the same time as other operations on the external system bus.

Each SDMA channel can be programmed to output one of eight function codes that are used to identify the channel that is currently accessing memory. The SDMA channel can be assigned a big-endian (Motorola) or little-endian format for accessing buffer data. These features are programmed in the receive and transmit function code registers associated with the SCCs, SMCs, SPI, and I$^2$C. If a bus error occurs on a RISC-related access by the SDMA, the CPM generates a unique interrupt in the RISC status register.

The interrupt service routine then reads the SDMA address register to determine the address the bus error occurred on. The channel that caused the bus error is determined by reading the RX internal data pointer and TX internal data pointers from the specific protocol parameters area in the parameter RAM for the serial channels. If an SDMA bus error occurs on a RISC-related cycle, all CP activity ceases, and the entire CP must be reset in the command register.

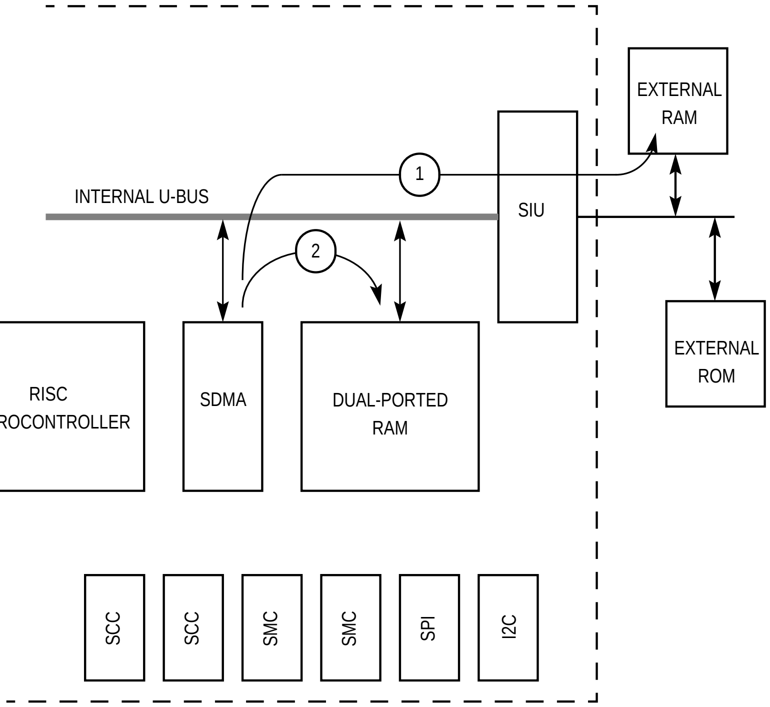


**Figure 16-38. SDMA Data Paths**

### 16.10.1 SDMA Bus Arbitration and Bus Transfers

On the MPC821, the I-cache, D-cache, system interface unit (SIU), and SDMA can become internal bus masters and to determine the relative priority of these masters, each is given an arbitration ID. Only the SDMA arbitration can be adjusted by a user. All other arbitration IDs are fixed. The 12 SDMA channels share the same ID, which is programmed by the user. Therefore, any SDMA channel can arbitrate for the bus against the other internal masters and any external masters that are present. Once an SDMA channel obtains the system bus, it remains the bus master for one transaction (which may be a byte, half-word, word, or burst) before relinquishing the bus. This feature, in combination with the zero clock arbitration overhead provided by the U-Bus, allows the simultaneous benefits of bus efficiency and low bus latency.

In the case of character-oriented protocols, the SDMA writes characters to memory (it does not wait for multiple characters to be received before writing), but the SDMA always reads words. This is consistent with the goal of providing low-latency operation on character-oriented protocols that tend to be used at slower rates. The read or write operation may take multiple bus cycles if the memory provides less than a 32-bit port size. For instance, a 32-bit word read from a 16-bit memory takes two SDMA bus cycles. The entire operand (4-word burst, 32 bits on reads and 8, 16, or 32 bits on writes) will be transferred in back-to-back bus cycles before the SDMA relinquishes the bus. The SDMA can steal cycles with no arbitration overhead when the MPC821 is the bus master.

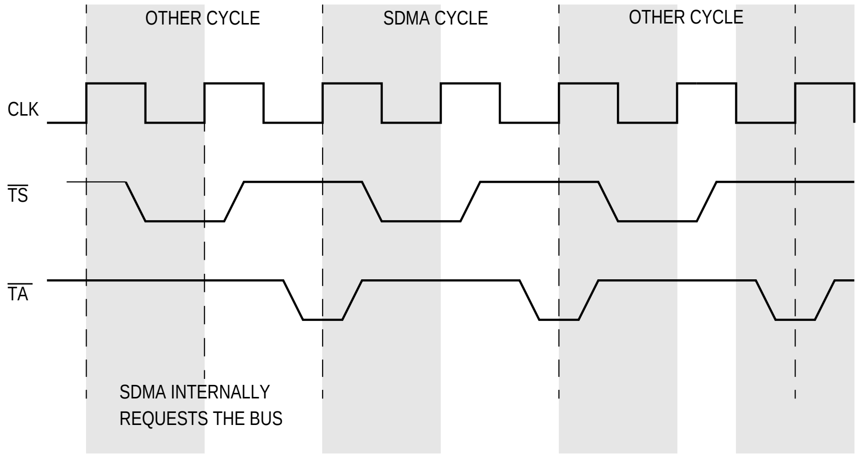


**Figure 16-39. SDMA Bus Arbitration**

### 16.10.2 SDMA Registers

The SDMA channels have one configuration register; otherwise, they are controlled transparently to the user, through the configuration of the SCCs, SMCs, SPI, and I$^2$C. The only user-accessible registers associated with the SDMA are the SDMA configuration register, SDMA address register, a read-only register used for diagnostics in case of an SDMA bus error, and the SDMA status register.

#### 16.10.2.1 SDMA CONFIGURATION REGISTER .

The 32-bit SDMA configuration register (SDCR) is used to configure all 16 SDMA channels. It is always readable and writable in the supervisor mode, although writing to the SDCR is not recommended unless the CP is disabled. The SDCR is cleared at reset.

SDCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | | | | | | | | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| ADDR | 030 | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | RES | FRZ | | RESERVED | | | | | | | | | LAID | | RAID | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| ADDR | 030 | | | | | | | | | | | | | | | |

Bits 0–16—Reserved

FRZ0–FRZ1—Freeze

These bits determine the action to be taken when the FREEZE signal is asserted. The SDMA negates $\overline{BR}$ and keeps it that way until FREEZE is negated or a reset occurs.

00 = The SDMA channels ignore the FREEZE signal.
01 = Reserved.
10 = The SDMA channels freeze on the next bus cycle.
11 = Reserved.

Bits 19–27—Reserved

RAID—RISC Controller Arbitration ID

These bits establish bus arbitration priority level among modules that have the capability of becoming bus master. In the MPC821, the I-cache, D-cache, SIU, and SDMAs can obtain bus mastership. The SDMA channel arbitration ID is determined by these bits. Arbitration IDs for all other bus masters are fixed internally.

00 = The SDMA uses the U-Bus arbitration priority 6 (BR6).
01 = The SDMA uses the U-Bus arbitration priority 5 (BR5).
10 = The SDMA uses the U-Bus arbitration priority 2 (BR2).
11 = The SDMA uses the U-Bus arbitration priority 1 (BR1).

NOTE

This value should be programmed to 01 for typical user applications.

LAID—LCD Controller Arbitration ID

00 = The LCD Controller uses the U-Bus arbitration priority 6 (BR6).
01 = The LCD Controller uses the U-Bus arbitration priority 5 (BR5).
10 = The LCD Controller uses the U-Bus arbitration priority 2 (BR2).
11 = The LCD Controller uses the U-Bus arbitration priority 1 (BR1).

NOTE

This value should be programmed to TBD for typical user applications.

#### 16.10.2.2 SDMA STATUS REGISTER.

**16.10.2.2 SDMA STATUS REGISTER.** Shared by all 12 SDMA channels, the SDMA status register (SDSR) is an 8-bit register used to report events recognized by the SDMA controller. On recognition of an event, the SDMA sets it’s corresponding bit in the SDSR. The SDSR is a memory-mapped register that can be read at any time. A bit is reset by writing a 1 and is left unchanged by writing a zero. More than one bit may be reset at a time, and the register is cleared by reset.

SDSR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | SBER | RINT | RESERVED | | | | DSP2 | DSP1 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ADDR | 908 | | | | | | | |

SBER—SDMA Channel Bus Error

This bit indicates that the SDMA channel terminated with an error during a read or write cycle. The SDMA bus error address can be read from the SDMA address register. This bit is cleared by writing a 1; writing a zero has no effect.

RINT—Reserved Interrupt

This status bit is reserved for factory testing and is cleared by writing a 1. Writing a zero has no effect.

Bits 2–5—Reserved

DSP2—DSP Chain 2 Interrupt

DSP1—DSP Chain 1 Interrupt

Refer to **Section 16.8.3 Programming Model** for details.

**16.10.2.3  SDMA MASK REGISTER.** The SDMA mask register (SDMR) is an 8-bit read/write register with the same bit format as the SDMA status register. If a bit in the SDMA mask register is a 1, the corresponding interrupt in the event register is enabled. If the bit is zero, the corresponding interrupt in the event register is masked. This register is cleared upon reset.

**16.10.2.4  SDMA ADDRESS REGISTER.** The 32-bit read-only SDMA address register (SDAR) shows the system address that is accessed during an SDMA bus error. It is undefined at reset.

## 16.11  IDMA EMULATION

The RISC microcontroller can be configured to provide a general-purpose DMA functionality through the SDMA channel. Two general-purpose independent DMA (IDMA) channels are supported. In this special emulation mode, the user can specify any memory-to-memory or peripheral-to-memory transfers that are carried out like they were by a dedicated DMA hardware.

The general-purpose IDMA controllers can operate in different modes of data transfer as programmed by the user. The IDMA can transfer data between any combination of memory and I/O. In addition, data may be transferred in either byte, half-word, word, or burst quantities and the source and destination addresses may be either odd or even. The most efficient packing algorithms are used in the IDMA transfers. The single address mode gives the highest performance, allowing data to be transferred between memory and a peripheral in a single bus cycle. The chip-select and wait-state generation logic on the MPC821 can be used with the IDMA.

The IDMA supports two buffer handling modes—auto buffer and buffer chaining. The auto buffer mode allows blocks of data to be repeatedly moved from one location to another without user intervention. The buffer chaining mode allows a chain of blocks to be moved. The user specifies the data movement using buffer descriptors that are similar to those used by an SCC. These buffer descriptions reside in the dual-port RAM.

### 16.11.1  Features

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

- Two independent, fully programmable DMA channels
- Dual address or single address transfers with 32-bit address and data capability
- 32-bit byte transfer counters
- 32-bit address pointers that can increment or remain constant
- Operand packing and unpacking for dual address transfers using the most efficient techniques
- Supports all bus-termination modes
- Provides DMA handshake for cycle steal and burst transfers
- Buffer handling modes (auto buffer and buffer chaining)

### 16.11.2 IDMA Interface Signals

The IDMA has two dedicated control signals per channel—DMA request and DMA acknowledge. The external request pins (DREQ0 and DREQ1) are used as the DMA request signals for the corresponding channel and the SDMA acknowledge signals (SDACK1 and SDACK2) are used as the DMA acknowledge. The peripheral used with these signals can either be a source or destination of the IDMA transfers. The external request signals are used also for memory-to-memory request generation and, in this case, should be connected typically to the timer that controls the transfer pace.

**16.11.2.1 DREQ AND SDACK .** These are the handshake signals between the peripheral requiring service and the MPC821. When the peripheral requires IDMA service, it asserts DREQ and the MPC821 begins the IDMA process. When the IDMA service is in progress, SDACK is asserted during accesses to the device. DREQ may be configured to be either edge sensitive or level sensitive. This is done by programming the DRnM bits in the RCCR. The DRQP bit field in the RCCR controls the priority of the IDMA channels relative to the serial channels (refer to **Section 16.4.6 RISC Controller Configuration Register**). To enable the DREQ pins, the corresponding DREQn bit in the PCSO register should be set. When the DREQs are configured as edge-sensitive requests, the edge on which a request is generated is controlled by the corresponding EDMn bit in the PCINT register. Refer to **Section 16.19.10 Port C Registers** for details.

### 16.11.3 IDMA Operation

Every IDMA operation involves the following steps—IDMA channel initialization, data transfer, and block termination. In the initialization phase, the CPU loads the IDMA-specific parameter RAM with control information, initializes the IDMA buffer descriptors, and starts the channel. In the transfer phase, the IDMA accepts requests for operand transfers and provides addressing and bus control for the transfers. The termination phase occurs when the operation is complete and the IDMA interrupts the core if interrupts are enabled. To initialize a block transfer operation, the user must initialize the IDMA registers. The IDMA buffer descriptors must be initialized with information describing the data block, device type, and other special control options. Refer to **Section 16.11.3.2 IDMA Parameter RAM** and **Section 16.11.3.7 IDMA Commands** for further details.

**16.11.3.1 AUTO BUFFER AND BUFFER CHAINING.** The host processor should initialize the IDMA buffer descriptor ring with the appropriate buffer handling mode, source address, destination address, and block length. Refer to Figure 16-40 for details.

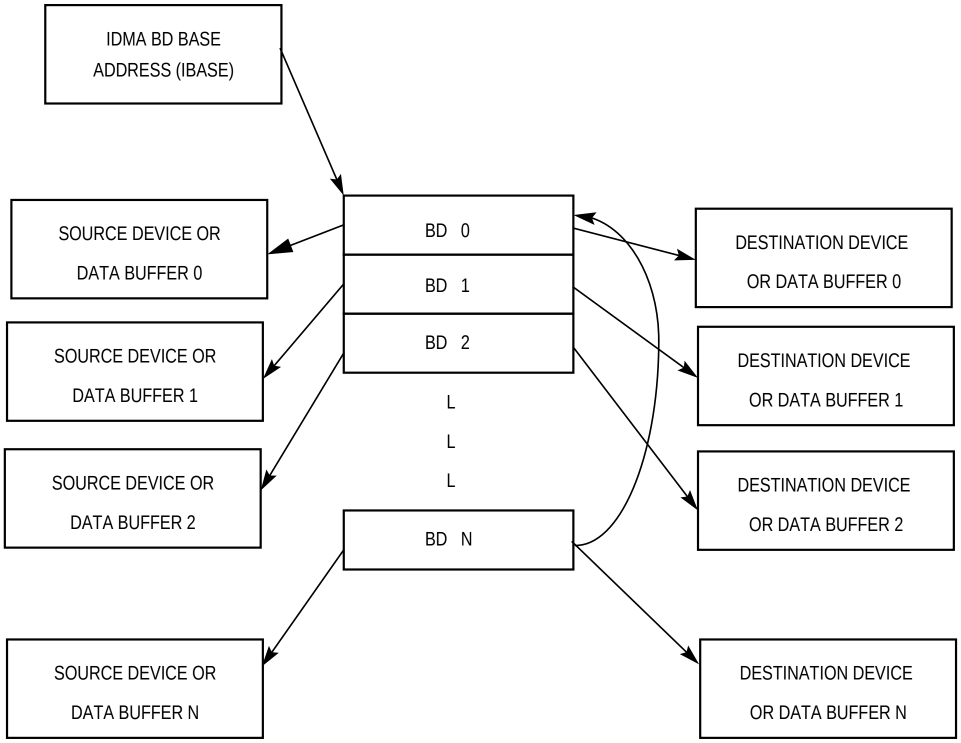

IDMA BD BASE ADDRESS (IBASE)
SOURCE DEVICE OR DATA BUFFER 0
SOURCE DEVICE OR DATA BUFFER 1
SOURCE DEVICE OR DATA BUFFER 2
SOURCE DEVICE OR DATA BUFFER N
BD 0
BD 1
BD 2
L
L
L
BD N
DESTINATION DEVICE OR DATA BUFFER 0
DESTINATION DEVICE OR DATA BUFFER 1
DESTINATION DEVICE OR DATA BUFFER 2
DESTINATION DEVICE OR DATA BUFFER N

**Figure 16-40. DMA BD Ring**

The data associated with each IDMA channel for the auto buffer and buffer chaining modes is stored in buffers and each buffer is referenced by a buffer descriptor that uses a ring structure located in the dual-port RAM.

**16.11.3.2 IDMA PARAMETER RAM.** When an IDMA channel is configured to the auto buffer or buffer chaining mode, the MPC821 uses the IDMA parameters listed in the table below.

**Table 16-21. IDMA Parameter RAM**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| DMA Base + 00 | **IBASE** | Half-word | IDMA BD Base Address |
| DMA Base + 02 | **DCMR** | Half-word | Dma Channel Mode Register |
| DMA Base + 04 | SAPR | Word | Source Internal Data Pointer |
| DMA Base + 08 | DAPR | Word | Destination Internal Data Pointer |
| DMA Base + 0C | IBPTR | Half-word | Buffer Descriptor Pointer |
| DMA_Base +0E | WRITE_SP | Half-word | |
| DMA Base + 10 | S_BYTE_C | Word | Internal Source Byte Count |
| DMA Base + 14 | D_BYTE_C | Word | Internal Destination Byte Count |
| DMA Base + 18 | S_STATE | Word | Internal State |
| DMA Base + 1C | ITEMP | Word 4 | Temp Data Storage |
| DMA Base + 2C | SR_MEM | Word | Data Storage for Peripheral Write |
| DMA Base + 30 | READ_SP | Half-word | |
| DMA Base + 32 | | Half-word | Diff Between Source and Destination Residue |
| DMA Base + 34 | | Half-word | Temp Storage Address Pointer |
| DMA Base + 36 | | Half-word | SR_MEM Byte Count |
| DMA Base + 38 | D_STATE | Word | Internal State |

NOTE: Items in bold must be initialized by the user.
DMA base = IMMR + 0x1CC0 (IDMA1) or 0x1DC0 (IDMA2).

The IBASE entry defines the starting location in the dual-port RAM for the set of IDMA buffer descriptors. It is an offset from the beginning of the dual-port RAM. The user must initialize this entry before enabling the IDMA channel and should not overlap buffer descriptor tables of two enabled serial channels or IDMA channels or erratic operation results. IBASE should contain a value that is divisible by 16.

The IBPTR entry points to the next buffer descriptor that the IDMA transfers data to when it is in IDLE state or points to the current buffer descriptor during transfer processing. After a reset or when the end of an IDMA buffer descriptor table is reached, the CP initializes this pointer to the value programmed in the IBASE entry.

**16.11.3.3 DMA CHANNEL MODE REGISTER.** The IDMA channel mode register (DCMR) is a 16-bit entry register that controls the operation mode of the IDMA channel.

DCMR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | | | | | SIZE | | S/D | | SC |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| ADDR | DMA BASE + 02 | | | | | | | | | | | | | | | |

BITS 0–10 — Reserved, should be written with zeros.

SIZE—Peripheral Port Size

- 00 = Word length.
- 01 = Half-word length.
- 10 = Byte length.
- 11= Reserved.

S/D—Source/Destination is PERIPHERAL or MEMORY

- 00 = Read and write from memory.
- 01 = Read from peripheral, write to memory.
- 10 = Read from memory, write to peripheral.
- 11 = Reserved.

SC—Single Cycle

- 0= Dual cycle mode.
- 1= Single cycle mode.

The source address pointer (SAPR) entry points to the next source data byte that the IDMA transfers and the destination address pointer (DAPR) entry points to the next destination byte that the IDMA writes. At the beginning of the buffer descriptor processing, the CP initializes these pointers to the values programmed in buffer descriptor. The remaining parameters are only for RISC use.

**16.11.3.4 IDMA STATUS REGISTER.** The IDMA status register (IDSR) is an 8-bit register used to report events recognized by the IDMA controller. On recognition of an event, the IDMA sets it's corresponding bit in the IDSR. The IDSR is a memory-mapped register that can be read at any time. A bit is reset by writing a 1 and is left unchanged by writing a zero. More than one bit can be reset at a time and the register is cleared by reset.

IDSR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | OB | DONE | AD |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R/W | R/W | R/W |
| ADDR | 910 (IDSR1), 918 (IDSR2) | | | | | | | |

Bits 0–4—Reserved

OB—Out of Buffers

This bit indicates that the IDMA channel has no valid buffer descriptors.

DONE—IDMA Transfer Done

This bit indicates that the IDMA channel terminated a transfer. It is set after servicing a buffer descriptor that has the L (Last) status bit set.

AD—Auxiliary Done

This status bit is set after a buffer descriptor, which has it's I (interrupt) set, is serviced.

**16.11.3.5 IDMA MASK REGISTER.** The IDMA mask register (IDMR) is an 8-bit read/write register with the same bit format as the IDSR. If a bit in the IDMR is a 1, the corresponding interrupt in the status register is enabled and if it is zero, the corresponding interrupt in the status register is masked. This register is cleared upon reset.

**16.11.3.6 IDMA BUFFER DESCRIPTORS.** Source addresses, destination addresses, and byte counts are presented to the RISC controller using the special IDMA buffer descriptors. The RISC controller reads the buffer descriptors (BDs), programs the SDMA channel, and notifies the CPU about the completion of a buffer transfer using the IDMA BDs. This concept is similar to the one used for the serial channels on the MPC821, except that the BD is larger because it contains additional information.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | V | — | W | I | L | — | CM | — | — | — | — | — | — | — | — | — |
| OFFSET + 2 | DFCR | | | | | | | | SFCR | | | | | | | |
| OFFSET + 4 | DATA LENGTH | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |
| OFFSET + 8 | SOURCE DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| OFFSET + A | | | | | | | | | | | | | | | | |
| OFFSET + C | DESTINATION DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| OFFSET + E | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

The following bits are prepared by the user before transfer and are set by the RISC controller after the buffer has been transferred.

V—Valid

0 = The data buffers associated with this BD are not currently ready for transfer. The user is free to manipulate this BD or its associated data buffer. When it is not in auto buffer mode, the RISC controller clears this bit after the buffer has been transferred (or after an error condition is encountered).

1 = The data buffers have been prepared for transfer by the user. Notice that only one data buffer needs to be prepared if the source/destination is a peripheral device. It can only be the source data buffer when the destination is a device or the destination data buffer when the source is a device. No fields of this BD can be written by the user once this bit is set.

**NOTE**

The only difference between auto buffer mode and buffer chaining mode is that the V-bit is not cleared by the RISC controller in the auto buffer mode. Auto buffer mode is enabled by the CM bit.

W—Wrap (Final BD in Table)

0 = This is not the last BD in the table.

1 = This is the last BD in the table. After the associated buffer has been used, the RISC controller transfers data from the first BD that IBASE points to in the table. The number of BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = Once this buffer is serviced by the RISC controller the AD bit in the IDSR is set, which can cause an interrupt.

L—Last

0 = This is not the last buffer to be transferred in the buffer chaining mode. The I-bit can be used to generate an interrupt when this buffer is serviced.

1 = This is the last buffer to be transferred in the buffer chaining mode. When the transfer count is exhausted, an interrupt (DONE) is generated, regardless of the I-bit.

CM—Continuous Mode

0 = Buffer chaining mode. The RISC clears the V-bit after this BD is serviced. The buffer chaining mode is used for transferring large quantities of data into noncontiguous buffer areas. The user can initialize BDs ahead of time, if needed. The RISC controller automatically reloads the IDMA registers from the next BD values when the transfer is terminated.

1 = Auto buffer mode (continuous mode). The RISC does not clear the V-bit after this BD is serviced. This is the only difference between auto buffer mode and buffer chaining mode behavior. The auto buffer mode is used to transfer multiple groups of data to/from a buffer ring. This mode does not require reprogramming. The RISC controller automatically reloads the IDMA registers from the next BD values when the transfer is terminated. Either a single BD or multiple BDs can be used in this mode to create an infinite loop of repeated data moves.

**NOTE**

The I-bit can still be used to generate an interrupt in this mode.

**Source Function Code Registers.** The source function code register (SFCR) is an 8-bit register containing the value that the user would like to appear on the address type pins AT0 through AT3 when the associated DMA channel accesses the source memory. It also controls the byte-ordering convention to be used in the transfers.

SFCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1 | AT2 | AT3 |
| ADDR | OFFSET + 3 | | | | | | | |

Bits 0–2—Reserved

BO—Byte Ordering

This bit field should be set by the user to select the required byte ordering of the data buffer. If this bit field is modified on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next BD.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during the SDMA channel memory access. AT0 is always driven to 1to identify this SDMA channel access as a DMA-type access.

**Destination Function Code Registers.** The destination function code register (DFCR) is an 8-bit register containing the value that the user would like to appear on the address type pins AT0 through AT3 when the associated DMA channel accesses the destination memory. It also controls the byte-ordering convention to be used in the transfers.

DFCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1 | AT2 | AT3 |
| ADDR | OFFSET + 2 | | | | | | | |

Bits 0–2—Reserved

BO—Byte Ordering

This bit field should be set by the user to select the required byte ordering of the data buffer. If this bit field is modified on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next BD.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during the SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

Data Length

The data length is the number of bytes that the IDMA should transfer to or from this BD data buffer. The data length should be programmed to a value greater than zero.

Source Buffer Pointer

The source buffer pointer contains the address of the associated source data buffer and can reside in internal or external memory.

**NOTE**

In single address mode when the source is a device, this field is ignored. In dual address mode when the source is a device, this field should contain the device address.

Destination Buffer Pointer

The destination buffer pointer contains the address of the associated destination data buffer. The buffer may reside in either internal or external memory.

**NOTE**

> In single address mode when the destination is a device, this field is ignored. In dual address mode when the destination is a device, this field should contain the device address.

The terminal count code AT(0:3) = 0xF will replace the normal SFCR/DFCR code for the last IDMA cycle in the peripheral side.

### 16.11.3.7 IDMA COMMANDS.

**16.11.3.7.1 INIT_IDMA.** This command causes the RISC controller to reinitialize it's IDMA internal state to the condition it had after a system reset. The IDMA BD pointer is reinitialized to the top of BD ring.

**16.11.3.7.2 STOP_IDMA.** This command causes the RISC controller to terminate current IDMA transfers. The DONE bit is set in the IDSR and the current BD is closed. If the peripheral device is the source, the IDMA internal buffer is transferred to memory before termination. At the next request, the following BD in the chain is processed.

**16.11.3.8 STARTING THE IDMA.** Once the channel has been initialized with all parameters required for a transfer operation, it is started by setting the DREQn bit in the port C special option (PCSO) register. Once DREQn has been set, the channel is active and accepts operand transfer requests through the channel's corresponding DREQ pin. When the first valid external request is recognized, the IDMA arbitrates for the bus and the DREQ input is ignored until the DREQn bit is set in the PCSO register.

The software can suspend channel transfer operation at any time by clearing the DREQn bit in the PCSO register. In response, any operand transfer in progress will be completed, and the bus will be released. No further bus cycles are started while DREQn remains cleared. During this time, the CPU can access IDMA internal registers to determine channel status or to alter operation. When DREQn is set again, if a transfer request is pending, the IDMA arbitrates for the bus and continues normal operation. Interrupts from the IDMA are sent to the interrupt controller. In the interrupt handler, the unmasked bits in the IDSR should be cleared (by writing them with a 1) to negate the interrupt request to the CPM interrupt controller.

**16.11.3.9 REQUESTING IDMA TRANSFERS.** Once the IDMA has been started, transfers to the IDMA can be requested. IDMA transfers may be initiated by externally generated request that are requested by an external device using the DREQ pin in conjunction with the activation of the DREQn bits in the PCSO register.

**16.11.3.10 LEVEL-SENSITIVE MODE.** For external devices requiring very high data transfer rates, level-sensitive mode allows the IDMA to use a maximum bandwidth to service the device. In this mode, the DREQ input to IDMA is level-sensitive and sampled at rising edges of the clock to determine when a valid request is asserted by the device. The device requests service by asserting DREQ and leaving it asserted as long as it needs service.

Each time the IDMA issues a bus cycle to either read or write the device, the IDMA outputs the SDACK signal. The device is either the source or destination of the transfers, as determined by the S/D bits in the DCMR. SDACK is the acknowledgment of the original burst request given on the DREQ pin. DREQ should be negated during SDACK active period to guarantee that no further cycles are performed.

**16.11.3.11 EDGE-SENSITIVE MODE.** For external devices that generate a pulsed signal for each operand to be transferred, the edge-sensitive mode should be used. In edge-sensitive mode, IDMA moves one operand for each falling edge of the DREQ input. In this mode, DREQ is sampled at each rising edge of the clock to determine when a valid request is asserted by the device. When IDMA detects a falling edge on DREQ, a request becomes pending and remains pending until it is serviced by the IDMA. Further falling edges on DREQ are ignored until the request begins to be serviced. The servicing of the request results in one operand being transferred. Each time the IDMA issues a bus cycle to either read or write the device, the IDMA outputs the SDACK signal. The device is either the source or destination of the transfers, as determined by the S/D bit in the DCMR. Thus, SDACK is the acknowledgment of the original cycle steal request given on the DREQ pin.

**16.11.3.12 IDMA OPERAND TRANSFERS.** Once the IDMA successfully arbitrates for the bus, it can begin making operand transfers. The source IDMA bus cycle has timing identical to an internal master read bus cycle. The destination IDMA bus cycle has timing identical to an internal master write bus cycle. The two-channel IDMA controller supports dual and single address transfers. The dual address operand transfer consists of a source operand read and a destination operand write. Each single address operand transfer consists of one external bus cycle, that allows either a read or write cycle to occur.

**16.11.3.12.1 Dual Address Mode.** The two IDMA channels can each be programmed to operate in a dual address transfer mode. In this mode, the operand is read from the source address specified by the pointer and placed in the internal storage. The operand read could take several bus cycles to complete because of differences in source and destination operand sizes. The operand is then written to the address specified in the destination address pointer. This transfer may also be several bus cycles long. In this manner, various combinations of peripheral, memory, and operand sizes can be used.

**Dual Address Source Read.** During this type of IDMA cycle, the SAPR drives the address bus, the SFCR drives the source address type, and the DCMR drives the size control. Data is read from the memory or peripheral and placed in the internal storage when the bus cycle is terminated. When the complete operand has been read, the SAPR is incremented by 1, 2, 4, or 16, depending on the address and size information specified by the channel mode register. Refer to **Section 16.11.3.2 IDMA Parameter RAM** for more information.

**Dual Address Destination Write*.*** During this type of IDMA cycle, the data in the internal storage is written to the device or memory selected by the address in the DAPR, the destination address type in the DFCR, and the size in the DCMR. The same options exist for operand size and alignment as in the dual address source read. When the complete operand is written, the DAPR is incremented by 1, 2, 4, or 16 according to the DCMR, and the BTC is decremented by the number of bytes transferred. If the BTC is equal to zero and the transfer is completed with no errors, the DONE bit in the IDSR is set. Refer to **Section 16.11.3.2 IDMA Parameter RAM** for more information.

**Dual Address Packing*.*** When the dual address mode is selected, the IDMA performs packing. Regardless of the source size, destination size, source starting address, or destination starting address, the IDMA uses the most efficient packing algorithm possible to perform the transfer in the fewest possible number of bus cycles.

**16.11.3.13 SINGLE ADDRESS MODE.** Each IDMA channel can be independently programmed to provide single address transfers. The internal storage is not used by the IDMA, since the transfer occurs directly from a device to memory. This mode is often referred to as flyby mode because the internal storage is not used. The external request is used to start a transfer when the single address mode is selected. The source/destination (S/D) bits in the channel mode register controls whether a source read or a destination write cycle occurs on the data bus. If the S/D field =01, the external handshake signals are used with the source operand and a single address source write occurs. If the S/D field =10, the external handshake signals are used with the destination operand and a single address destination read occurs.

**Single Address Source Read.** During the single address source read cycle, the device or memory selected by the address in the SAPR, the source address type in the SFCR, and the size in the channel mode register provides the data and control signals on the data bus. This bus cycle operates like a normal read bus cycle. The destination device is controlled by the IDMA handshake signals (DREQ and SDACK). The assertion of SDACK provides the write control to the destination device. For more details about the IDMA handshake signals, refer to **Section 16.11.2 IDMA Interface Signals**.

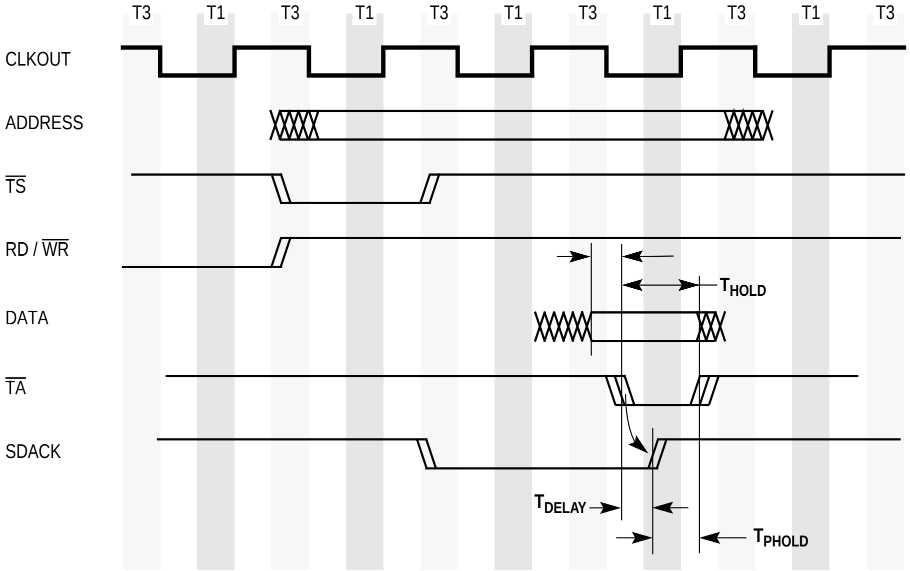


Figure 16-41. SDACK Timing Diagram–Single Address Peripheral Write, Asynchronous TA

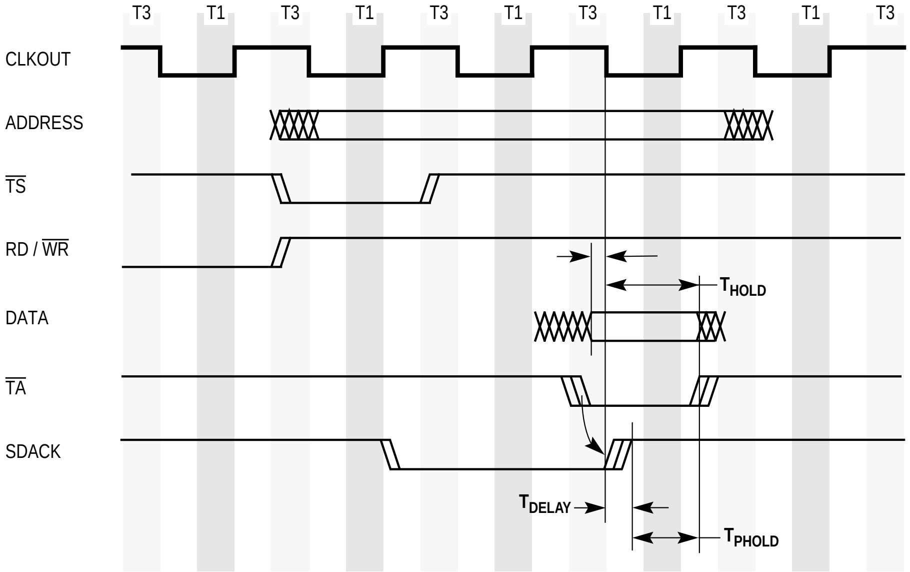


Figure 16-42. SDACK Timing Diagram–Single Address Peripheral Write, Synchronous TA

**Single Address Destination Write.** During the single address destination write cycle, the source device is controlled by the IDMA handshake signals (DREQ and SDACK). When the source device requests service from the IDMA channel, the IDMA asserts SDACK to allow the source device to drive data onto the data bus. The data is written to the device or memory selected by the address in the DAPR, the destination address type in the DFCR, and the size in the DCMR. The data bus is driven to three-state for this write cycle. For more details about the IDMA handshake signals, refer to **Section 16.11.2 IDMA Interface Signals**.

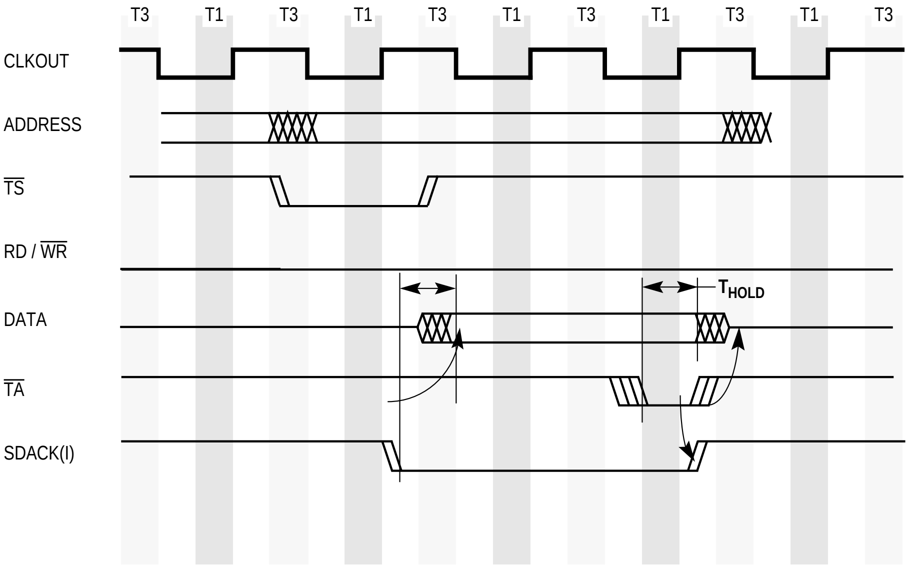


**Figure 16-43. SDACK Timing Diagram–Single Address Peripheral Read, Synchronous $\overline{TA}$**

**16.11.3.13.1 Externally Recognizing IDMA Operand Transfers.** There are a few ways to externally determine that a bus cycle is being executed by the IDMA:

1. The address type lines or SDMA channels can be programmed to a unique function code that identifies an IDMA transfer.
2. The SDACK signal shows accesses to the peripheral device. SDACK activates on either the source or destination bus cycles, depending on the S/D bits in the DCMR.

**16.11.3.14 BUS EXCEPTIONS.** While the IDMA has the bus and is performing operand transfers, it is possible for bus exceptions to occur. In any computer system, the possibility exists that an error will occur during a bus cycle due to a hardware failure, random noise, or an improper access. When a synchronous bus structure (like those supported by the MPC821) is used, it is easy to make provisions allowing a bus master to detect and respond to errors during a bus cycle. The IDMA recognizes the same bus exceptions as the CPU core—reset and transfer error.

**16.11.3.14.1 Reset.** On an external reset, the IDMA immediately aborts the channel operation, returns to the idle state, and clears the IDSR. If a bus cycle is in progress when reset is detected, the cycle is terminated, the control and address/data pins are three-stated, and bus ownership is released.

**16.11.3.14.2 Transfer Error.** When a fatal error occurs during a bus cycle, a bus error exception is used to abort the cycle and systematically terminate that channel operation. The IDMA terminates the current bus cycle, signals an error in the SDSR, and signals an interrupt if the corresponding bit in the SDMR is set. The IDMA waits for a reset of the RISC microcontroller before starting any new bus cycles. Any data that was previously read from the source into the internal storage is lost.

**NOTE**

Any device that is the source or destination of the operand under IDMA handshake control for single address transfers may need to monitor $\overline{\text{TEA}}$ to detect a bus exception for the current bus cycle. $\overline{\text{TEA}}$ terminates the cycle immediately and negates SDACK, which is used to control the transfer to or from the device.

## 16.12 SERIAL INTERFACE WITH TIME-SLOT ASSIGNER

The serial interface (SI) connects the physical layer serial lines to the two SCCs and two serial managment controllers (SMCs). In its simplest configuration, the SI allows the two SCCs and SMCs to be connected to their own set of individual pins. Each SCC or SMC that connects to the external world in this way is said to connect to a nonmultiplexed serial interface (NMSI). In the NMSI configuration, the SI provides a flexible clocking assignment for each SCC and SMC from a bank of external clock pins and/or internal baud rate generators. Refer to Figure 16-44 for details.

However, the main feature of the SI is its time-slot assigner (TSA) which allows any combination of SCCs and SMCs to multiplex their data together on one or two time-division multiplexed (TDM) channels. Common examples of TDMs are the T1 lines in the U.S. and Japan and the CEPT lines in Europe. Even if the TSA is not used in it’s intended capacity, it can still be used to generate complex waveforms on four output pins. For instance, these pins can be programmed by the TSA to implement stepper motor control or variable duty cycle and period control on these pins. Any programmed configuration can be changed on-the-fly.

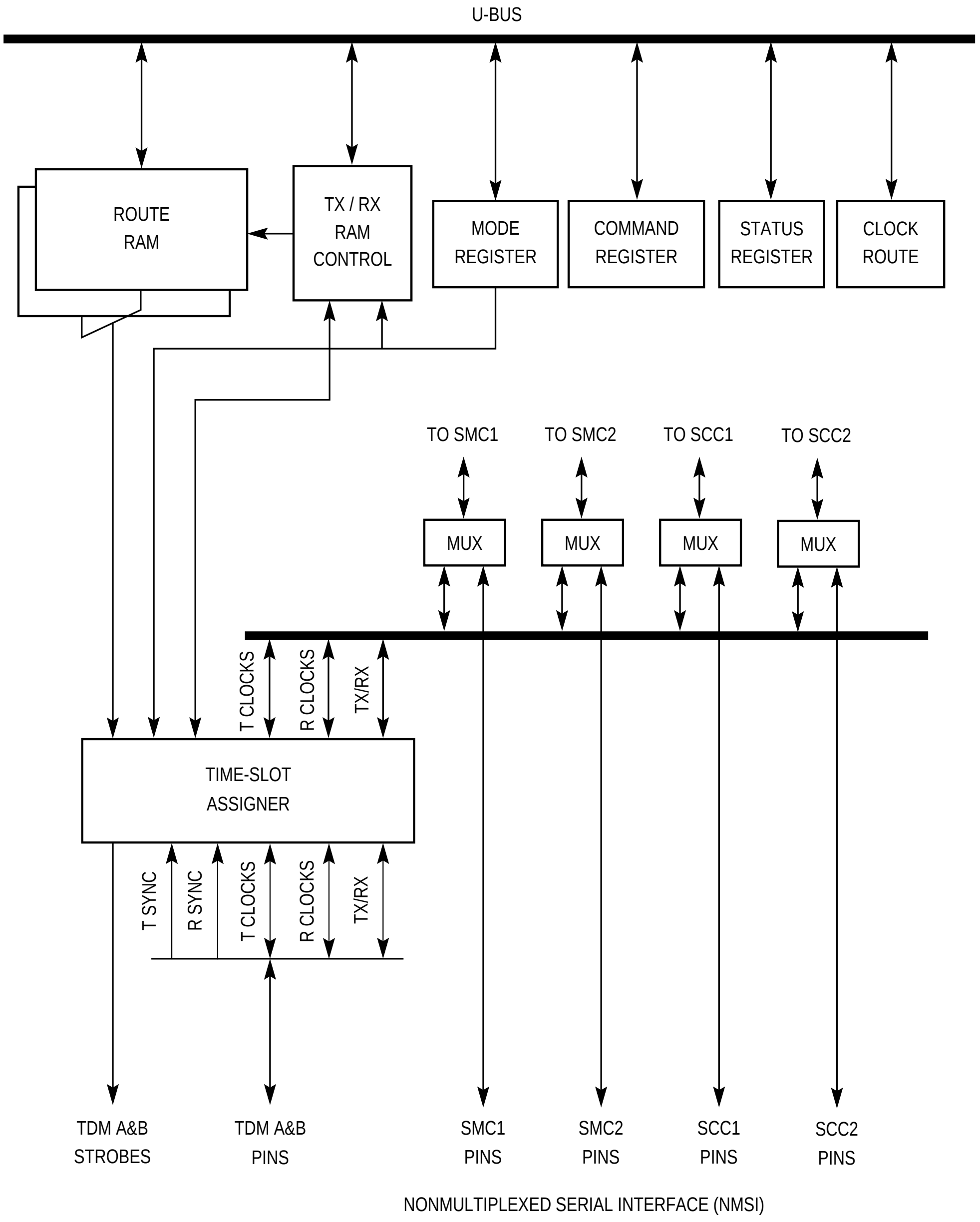


**Figure 16-44. SI Block Diagram**

### 16.12.1 Features

The two major features of the serial interface are the TSA and NMSI. The TSA contains the following important features:

- Can connect to two independent TDM channels. Each TDM can be one of the following:
  - T1 or CEPT line
  - Pulse code modulation highway
  - Integrated services digital network primary rate
  - ISDN basic rate–interchip digital link
  - ISDN basic rate–general circuit interface
  - User-defined interfaces
- Independent, programmable transmit and receive routing paths
- Independent transmit and receive frame syncs allowed
- Independent transmit and receive clocks allowed
- 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 (2×) clock output pins
- 1- or 8-bit resolution in routing, masking, and strobe selection
- Supports frames up to 8,192 bits long
- Internal routing and strobe selection can be dynamically programmed
- Supports automatic echo and loopback mode for each TDM

The NMSI contains the following features:

- Each SCC and SMC can be independently programmed to work with it's own set of pins in a nonmultiplexed manner
- Each SCC can have it's own set of modem control pins
- Each SMC can have it's own set of four pins
- Each SCC and SMC can derive clocks externally from a bank of eight clock pins or a bank of four baud rate generators

### 16.12.2 Overview

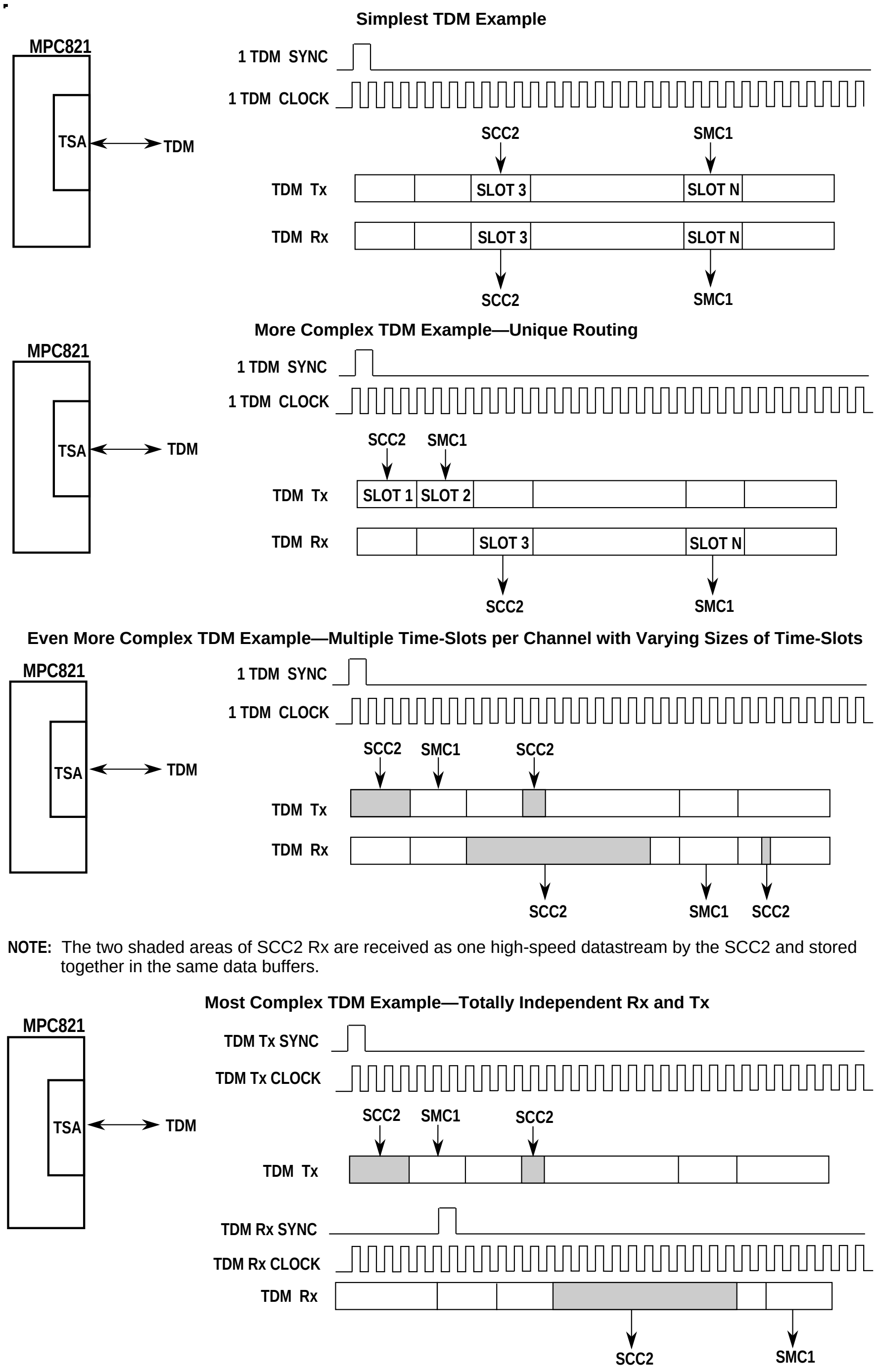
The TSA implements both the internal route selection and time-division multiplexing for multiplexed serial channels. The TSA supports the serial bus rate and format for most standard TDM busses, including the T1 and CEPT highways, the pulse code modulation (PCM) highway, and the ISDN busses in both basic and primary rates. The two popular ISDN basic rate busses–interchip digital link (IDL) and general circuit interface (GCI), also known as IOM-2–are supported. Because it supports two TDMs, the TSA provides an additional level of flexibility. It is therefore possible to simultaneously support a T1 line and a CEPT line, a basic rate, and a primary rate ISDN channel.

TSA programming is completely independent of the protocol used by the SCC or SMC. For instance, the fact that SCC2 can be programmed for the HDLC protocol has no impact on the programming of the TSA. The purpose of the TSA is to route the data from the specified pins to the preferred SCC or SMC at the correct time. It is the job of the SCC or SMC to handle the data it receives. In it’s simplest mode, the TSA identifies the frame using one sync pulse and one clock signal provided externally by the user. This can be enhanced to allow independent routing of the receive and transmit data on the TDM. Additionally, the definition of a time-slot need not be limited to 8 bits or even to a single contiguous position within the frame. Finally, the user can provide separate receive and transmit syncs as well as clocks. These various configurations are illustrated in Figure 16-45.

The TSA also allows two TDM channels to be simultaneously supported. Thus, in its most flexible mode, the TSA can provide two separate TDM channels, each with an independent receive and transmit routing assignment and independent sync pulse and clock inputs. Thus, the TSA can support four, independent, half-duplex TDM sources, two in reception and two in transmission, using four sync inputs and four clock inputs. Refer to Figure 16-46 for details. In addition to channel programming, the TSA supports up to four strobe outputs that may be asserted on a bit or byte basis. These strobes are completely independent from the channel routing used by the SCCs 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 multitransmitter architecture. Notice that open-drain programming on the TXDx pins that support a multitransmitter architecture is programmed in the parallel I/O block. These strobes can also be used for generating output waveforms to support such applications as stepper motor control.

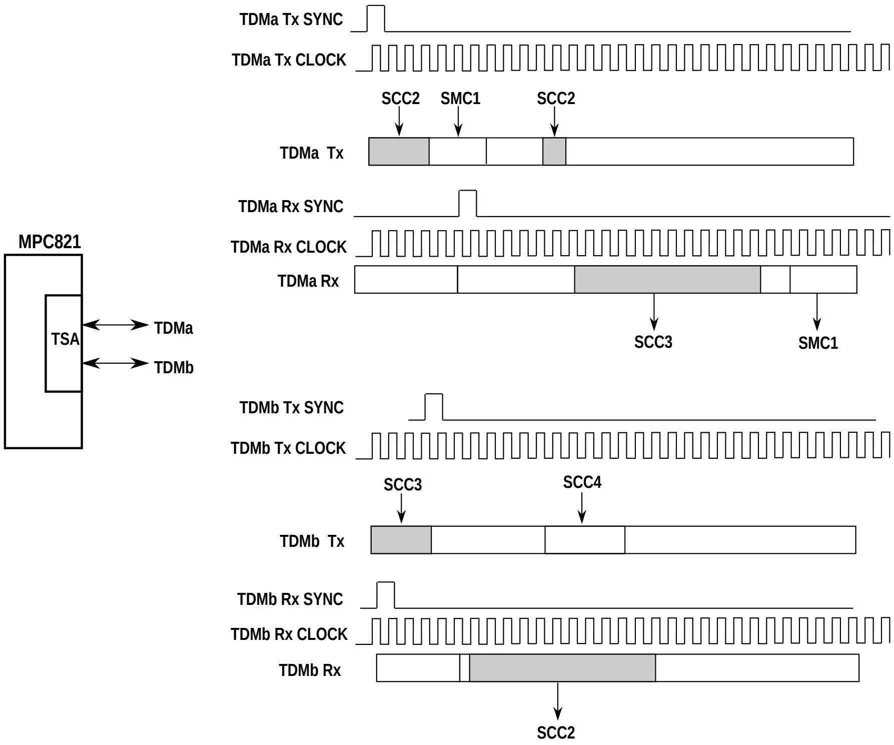
Most TSA programming is accomplished in two SI RAMs, each a size of 64 × 16 bits. These SI RAMs are directly accessible by the host processor in the internal register section of the MPC821 and are not associated with the dual-port RAM. One SI RAM is always used to program the transmit routing and the other SI RAM is always used to program the receive routing. With the SI RAMs, the user can define the number of bits/bytes to be routed to the SCC or SMC and decide when the external strobes are to be asserted and negated.

The size of the SI RAM that is available for time-slot programming depends on the configuration. If two TDM channels are selected, the available SI RAM entries on a channel are reduced by half. If on-the-fly changes are also allowed, the SI RAM entries are further reduced by one-half. Even in a configuration with two TDM channels and on-the-fly changes allowed, the SI RAM size is still sufficient to allow extensive time-slot programming flexibility. The maximum frame length that can be supported in any configuration is 8,192 bits. 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 has received.



**NOTE:** The two shaded areas of SCC2 Rx are received as one high-speed datastream by the SCC2 and stored together in the same data buffers.

**Figure 16-45. Various Configurations of a Single TDM Channel**



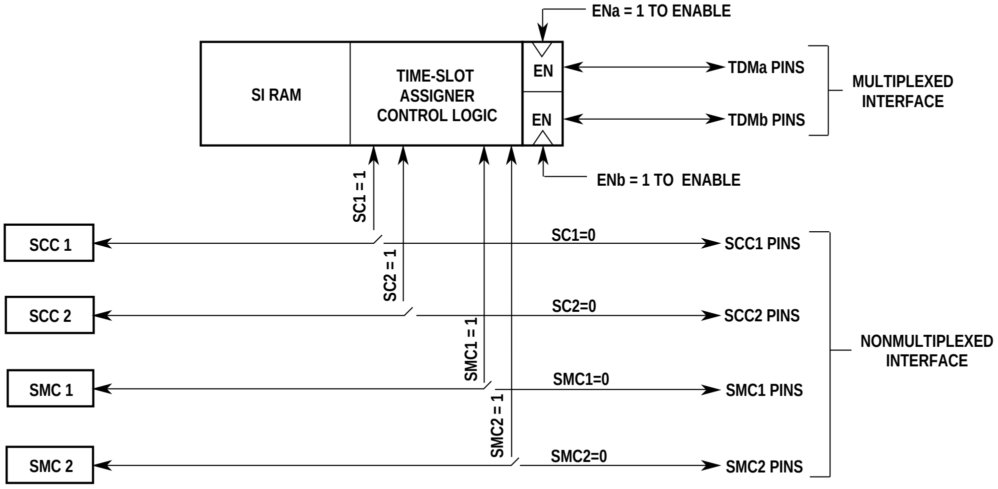
NOTE: SCCs can receive on one TDM and transmit on another (SCC2 and SCC3).

Figure 16-46. Dual TDM Channel Example

The physical interface echo mode differs from the individual SCC echo mode in that it can operate on the entire TDM signal, rather than just on a particular SCC channel. The loopback mode causes the physical interface to receive the same signal it is transmitting. The SI loopback mode checks more than the individual SCC loopback; it checks the SI and the internal channel routes. The maximum external serial clock that may be an input to the TSA is system CLK/2.5. If an SCC or SMC is operating with the NMSI, then the serial clock rate may be slightly faster at a value not to exceed system CLK/2.

### 16.12.3 Enabling Connections to the Time-Slot Assigner

Each SCC and SMC can be independently enabled to connect to the TSA. Notice that separate bits decide whether each SCC or SMC is connected to the TSA or to it's own set of external pins. Additionally, the two TDM interfaces must be enabled to be connected to the TSA. Refer to Figure 16-47 for more information. Once the connections are made, the exact routing decisions are made in the SI RAM.



NOTES:

1. The ENx bits are located in the SIGMR.
2. The SCx bits are located in the SICR.
3. The SMCx bits are located in the SIMODE.
4. The clocking paths are not shown for the nonmultiplexed interface .

**Figure 16-47. Enabling Connections Through the SI**

### 16.12.4 Serial Interface RAM

The SI has two 64 × 16 static RAMs to control the routing of the TDM channels to the SCCs and SMCs. The RAMs are uninitialized after power-on. For proper operation, the host should program the RAMs before enabling the multiplexed channels or unwanted results can occur. The RAM consists of 16-bit entries that are used to define the routing control. Each entry can control from 1 to 16 bits or from 1 to 16 bytes at a time as determined in the entry. In addition to the routing, up to four strobe pins can be asserted according to the programming of the RAM. The strobes are active high. The two SI RAMs can be configured in four different ways to support various TDM channels.

## NOTE

When using a single TDM, is it simplest to use IDMA since the entire SI routing RAM is available to TDM A, whereas only the last half of the SI RAM is accessible by TDM B. In other words, the SI RAM usage for TDM B begins at entry 64 rather than entry 0. For more information, see **Section 16.12.5.5 SI Status Register**.

**16.12.4.1 ONE MULTIPLEXED CHANNEL WITH STATIC FRAMES.** With this configuration, there are 64 entries in the SI RAM for transmit data and strobe routing and 64 entries for receive data and strobe routing. This configuration should be chosen only when one TDM is required and the routing on that TDM does not need to be dynamically changed. Refer to Figure 16-48 for details.

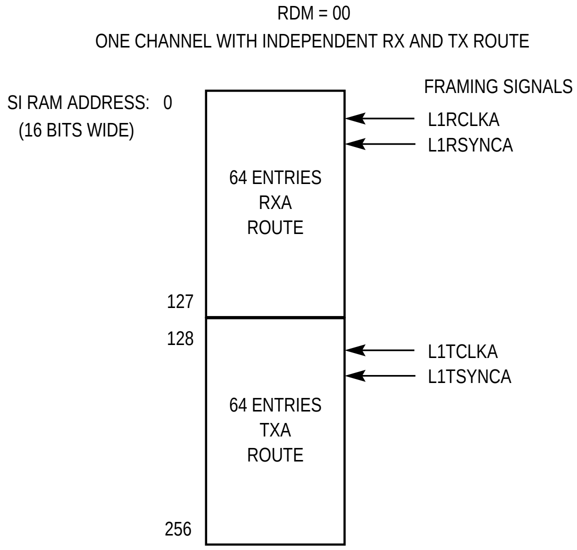


**Figure 16-48. SI RAM–One TDM With Static Frames**

**16.12.4.2 ONE MULTIPLEXED CHANNEL WITH DYNAMIC FRAMES.** With this configuration, there is one multiplexed channel and it has 32 entries for transmit data and strobe routing and 32 entries for receive data and strobe routing. In each RAM, one of the partitions is the current-route RAM and the other is a shadow RAM that allows the user to change the serial routing. After programming the shadow RAM, the user sets the CSRx bit of the associated channel in the SI CR. When the next frame sync arrives, the SI automatically exchanges the current-route RAM for the shadow RAM. Refer to **Section 16.12.4.7 SI RAM Dynamic Changes** for more details on how to dynamically change the channel route. This configuration should be chosen when only one TDM is required but the routing on that TDM may need to be dynamically changed. Refer to Figure 16-49 for details.

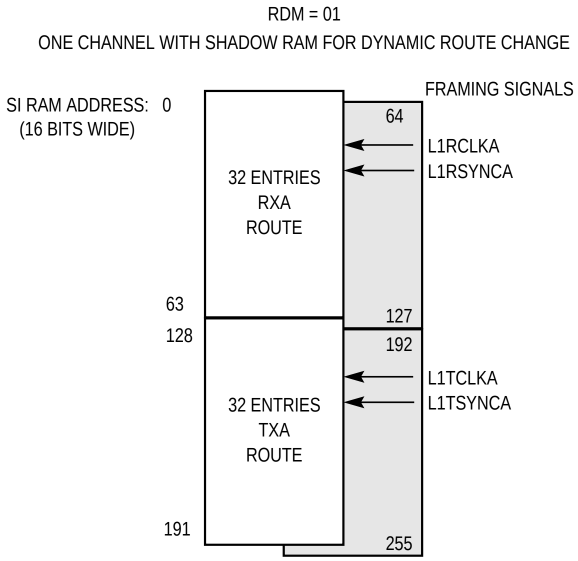


**Figure 16-49. SI RAM–One TDM With Dynamic Frames**

**16.12.4.3 TWO MULTIPLEXED CHANNELS WITH STATIC FRAMES.** With this configuration, there are 32 entries for transmit data and strobe routing and 32 entries for receive data and strobe routing. This configuration should be chosen when two TDMs are required and the routing on that TDM does not need to be dynamically changed. Refer to Figure 16-50 for details.

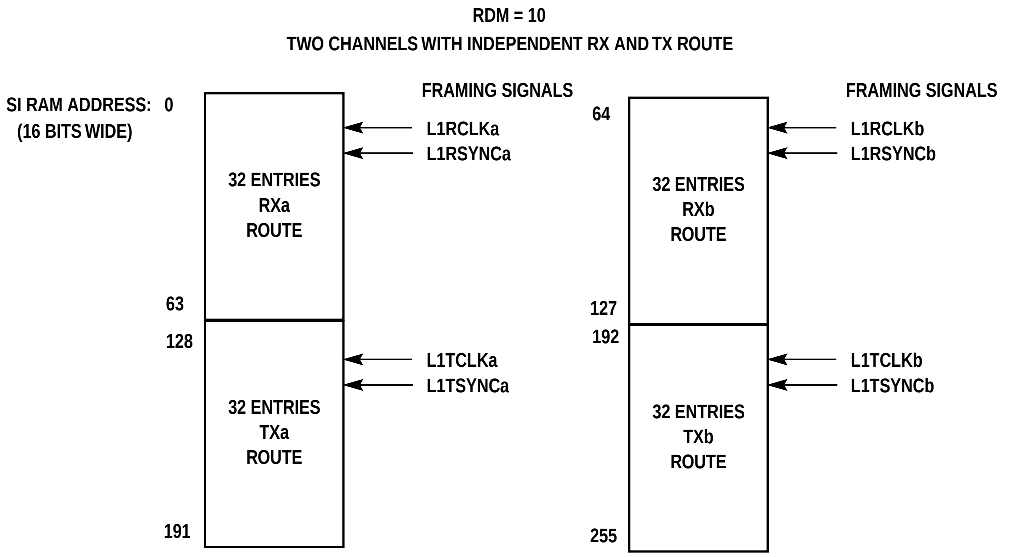


**Figure 16-50. SI RAM–Two TDMs With Static Frames**

**16.12.4.4 TWO MULTIPLEXED CHANNELS WITH DYNAMIC FRAMES.** With this configuration, there are two multiplexed channels. Each channel has 16 entries for transmit data and strobe routing and 16 entries for receive data and strobe routing. In each RAM, one of the partitions is the current-route RAM and the other is a shadow RAM that allows the user to change the serial routing. After programming the shadow RAM, the user sets the CSRx bit of the associated channel in the SI CR. When the next frame sync arrives, the SI automatically exchanges the current-route RAM for the shadow RAM. Refer to **Section 16.12.4.7 SI RAM Dynamic Changes** for more details on how to dynamically change the channel route. This configuration should be chosen when two TDMs are required and the routing on each TDM may need to be dynamically changed. Refer to Figure 16-51 for details.

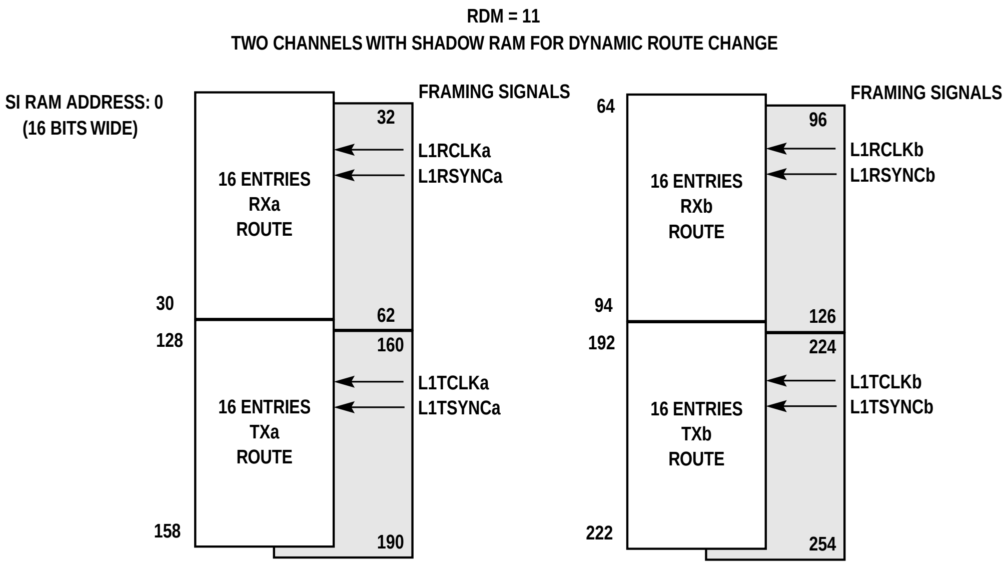


**Figure 16-51. Two TDMs With Dynamic Frames**

**16.12.4.5 PROGRAMMING SI RAM ENTRIES.** The programming of each word within the RAM determines the routing of the serial bits (or bit groups) and the assertion of strobe outputs. The RAM programming codes are shown in the following table.

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | LOOP | SW TR | SSEL 1 | SSEL 2 | SSEL 3 | SSEL 4 | RES | CSEL | | | CNT | | | | BYT | LST |
| 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 |
| ADDR | C00 – DFF | | | | | | | | | | | | | | | |

LOOP—Loop Back on This Time-Slot

0 = Normal mode.
1 = Loopback mode for this time-slot.

SWTR—Switch Tx and Rx

The SWTR bit is only valid in the receive route RAM and is ignored in the transmit route RAM. This bit affects the operation of both the L1RXD and L1TXD pins.

The SWTR bit is only set in special situations where the user prefers to receive data from a transmit pin and transmit data on a receive pin. For instance, consider the situation where devices A and B are connected to the same TDM, each with different time-slots. Normally, there is no opportunity for stations A and B to communicate with each other directly over the TDM, since they both receive the same TDM receive data and transmit on the same TDM transmit signal. Refer to Figure 16-52 for details.

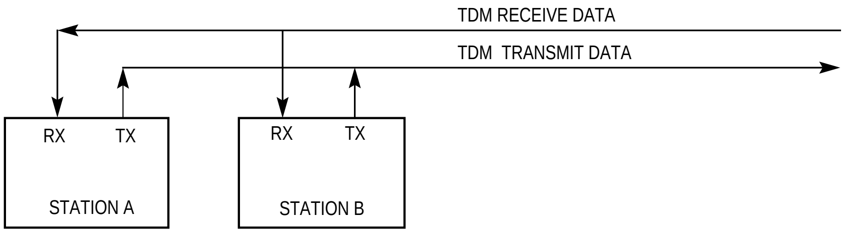


**Figure 16-52. Using the SWTR Feature**

The SWTR option gives station B the opportunity to listen to transmissions from station A and transmit data to station A. To do this, station B would set the SWTR bit in it's receive route RAM. For this entry, receive data is taken from the L1TXD pin and data is transmitted on the L1RXD pin. If the user only wants to listen to station A transmissions and not transmit data on L1RXD, then the CSEL bits in the corresponding transmit route RAM entry should be cleared to prevent transmission on the L1RXD pin.

It is also possible for station B to transmit data to station A by setting the SWTR bit of the entry in its receive route RAM. Data is transmitted on the L1RXD pin rather than the L1TXD pin, according to the transmit route RAM. Note that this configuration could cause collisions with other data on the L1RXD pin unless care is taken to choose an available (quiet) time-slot. If the user only wants to transmit on L1RXD and not receive data on L1TXD, then the CSEL bits in the receive route RAM should be cleared to prevent reception of data on L1TXD.

**NOTE**

> If the transmit and receive sections of the TDM do not use a single clock source, this feature can cause erratic results to occur.

0 = Normal operation of the L1TXD and L1RXD pins.
1 = Data is transmitted on the L1RXD pin and is received from the L1TXD pin for the duration of this entry.

SSEL4–SSEL1—Strobe Select

The four strobes (L1ST1, L1ST2, L1ST3, and L1ST4) can be assigned to the receive RAM and asserted/negated with L1RCLKa or L1RCLKb. Or it can be assigned to the transmit RAM and asserted/negated with L1TCLKa or L1TCLKb. Each bit corresponds to the value the strobe should have during this bit/byte group. Multiple strobes can be asserted simultaneously, if preferred. If a strobe is configured to be asserted in two consecutive SI RAM entries, then it remains continuously asserted during the processing of both SI RAM entries. If a strobe is asserted on the last entry in the table, the strobe is negated after the last entry has completed processing.

**NOTE**

> Each strobe is changed with the corresponding RAM clock and is only output if the corresponding parallel I/O is configured as a dedicated pin. If a strobe is programmed to be asserted in more than one set of entries (the SI Rx route for the TDMa entries and the SI Tx route for TDMb entries both select the same strobe), then the assertion of the strobe corresponds to the logical OR of all possible sources. This use of the strobes is not useful for most applications. It is recommended that a given strobe be selected in only one set of SI RAM entries.

Bits 6—Reserved

CSEL—Channel Select

000 = The bit/byte group is not supported by the MPC821. The transmit data pin is three-stated and the receive data pin is ignored.
001 = The bit/byte group is routed to SCC1.
010 = The bit/byte group is routed to SCC2.
011 = Reserved.
100 = Reserved.
101 = The bit/byte group is routed to SMC1.
110 = The bit/byte group is routed to SMC2.
111 = The bit/byte group is not supported by the MPC821. This code is also used in the SCIT mode as the D channel grant. Refer to **Section 16.12.7.2.2 SCIT Programming** for more information.

CNT—Count

This value indicates the number of bits/bytes (according to the BYT bit) that the routing and strobe select of this entry controls. If CNT = 0000, then 1 bit/byte is chosen; if CNT = 1111, then 16 bits/bytes are selected.

BYT—Byte Resolution

0 = Bit resolution–The CNT value indicates the number of bits in this group.
1 = Byte resolution–The CNT value indicates the number of bytes in this group.

LST—Last Entry in the RAM

Whenever the SI RAM is used, this bit must be set in one of the TX or RX entries of each group. Even if all entries of a group are used, this bit must still be set in the last entry.

0 = This is not the last entry in this section of the route RAM.
1 = This is the last entry in this RAM. After this entry, the SI waits for the sync signal to start the next frame.

Bits 16–31—Reserved

**16.12.4.6 SI RAM PROGRAMMING EXAMPLE.** This example shows how to program the RAM to support the 10-bit IDL bus. Refer to Figure 16-53 for the 10-bit IDL bus format. In this example, the TSA supports the B1 channel with SCC2, the D channel with SCC1, the first 4 bits of the B2 channel with an external device (using a strobe to enable the external device), and the last 4 bits of B2 with SCC4. Additionally, the TSA marks the D channel with another strobe signal.

First, divide the frame from the start (the sync) to the end of the frame according to the support that is required:

1. 8 bits (B1)—SCC2
2. 1 bit (D)—SCC1 + strobe1
3. 1 bit—no support
4. 4 bits (B2)—strobe2
5. 4 bits (B2)—SMC1
6. 1 bit (D)—SCC1 + strobe1

Each of these six divisions can be supported by just one SI RAM entry. Thus, a total of only six entries is needed in the SI RAM.

| ENTRY NO. | RAM WORD | | | | | | |
|---|---|---|---|---|---|---|---|
| | SWTR | SSEL | CSEL | CNT | BYT | LST | DESCRIPTION |
| 1 | 0 | 0000 | 010 | 0000 | 1 | 0 | 8 Bits SCC2 |
| 2 | 0 | 0001 | 001 | 0000 | 0 | 0 | 1 Bit SCC1 Strobe1 |
| 3 | 0 | 0000 | 000 | 0000 | 0 | 0 | 1 Bit No Support |
| 4 | 0 | 0010 | 000 | 0011 | 0 | 0 | 4 Bits Strobe2 |
| 5 | 0 | 0000 | 101 | 0011 | 0 | 0 | 4 Bits SMC1 |
| 6 | 0 | 0001 | 001 | 0000 | 0 | 1 | 1 Bit SCC1 Strobe1 |

**NOTE**

Since IDL requires the same routing for both receive and transmit, an exact duplicate of the above entries should be written to both the receive and transmit sections of the SI RAM. Then the CRTx bit in the SIMODE register can be used to instruct the SI RAM to use the same clock and sync to simultaneously control both sets of SI RAM entries.

**16.12.4.7 SI RAM DYNAMIC CHANGES.** The SI RAM has four operating modes:

1. A TDM with a static routing definition. SI RAM divided into two parts (RX and TX).
2. A TDM allowing dynamic changes. SI RAM divided into four parts.
3. Two TDMs with static routing definition. SI RAM divided into four parts.
4. Two TDMs allowing dynamic changes. SI RAM divided into eight parts.

Dynamic changes mean that the routing definition of a TDM can be modified while the SCCs and SMCs are connected to the TDM. With fixed routing, a change has three requirements that must be met before the new routing takes effect.

- All SCCs and SMCs connected to the TSA must be disabled
- The SI routing must be modified
- All SCCs and SMCs connected to the TSA must be reenabled

Dynamic changes divide portions of the SI RAM into current-route and shadow RAM. Once the current-route RAM is programmed, the TSA and SI channels are enabled, and TSA operation begins. When the user decides that a change in routing is required, the shadow RAM must be programmed with the new route and the CSRx bit in the SI CR must be set. As a result, the SI exchanges the shadow RAM and the current-route RAM as soon as the corresponding sync arrives and resets the CSRx bit to signify that the operation is complete. At this time, the user may change the routing again. Notice that the original current-route RAM is now the shadow RAM and vice versa. Figure 16-53 illustrates an example of the shadow RAM exchange process. If one TDM with dynamic changes is programmed, the initial current-route RAM addresses in the SI RAM are as follows.

| | |
|---|---|
| 0–63 | RXa Route |
| 128–191 | TXa Route |

The shadow RAMs are at addresses:

| | |
|---|---|
| 64–127 | RXa Route |
| 192–255 | TXa Route |

If two TDMs with dynamic changes are programmed, the initial current-route RAM addresses in the SI RAM are as follows:

| | |
|---|---|
| 0–31 | RXa Route |
| 64–95 | RXb Route |
| 128–159 | TXa Route |
| 192–223 | TXb Route |

The shadow RAMs are at addresses:

| | |
|---|---|
| 32–63 | RXa Route |
| 96–127 | RXb Route |
| 160–191 | TXa Route |
| 224–255 | TXb Route |

The user can read any RAM at any time, but for proper operation of the SI the user must not attempt to write the current-route RAM. The user can read the SI status register (SISTR) to find out which part of the RAM is the current-route RAM.

Beyond knowing which RAM is the current-route RAM, the user might need to know which entry the TSA is currently using within the current-route RAM. This information is provided in the SI RAM pointer register (SIRP). The user can also externally connect one of the four strobes to an interrupt pin to generate an interrupt on a particular SI RAM entry starting or ending execution by the TSA.

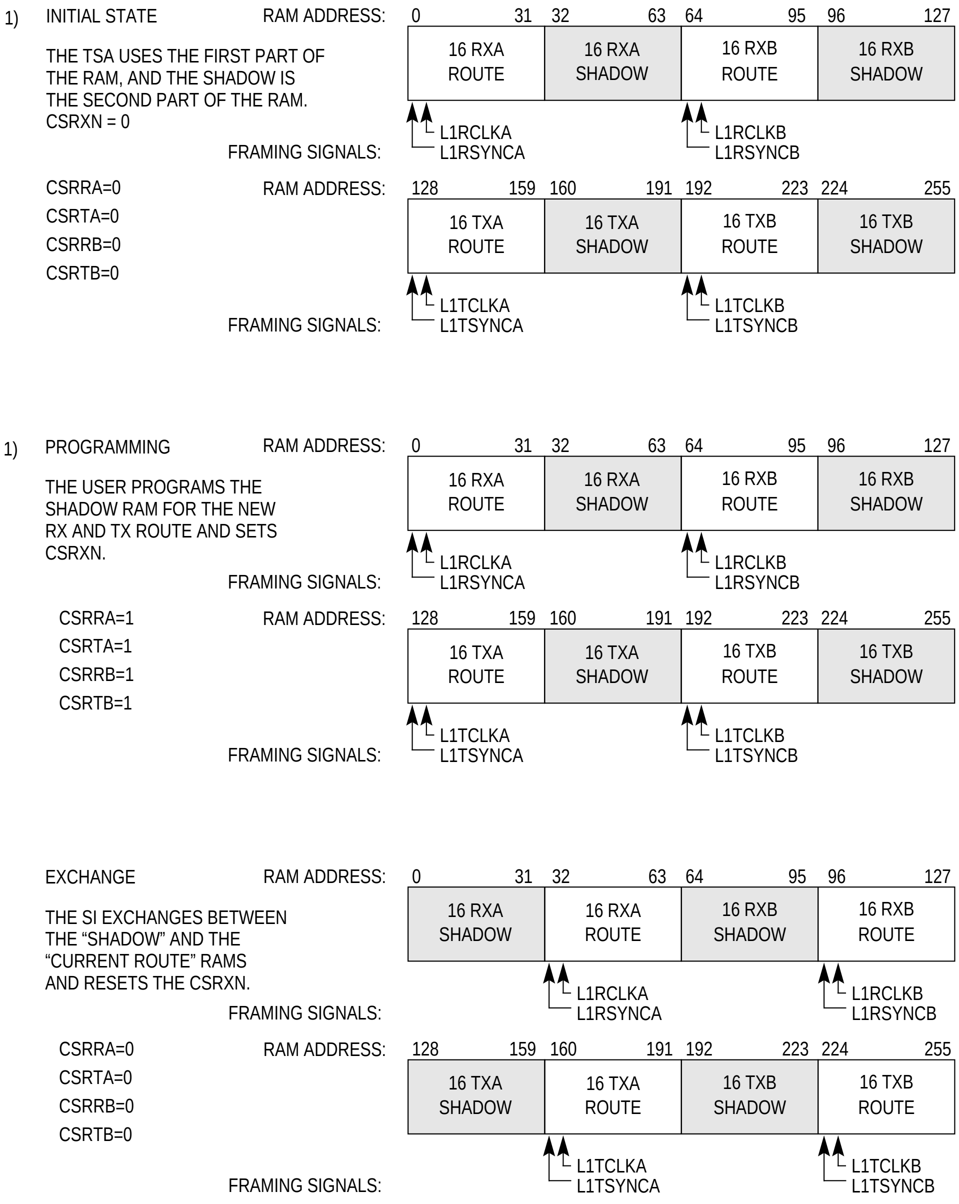


**Figure 16-53. SI RAM Dynamic Changes**

### 16.12.5 Serial Interface Registers

**16.12.5.1 SI GLOBAL MODE REGISTER.** The 8-bit SI global mode register (SIGMR) defines the RAM division modes and appears to the user as a memory-mapped, read/write register cleared at reset.

SIGMR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | ENB | ENA | RDM | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | AE4 | | | | | | | |

Bits 0–3—Reserved

ENb—Enable Channel b

0 = Channel b is disabled. The SI RAMs and TDM routing are in a state of reset, but all other SI functions still operate.
1 = The SI is enabled.

ENa—Enable Channel a

0 = Channel a is disabled. The SI RAMs and TDM routing are in a state of reset, but all other SI functions still operate.
1 = The SI is enabled.

RDM—RAM Division Mode

These bits define the RAM division mode and the number of multiplexed channels supported in the SI.

00 = The SI supports one TDM channel with 64 entries for receive routing and another 64 for transmit routing.
01 = The SI supports one TDM channel with 32 entries for receive routing and another 32 for transmit routing. There are an additional 32 shadow entries for the receive routing and 32 shadow entries for transmit routing that can be used to dynamically change the routing.
10 = The SI supports two TDM channels with 32 entries for the receive routing and another 32 for transmit routing for each of the two TDMs.
11 = The SI supports two TDM channels with 16 entries for receive routing and another 16 for transmit routing for each channel. There are an additional 16 shadow entries for receive routing and 16 shadow entries for transmit routing that can be used to dynamically change the channel routing.

**NOTE**

TSAa must be used in RDM1–0 if 00 or 01 setting is preferred.

**16.12.5.2 SI MODE REGISTER.** The 32-bit SI mode register (SIMODE) defines the SI operation modes and allows the user (with SI RAM) to support any or all of the ISDN channels independently when in IDL or GCI mode. Any extra SCC channel can then be used for other purposes in NMSI mode. SIMODE appears to the user as a memory-mapped, read/write register cleared at reset.

SIMODE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | SMC2 | SMC2CS | | | SDMB | | RFSDB | | DSCB | CRTB | STZB | CEB | FEB | GMB | TFSDB | |
| 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 | AE0 | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | SMC1 | SMC1CS | | | SDMA | | RFSDA | | DSCA | CRTA | STZA | CEA | FEA | GMA | TFSDA | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | AE2 | | | | | | | | | | | | | | | |

SMCx—SMCx Connection

0 = NMSI mode. The clock source is determined by the SMCxCS bit and the data comes from a dedicated pin (SMTXD1 and SMRXD1 for SMC1 or SMTXD2 and SMRXD2 for SMC2) in the NMSI.

1 = SMCx is connected to the multiplexed SI (TDM channel).

SMC2CS—SMC2 Clock Source (NMSI mode)

SMC2 can take its clocks from one of the baud rate generators or one of four pins from the bank of clocks. However, the SMC2 transmit and receive clocks must be the same when it is connected to the NMSI.

000 = SMC2 transmit and receive clocks are BRG1.
001 = SMC2 transmit and receive clocks are BRG2.
010 = SMC2 transmit and receive clocks are BRG3.
011 = SMC2 transmit and receive clocks are BRG4.
100 = SMC2 transmit and receive clocks are CLK5.
101 = SMC2 transmit and receive clocks are CLK6.
110 = SMC2 transmit and receive clocks are CLK7.
111 = SMC2 transmit and receive clocks are CLK8.

SMC1CS—SMC1 Clock Source (NMSI mode)

SMC1 can take its clocks from one of the baud rate generators or one of four pins from the bank of clocks. However, the SMC1 transmit and receive clocks must be the same when it is connected to the NMSI.

000 = SMC1 transmit and receive clocks are BRG1.
001 = SMC1 transmit and receive clocks are BRG2.
010 = SMC1 transmit and receive clocks are BRG3.
011 = SMC1 transmit and receive clocks are BRG4.
100 = SMC1 transmit and receive clocks are CLK1.
101 = SMC1 transmit and receive clocks are CLK2.
110 = SMC1 transmit and receive clocks are CLK3.
111 = SMC1 transmit and receive clocks are CLK4.

SDMx—SI Diagnostic Mode for TDM A or B

00 = Normal operation.
01 = Automatic echo. In this mode, the channel_x transmitter automatically retransmits the TDM received data on a bit-by-bit basis. The receive section operates normally, but the transmit section can only retransmit received data. In this mode, the L1GRx line is ignored.
10 = Internal loopback. In this mode, the TDM transmitter output is internally connected to the TDM receiver input (L1TXDx is connected to L1RXDx). The receiver and transmitter operate normally. The data appears on the L1TXDx pin and in this mode, the L1RQx line is asserted normally. The L1GRx line is ignored.
11 = Loopback control. In this mode, the TDM transmitter output is internally connected to the TDM receiver input (L1TXDx is connected to L1RXDx). The transmitter output (L1TXDx) and the L1RQx pin is inactive. This mode is used to accomplish loopback testing of the entire TDM without affecting the external serial lines.

**NOTE**

In modes 01,10, and 11, the receive and transmit clocks should be identical.

RFSDx—Receive Frame Sync Delay for TDM A or B

These two bits determine the number of clock delays between the receive sync and the first bit of the receive frame. Even if the CRTx bit is set, these bits do not control the delay for the transmit frame.

00 = No bit delay. The first bit of the frame is transmitted/received on the same clock as the sync; use for GCI.
01 = 1-bit delay. Use for IDL.
10 = 2-bit delay.
11 = 3-bit delay.

Refer to Figure 16-54 and Figure 16-55 for an example of the use of these bits.

DSCx—Double Speed Clock for TDM A or B

Some TDMs, such as GCI, define the input clock to be 2× faster than the data rate and this bit controls this option.

0 = The channel clock (L1RCLKx and/or L1TCLKx) is equal to the data clock. Use for IDL and most TDM formats.
1 = The channel clock rate is twice the data rate. Use for GCI.

CRTx—Common Receive and Transmit Pins for TDM A or B

This bit is useful when the transmit and receive sections of a given TDM use the same clock and sync signals. In this mode, L1TCLKx and L1TSYNCx pins can be used as general-purpose I/O pins.

0 = Separate pins. The receive section of this TDM uses L1RCLKx and L1RSYNCx pins for framing and the transmit section uses L1TCLKx and L1TSYNCx for framing.
1 = Common pins. The receive and transmit sections of this TDM use L1RCLKx as clock pin of channel x and L1RSYNCx as the receive and transmit sync pin. Use for IDL and GCI. RSFD and TSFD are independent of one another in this mode.

STZx—Set L1TXDx to Zero for TDM A or B

0 = Normal operation.
1 = L1TXDx is set to zero until serial clocks are available, which is useful for GCI activation. Refer to **Section 16.12.7.1 SI GCI Activation/Deactivation Procedure** for details.

CEx—Clock Edge for TDM A or B

When DSCx = 0

0 = The data is transmitted on the rising edge of the clock and received on the falling edge (use for IDL and GCI).
1 = The data is transmitted on the falling edge of the clock and received on the rising edge.

When DSCx = 1

0 = The data is transmitted on the rising edge of the clock and received on the rising edge.
1 = The data is transmitted on the falling edge of the clock and received on the falling edge.

FEx—Frame Sync Edge for TDM A or B

The L1RSYNCx and L1TSYNCx pulses are sampled with the falling/rising edge of the channel clock according to this bit.

0 = Falling edge. Use for IDL and GCI.
1 = Rising edge.

GMx—Grant Mode for TDM A or B

0 = GCI/SCIT mode. The GCI/SCIT D channel grant mechanism for transmission is internally supported. The grant is one bit from the receive channel. This bit is marked by programming the channel select bits of the SI RAM with 111 to assert an internal strobe on it. For details, see **Section 16.12.7.2.2 SCIT Programming**.

1 = IDL mode. A GRANT mechanism is supported if the corresponding GR1–GR4 bits in the SICRn are set. The grant is a sample of the L1GRx pin while L1TSYNCx is asserted. This GRANT mechanism implies the IDL access controls for transmission on the D channel. Refer to **Section 16.12.6.2 IDL Interface Programming** for more information.

TFSDx—Transmit Frame Sync Delay for TDM A or B

These two bits determine the number of clock delays between the transmit sync and the first bit of the transmit frame.

00 = No bit delay. The first bit of the frame is transmitted/received on the same clock as the sync.

01 = 1-bit delay.

10 = 2-bit delay.

11 = 3-bit delay.

Refer to the figures below for an example of the use of these bits.

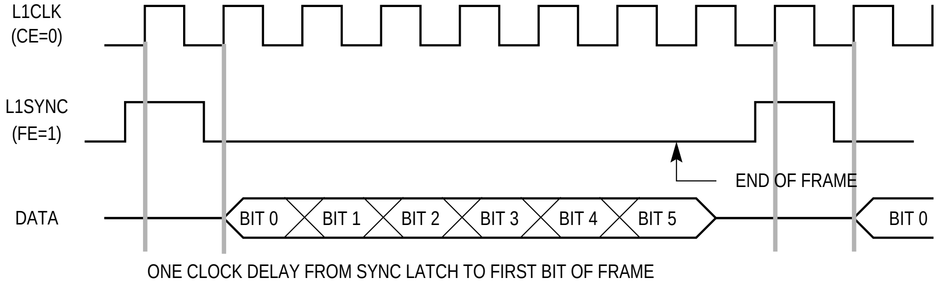


**Figure 16-54. One Clock Delay from Sync to Data (RFSD = 01)**

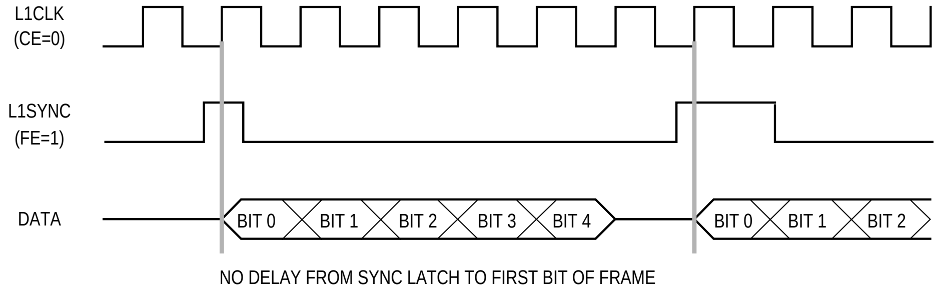


**Figure 16-55. No Delay from Sync to Data (RFSD = 00)**

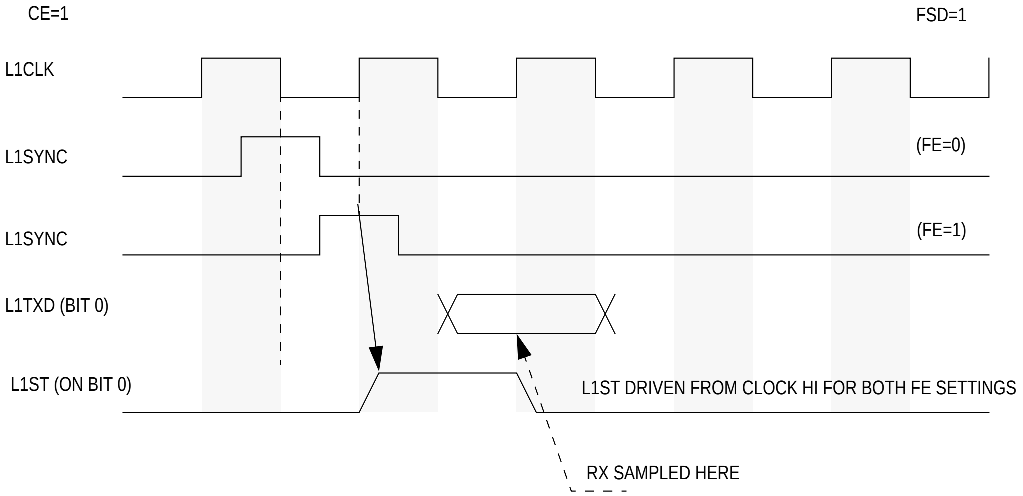


**Figure 16-56. Clock Edge (CE) Effect When DSC = 0**

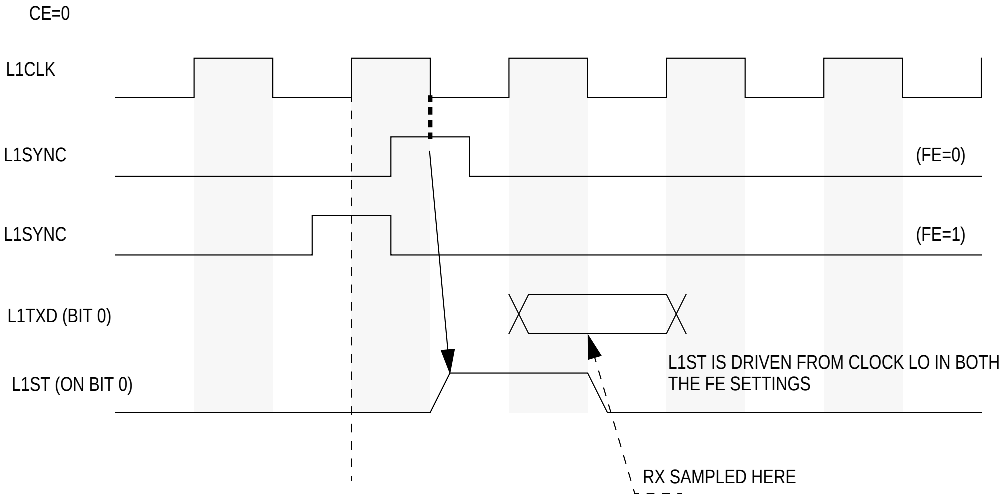


**Figure 16-57. Clock Edge (CE) Effect When DSC = 1**

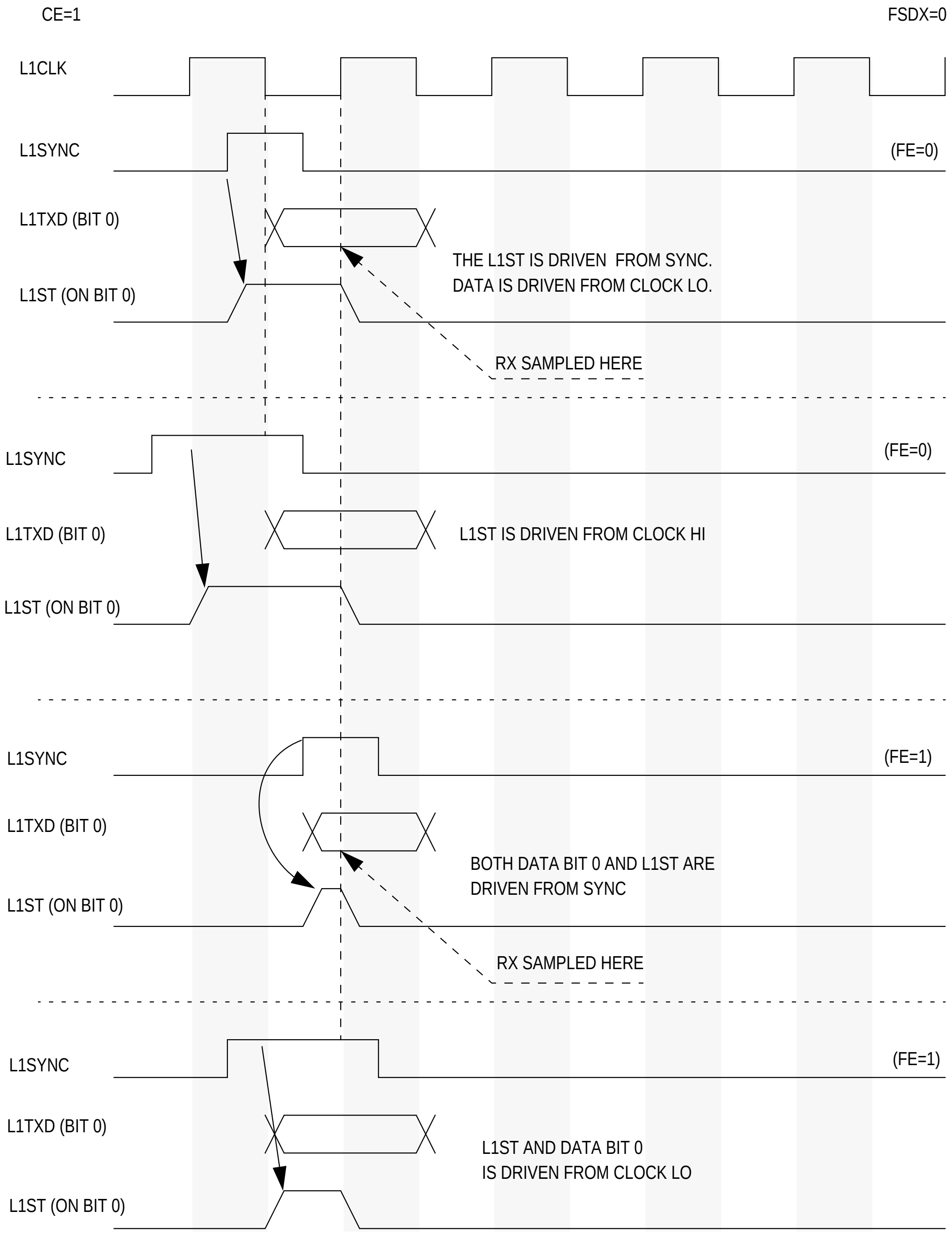


**Figure 16-58. Frame Transmission Reception When RFSDx or TFSDx = 0 and CD = 1**

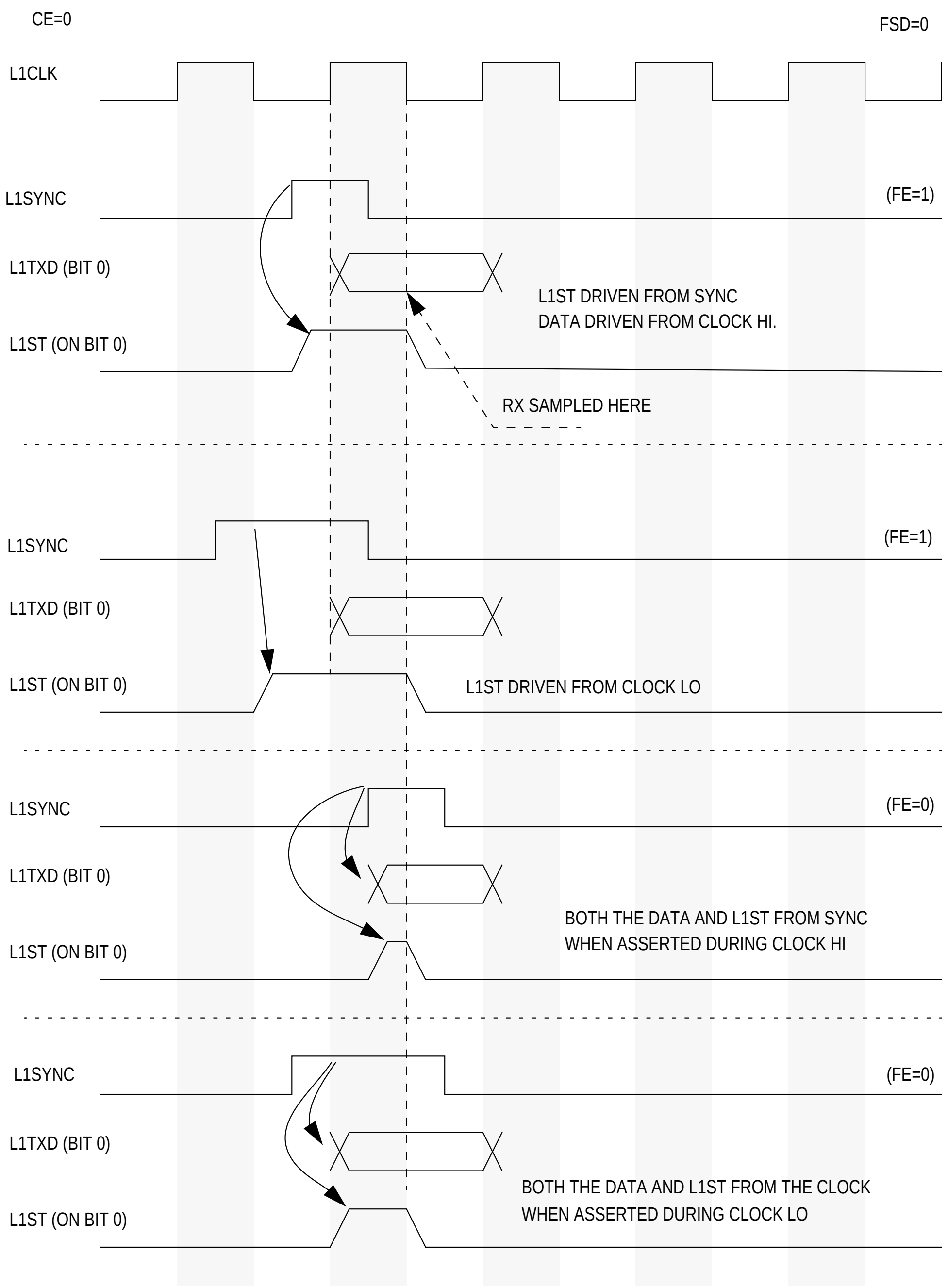


**Figure 16-59. CEx = 0 and FEx Interaction, XFSD = 0**

**16.12.5.3 SI CLOCK ROUTE REGISTER.** The 32-bit SI clock route register (SICR) is used to define the SCC clock sources that can be one of the four baud rate generators or an input from a bank of clock pins. The SICR appears to the user as a memory-mapped, read/write register cleared at reset.

SICR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | | | | | | | | | | | | | | | | |
| 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 | AEC | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | GR2 | SC2 | R2CS | | | T2CS | | | GR1 | SC1 | R1CS | | | T1CS | | |
| 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 | AEE | | | | | | | | | | | | | | | |

GRx—Grant Support of SCCx

0 = SCCx transmitter does not support the grant mechanism. The grant is always asserted internally.

1 = SCCx transmitter supports the grant mechanism as determined by the GMx bit of it's channel.

SCx—SCCx Connection

0 = SCCx is not connected to the multiplexed SI but is either connected directly to the NMSIx pins or is not used. The choice of general-purpose I/O port pins versus SCCn pins is made in the parallel I/O control register.

1 = SCCx is connected to the multiplexed SI. The NMSIx receive pins are available for other purposes.

R1CS,R2CS—Receive Clock Source for SCCx

These bits are ignored when the SCCx is connected to the TSA (SCx = 1).

000 = SCCx receive clock is BRG1.
001 = SCCx receive clock is BRG2.
010 = SCCx receive clock is BRG3.
011 = SCCx receive clock is BRG4.
100 = SCCx receive clock for x = 1,2 is CLK1.
101 = SCCx receive clock for x = 1,2 is CLK2 .
110 = SCCx receive clock for x = 1,2 is CLK3 .
111 = SCCx receive clock for x = 1,2 is CLK4 .

T1CS,T2CS—Transmit Clock Source for SCCx

These bits are ignored when SCCx is connected to the TSA (SCx = 1).

000 = SCCx transmit clock is BRG1.
001 = SCCx transmit clock is BRG2.
010 = SCCx transmit clock is BRG3.
011 = SCCx transmit clock is BRG4.
100 = SCCx transmit clock for x = 1, 2 is CLK1.
101 = SCCx transmit clock for x = 1, 2 is CLK2.
110 = SCCx transmit clock for x = 1, 2 is CLK3.
111 = SCCx transmit clock for x = 1, 2 is CLK4.

**16.12.5.4 SI COMMAND REGISTER.** The 8-bit SI command register (SICMR) allows the user to dynamically program the SI RAM. For more information about dynamic programming, refer to **Section 16.12.4.7 SI RAM Dynamic Changes**. The contents of this register are valid only in the RAM division mode (the RDM1–RDM0 bits in the SICMR equal 01 or 11) and this register is cleared at reset.

SICMR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | CSRRA | CSRTA | CSRRB | CSRTB | RESERVED | | | |
| RESET | | | | | | | | |
| R/W | | | | | | | | |
| ADDR | AE7 | | | | | | | |

CSRRx—Change Shadow RAM for TDM A or B Receiver

When set, this bit causes the SI receiver to replace the current route with the shadow RAM. The bit is set by the user and cleared by the SI.

0 = The receiver shadow RAM is not valid. The user can write into the shadow RAM to program a new routing.
1 = The receiver shadow RAM is valid. The SI exchanges between the RAMs and take the new receive routing from the receiver shadow RAM. This bit is cleared as soon as the switch has completed.

CSRTx—Change Shadow RAM for TDM A or B Transmitter

When set, this bit causes the SI transmitter to replace the current route with the shadow RAM. The bit is set by the user and cleared by the SI.

0 = The transmitter shadow RAM is not valid. The user can write into the shadow RAM to program a new routing.
1 = The transmitter shadow RAM is valid. The SI exchanges between the RAMs and take the new transmitter routing from the receiver shadow RAM. This bit is cleared as soon as the switch has completed.

Bits 4–7—Reserved

These bits should be set to zero by the user.

**16.12.5.5 SI STATUS REGISTER.** The 8-bit SI status register (SISTR) indicates to the user which part of the SI RAM is the current-route RAM. The value of this register is valid only when the corresponding bit in the SIGMR is clear. This register is cleared at reset.

SISTR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | CRORA | CROTA | CRORB | CROTB | RESERVED | | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R | R |
| ADDR | AE6 | | | | | | | |

CRORa—Current Route of TDMa Receiver

0 = The current-route receiver RAM is in address:
0–63 when the SI supports one TDM (RDM = 01).
0–31 when the SI supports two TDMs (RDM = 11).
1 = The current route receiver RAM is in address:
64–127 when the SI supports one TDM (RDM = 01).
32–63 when the SI supports two TDMs (RDM = 11).

CROTa—Current Route of TDMa Transmitter

0 = The current-route transmitter RAM is in address:
128–191 when the SI supports one TDM (RDM = 01).
128–159 when the SI supports two TDMs (RDM = 11).
1 = The current-route transmitter RAM is in address:
192–255 when the SI supports one TDM (RDM = 01).
160–191 when the SI supports two TDMs (RDM = 11).

CRORb—Current Route of TDMb Receiver

This bit is only valid in the RAM division mode (RDM bits in the SIGMR equal 11).

0 = The current-route receiver RAM is in address 64–95.
1 = The current-route receiver RAM is in address 96–127.

CROTb—Current Route of TDMb Transmitter

This bit is only valid in the RAM division mode (RDM bits in the SIGMR equal 11).

0 = The current-route transmitter RAM is in address 192–223.
1 = The current-route transmitter RAM is in address 224–255.

Bits 4–7—Reserved

**16.12.5.6 SI RAM POINTERS.** This 32-bit, read-only register indicates to the user which RAM entry is currently being serviced. This gives a real-time status of where the SI currently is inside the TDM frame.

Although the SI RAM pointer (SIRP) register does not need to be accessed by most users, it does provide information that may be helpful for debugging and synchronization of some system activity to the TDMs' activity. Reading the SISTR should be sufficient for most applications.

The user can determine which RAM entry in the SI RAM is currently in progress, but cannot determine the status within that entry. For instance, if the RAM entry is programmed to select four contiguous time-slots from the TDM and the SIRP register indicates the entry is currently active, the user does not know which of the four time-slots is currently in progress. The SIRP register does, however, change its status immediately when the next SI RAM entry begins processing.

**NOTE**

> The user can externally connect one of the four strobes to an interrupt pin to generate an interrupt on a particular SI RAM entry starting or ending execution by the TSA.

The value of this register is changed on transitions of the serial clocks. Before acting on the information in this register, the user should perform two reads and verify that they returned the same value.

The pointers provided by this register indicate the SI RAM entry word offset that is currently in progress. The register is cleared at reset.

SIRP REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | $V_{TB}$ | TBPTR | | | | | — | — | $V_{TA}$ | TAPTR | | | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| ADDR | AF0 | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | — | — | $V_{RB}$ | RBPTR | | | | | — | — | $V_{RA}$ | RAPTR | | | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| ADDR | AF2 | | | | | | | | | | | | | | | |

In all cases, the value in the TxPTR or RxPTR increments by one for each entry (16-bit SI RAM word) that the SI processes. Since each TxPTR and RxPTR is 5 bits each, the values in each TxPTR and RxPTR can range from 0 to 31, corresponding to 32 different SI RAM entries. The full pointer range may not necessarily be used. For instance, if the last bit is set in the fifth SI RAM entry, then the pointer only reflects values from 0 to 4, but once the fifth entry is processed by the SI, the pointer is reset to 0.

The V-bit in each entry shows that the entry is valid. This information is particularly useful if the PTR value happens to be zero. Additionally, the V-bits save the user from having to read both the SIRP and the SISTRs to obtain the needed information. The pointer values are described based on the four possible ways of configuring the SI RAM.

**16.12.5.6.1 SIRP When RDM = 00.** In this case, since 64 entries cannot be signified with a single 5-bit pointer, two 5-bit pointers are used—one for the first 32 entries and one for the second 32 entries.

- RaPTR and RbPTR contain the address of the currently active RAM entry. When the SI services entries 1–32, RaPTR is incremented and RbPTR is continuously cleared. When the SI services entries 33–64, RaPTR is continuously cleared and RbPTR is incremented.
- TaPTR and TbPTR contain the address of the currently active Tx entry. When the SI services entries 1–32, TaPTR is incremented and TbPTR is continuously cleared. When the SI services entries 33–64, TaPTR is continuously cleared and TbPTR is incremented.

**16.12.5.6.2 SIRP When RDM = 01.** For the receiver, either RaPTR or RbPTR is used, depending on which portion of the SI Rx RAM is currently active. For the transmitter, either TaPTR or TbPTR is used, depending on which portion of the SI Tx RAM is currently active.

- If it’s V-bit is set, RaPTR contains the address of the currently active Rx entry. The SI RAM receive address block in use is 0–63 and CRORa = 0 in the SISTR.
- If it’s V-bit is set, RbPTR contains the address of the currently active Rx entry. The SI RAM receive address block in use is 64–127 and CRORa = 1 in the SISTR.
- If it’s V-bit is set, TaPTR contains the address of the currently active Tx entry. The SI RAM transmit address block in use is 128–191 and CROTa = 0 in the SISTR.
- If it’s V-bit is set, TbPTR contains the address of the currently active Tx entry. The SI RAM transmit address block in use is 192–255 and CROTa = 1 in the SISTR.

**16.12.5.6.3 SIRP When RDM = 10.** This is the simplest case, since each pointer is used continuously and only has one function.

- RaPTR contains the address of the currently active RXa entry.
- RbPTR contains the address of the currently active RXb entry.
- TaPTR contains the address of the currently active TXa entry.
- TbPTR contains the address of the currently active TXb entry.

**16.12.5.6.4 SIRP When RDM = 11.** In this case, each pointer is used continuously, but points to different sections of the SI RAM, depending on whether the pointer's value is in the first half (0–15) or the second half (16–31).

- RaPTR contains the address of the currently active RXa entry. If the pointer has a value from 0–15, the current-route RAM is SI RAM address block 0–31 and CRORa = 0 in the SISTR. If the pointer has a value from 16–31, the current-route RAM is SI RAM address block 32–63 and CRORa = 1 in the SISTR.
- RbPTR contains the address of the currently active RXb entry. If the pointer has a value from 0–15, the current route RAM is SI RAM address block 64–95 and CRORb = 0 in the SISTR. If the pointer has a value from 16–31, the current-route RAM is SI RAM address block 96–127 and CRORb = 1 in the SISTR.
- TaPTR contains the address of the currently active TXa entry. If the pointer has a value from 0–15, the current route RAM is SI RAM address block 128–159 and CROTa = 0 in the SISTR. If the pointer has a value from 16–31, the current-route RAM is SI RAM address block 160–191 and CROTa = 1 in the SISTR.
- TbPTR contains the address of the currently active TXb entry. If the pointer has a value from 0–15, the current-route RAM is SI RAM address block 192–223 and CROTb = 0 in the SISTR. If the pointer has a value from 224–255, the current-route RAM is SI RAM address block 160–191 and CROTb = 1 in the SISTR.

## 16.12.6 Serial Interface IDL Interface Support

The IDL interface is a full-duplex ISDN interface used to connect a physical layer device to the MPC821. The MPC821 supports both the basic and primary rate of the IDL bus. In the basic rate of IDL, data on three channels (B1, B2, and D) is transferred in a 20-bit frame, providing 160-kbps full-duplex bandwidth. The MPC821 is an IDL slave device that is clocked by the IDL bus master (physical layer device) and has separate receive and transmit sections. Because the MPC821 can support two TDMs, it can actually support two independent IDL buses using separate clocks and sync pulses as illustrated in Figure 16-60.

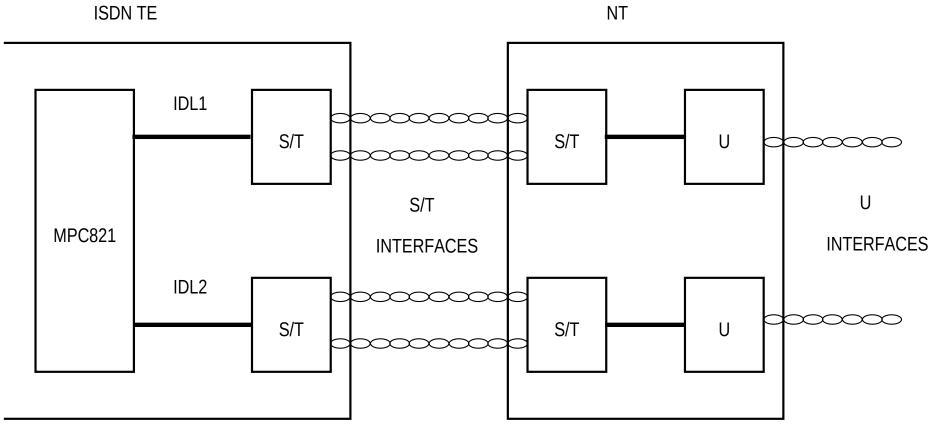


**Figure 16-60. Dual IDL Bus Application Example**

**16.12.6.1 IDL INTERFACE EXAMPLE.** An example of the IDL application is the ISDN terminal adaptor illustrated in Figure 16-61. In such an application, the IDL interface is used to connect the 2B+D channels between the MPC821, CODEC, and S/T transceiver. One of the MPC821 SCCs is configured to HDLC mode to handle the D channel; another MPC821 SCC is used to rate adapt the terminal datastream over the first B channel. That SCC is configured for HDLC mode if V.120 rate adaption is required. The second B channel is then routed to the CODEC as a digital voice channel, if preferred. The SPI is used to send initialization commands and periodically check status from the S/T transceiver. The SMC connected to the terminal is configured for UART.

The MPC821 can identify and support each IDL channel or can output strobe lines for interfacing devices that do not support the IDL bus. The IDL signals for each transmit and receive channel are as follows:

1. L1RCLKx—IDL clock; input to the MPC821.
2. L1RSYNCx—IDL sync signal; input to the MPC821. This signal indicates that the clock periods following the pulse designate the IDL frame.
3. L1RXDx—IDL receive data; input to the MPC821. Valid only for the bits that are supported by the IDL; ignored for other signals that may be present.
4. L1TXDx—IDL transmit data; output from the MPC821. Valid only for the bits that are supported by the IDL; otherwise, three-stated.
5. L1RQx—IDL request permission to transmit on the D channel; output from the MPC821 on the L1RQx pin.
6. L1GRx—IDL grant permission to transmit on the D Channel; input to the MPC821 on the L1TSYNCx pin.

**NOTE**

x = a and b for TDMa and TDMb.

The basic rate IDL bus has three channels:

- B1 is a 64 kbps bearer channel
- B2 is a 64 kbps bearer channel
- D is a 16 kbps signaling channel

There are two definitions of the IDL bus frame structure—8 and 10 bits. The only difference between them is the channel order within the frame. Refer to Figure 16-62 for details.

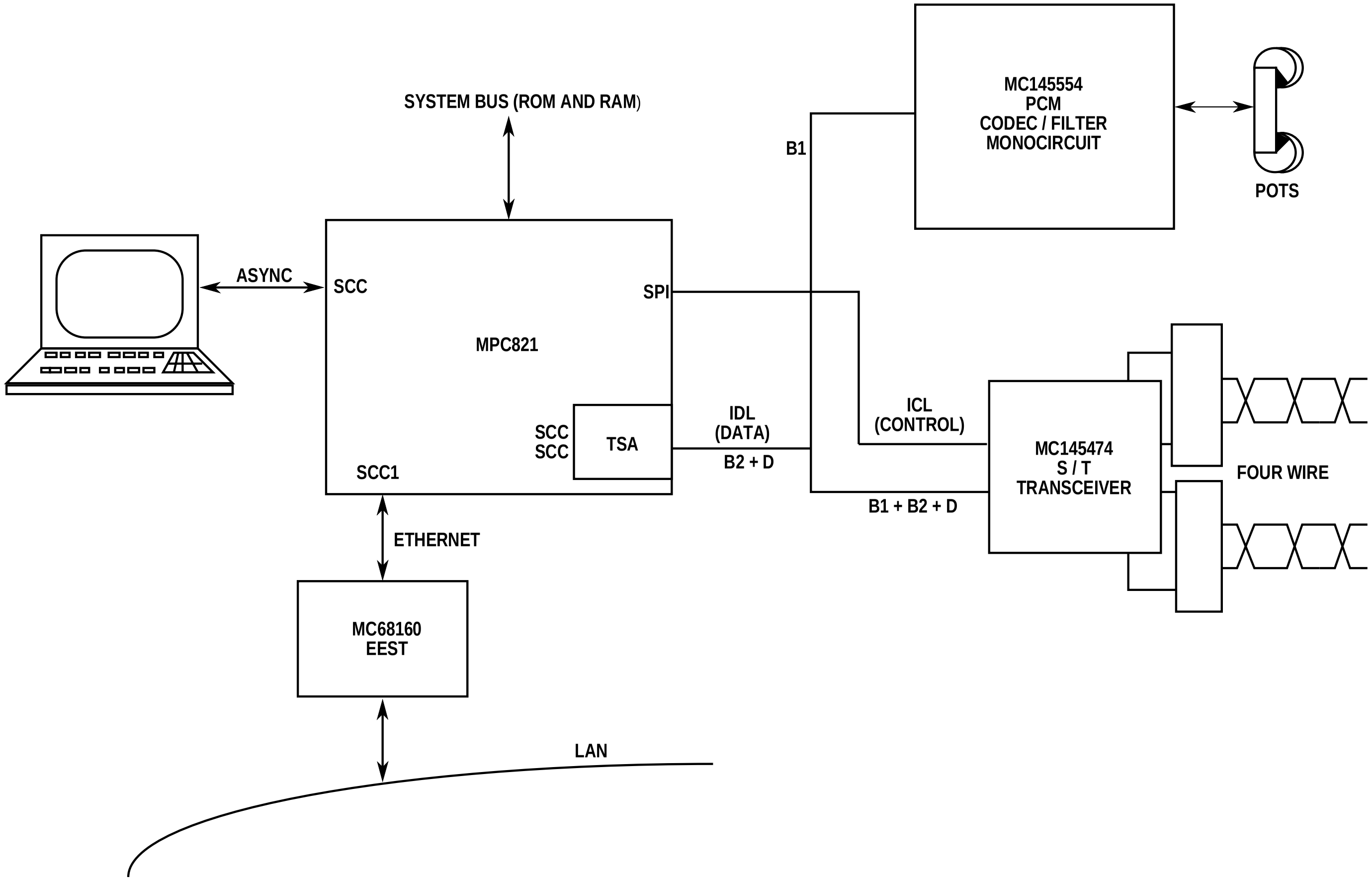
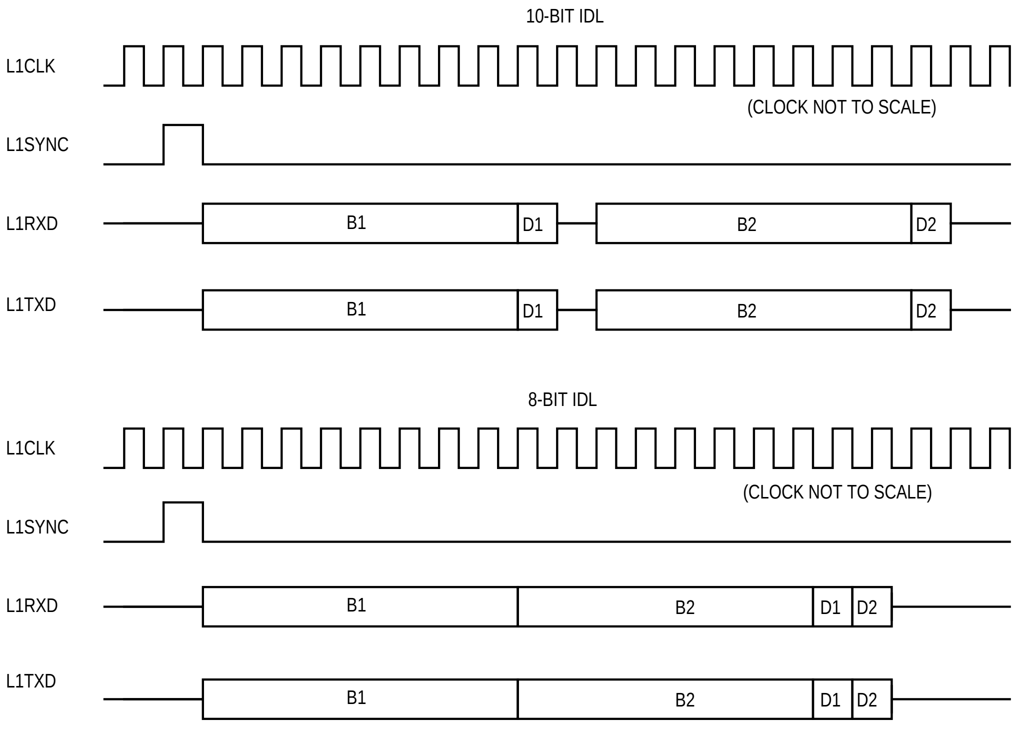


Figure 16-61. IDL Terminal Adaptor



NOTE: L1RQN AND L1GRN ARE NOT SHOWN.

**Figure 16-62. IDL Bus Signals**

### NOTE

Previous versions of Motorola IDL-defined bit functions, called auxiliary (A) and maintenance (M), were eliminated from the IDL definition when it was concluded that the IDL control channel would be out-of-band. They were defined as a subset of the Motorola SPI format called serial control port (SCP). If a user prefers to implement the A and M bit functions as originally defined, the TSA can be programmed to access these bits and route them transparently to an SCC or SMC. To perform the out-of-band signaling required, the MPC821 SPI can be used.

The MPC821 supports all channels of the IDL bus in the basic rate. Each bit in the IDL frame can be routed to every SCC and SMC or can assert a strobe output for supporting an external device. The MPC821 supports the request-grant method for contention detection on the D channel of the IDL basic rate and when the MPC821 has data to transmit on the D channel, it asserts L1RQx. The physical layer device monitors the physical layer bus for activity on the D channel and indicates that the channel is free by asserting L1GRx. The MPC821 samples the L1GRx signal when the IDL sync signal (L1RSYNCx) is asserted. If L1GRx is high (active), the MPC821 transmits the first zero of the opening flag in the first bit of the D channel. If a collision is detected on the D channel, the physical layer device negates L1GRx. The MPC821 then stops it’s transmission and retransmits the frame when L1GRx is reasserted. This procedure is handled automatically for the first two buffers of a frame.

For the primary rate IDL, the MPC821 supports up to four 8-bit channels in the frame, determined by the SI RAM programming. To support more channels, the user can route more than one channel to every SCC and the SCC will treat it as one high-speed stream and store it in the same data buffers (this approach is appropriate only for transparent data). Additionally, the MPC821 can be used to assert strobes for support of additional external IDL channels.

The IDL interface supports the CCITT I.460 recommendation for data rate adaptation since it separately accesses each bit of the IDL bus. The current-route RAM specifies which bits are supported by the IDL interface and by which serial controller. The receiver only receives the bits that are enabled by the receiver route RAM. Otherwise, the transmitter only transmits the bits that are enabled by the transmitter route RAM and three-states L1TXDx.

**16.12.6.2 IDL INTERFACE PROGRAMMING.** The user can program the channels used for the IDL bus interface to the appropriate configuration. First, the user should program the SIMODE to the IDL grant mode for that channel, using the GMx bits. More than one channel can be programmed to interface with the IDL bus. If the receive and transmit section are used for interfacing to the same IDL bus, the user can internally connect the receive clock and sync signals to the SI RAM transmit section, using the CRTx bits. The RAM section used for the IDL channels must be programmed to the preferred routing. Refer to **Section 16.12.4.6 SI RAM Programming Example** for more information.

The user should then define the IDL frame structure to be a delay of 1-bit from frame sync to data, to falling edge sample sync, and the clock edge to transmit on the rising edge of the clock. The L1TXDx pin should be programmed to be three-stated when inactive (through the parallel I/O open-drain register). To support the D channel, the user must program the appropriate GRx bit in the SIMODE register and program the RAM entry to route data to that serial controller. The two definitions of IDL, 8 and 10 bits, are only supported by modifying the SI RAM programming. In both cases, the L1GRx pin is sampled with the L1TSYNCx signal and transferred to the D channel SCC as a grant indication. The same procedure is valid for supporting an IDL bus in the second channel.

For example, assuming the **Section 16.12.4.6 SI RAM Programming Example**, which uses SCC1, SCC2, and SMC1 connected to the TDMx pins, with no other SCCs connected, the initialization sequence is as follows:

1. Program the SI RAM. Write all entries that are not used with $0001, set the LST bit, and disable the routing function.

| ENTRY NO. | RAM WORD | | | | | | |
|---|---|---|---|---|---|---|---|
| | SWTR | SSEL | CSEL | CNT | BYT | LST | DESCRIPTION |
| 1 | 0 | 0000 | 010 | 0000 | 1 | 0 | 8 Bits SCC2 |
| 2 | 0 | 0000 | 001 | 0000 | 0 | 0 | 1 Bit SCC1 |
| 3 | 0 | 0000 | 000 | 0000 | 0 | 0 | 1 Bit No Support |
| 4 | 0 | 0000 | 101 | 0000 | 1 | 0 | 8 Bits SMC1 |
| 5 | 0 | 0001 | 001 | 0000 | 0 | 1 | 1 Bit SCC1 Strobe1 |

**NOTE**

Since IDL requires the same routing for both receive and transmit, an exact duplicate of the above entries should be written to both the receive and transmit sections of the SI RAM beginning at SI RAM addresses 0 and 128, respectively.

2. SIMODE = $00008145. Only TDMa is used; the SMC1 is connected to the TSA.
3. SICR = $000040C0. SCC2 and SCC1 are connected to the TSA. SCC1 supports the grant mechanism since it is on the D channel.
4. PAODR bit 9 = 1. Configures L1TXDa to an open-drain output.
5. PAPAR bits 9, 8, and 7 = 1. Configures L1TXDa, L1RXDa, and L1RCLKa.
6. PADIR bits 9 and 8 = 1. PADIR bit 7 = 0. Configures L1TXDa, L1RXDa, and L1RCLKa.
7. PCPAR bits 12, 5, and 11 = 1. Configures L1RQa, L1TSYNCa, and L1RSYNCa.
8. PCDIR bit 12 = 0. L1RQa is an input. L1TSYNCa performs the L1GRa function and is therefore an output, but it does not need to be configured with PCDIR bit 5 = 0. L1RSYNCa is an input, but it does not need to be configured with a PCDIR bit.
9. SIGMR = $04. Enable TDMa (one static TDM).
10. SICMR is not used.
11. SISTR and SIRP do not need to be read, but can be used for debugging information once the channels are enabled.
12. Enable the SCC1 for HDLC operation (to handle the LAPD protocol of the D channel), and set SCC2 and SMC1 as preferred.

### 16.12.7 Serial Interface GCI Support

The normal mode of the GCI (also known as the ISDN-oriented modular rev 2.2 (IOM-2)) and the SCIT are fully supported by the MPC821. The MPC821 also supports the D channel access control in S/T interface terminals by using the command/indication (C/I) channel for that function.

The GCI bus consists of four lines—two data lines, a clock, and a frame synchronization line. Usually, an 8-kHz frame structure defines the various channels within the 256-kbps data rate. The MPC821 supports two independent GCI busses and has independent receive and transmit sections for each one. The interface can also be used in a multiplexed frame structure on which up to eight physical layer devices multiplex their GCI channels. In this mode, the data rate would be 2,048 kbps.

In the GCI bus, the clock rate is twice the data rate. The SI divides the input clock by two to produce the data clock. The MPC821 also has data strobe lines and the 1× data rate clock L1CLKOx output pins. These signals are used for interfacing devices to GCI that do not support the GCI bus. The GCI signals for each transmit and receive channel are as follows:

L1RSYNCx—Used as a GCI sync signal; input to the MPC821. This signal indicates that the clock periods following the pulse designate the GCI frame.

L1RCLKx—Used as a GCI clock; input to the MPC821. The L1RCLKx signal is twice the data clock.

L1RXDx—Used as a GCI receive data; input to the MPC821.

L1TXDx—Used as a GCI transmit data; open-drain output. Valid only for the bits that are supported by the IDL; otherwise, three-stated.

L1CLKOx—Optional signal; output from the MPC821. This 1× clock output is used to clock devices that do not interface directly to the GCI. If the double-speed clock is used, (DSCx bit is set in the SIMODE), this output is the L1RCLKx divided by 2; otherwise, it is simply a 1× output of the L1RCLKx signal.

**NOTE**

x = a and b for TDMa and TDMb.

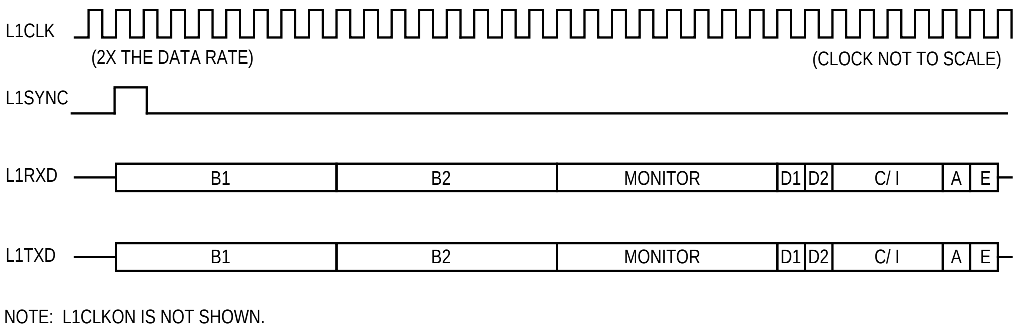


**Figure 16-63. GCI Bus Signals**

In addition to the 144-kbps ISDN 2B+D channels, the GCI provides five channels for maintenance and control functions:

- B1 is a 64 kbps bearer channel
- B2 is a 64 kbps bearer channel
- M is a 64 kbps monitor (M) channel
- D is a 16 kbps signaling channel
- C/I is a 48 kbps C/I channel (includes A and E bits)

The M channel is used to transfer data between layer 1 devices and the control unit (the CPU) and the C/I channel is used to control activation/deactivation procedures or to switch test loops by the control unit. The M and C/I channels of the GCI bus should be routed to SMC1 or SMC2, which have modes to support the channel protocols. The MPC821 can support any channel of the GCI bus in the primary rate by modifying the SI RAM programming.

The GCI supports the CCITT I.460 recommendation as a method for data rate adaptation since it can access each bit of the GCI separately. The current-route RAM specifies which bits are supported by the interface and by which serial controller. The receiver only receives the bits that are enabled by the SI RAM and the transmitter only transmits the bits that are enabled by the SI RAM and does not drive L1TXDx. Otherwise, L1TXDx is an open-drain output and should be pulled high externally.

The MPC821 supports contention detection on the D channel of the SCIT bus. When the MPC821 has data to transmit on the D channel, it checks a SCIT bus bit that is marked with a special route code (usually, bit 4 of C/I channel 2). The physical layer device monitors the physical layer bus for activity on the D channel and indicates on this bit that the channel is free. If a collision is detected on the D channel, the physical layer device sets bit 4 of C/I channel 2 to logic high. The MPC821 then aborts its transmission and retransmits the frame when this bit is set again. This procedure is automatically handled for the first two buffers of a frame.

**16.12.7.1 SI GCI ACTIVATION/DEACTIVATION PROCEDURE.** In the deactivated state, the clock pulse is disabled and the data line is at a logic one. The layer 1 device activates the MPC821 by enabling the clock pulses and by an indication in the channel 0 C/I channel. The MPC821 reports to the CPU core (via a maskable interrupt) that a valid indication is in the SMC receive BD.

When the CPU core activates the line, the data output of L1TXDn is programmed to zero by setting the STZx bit in the SIMODE register. Code 0 (command timing TIM) is transmitted on channel 0 C/I channel to the layer 1 device until the STZx bit is reset. The physical layer device resumes the clock pulses and gives an indication in the channel 0 C/I channel. The CPU core should reset the STZx bit to enable data output.

#### 16.12.7.2 SERIAL INTERFACE GCI PROGRAMMING.

**16.12.7.2.1 Normal Mode GCI Programming.** The user can program and configure the channels used for the GCI bus interface. First, the SIMODE register to the GCI/SCIT mode for that channel must be programmed, using the DSCx, FEx, CEx, and RFSDx bits. This mode defines the sync pulse to GCI sync for framing and data clock as one-half the input clock rate. The user can program more than one channel to interface to the GCI bus. Also, if the receive and transmit section are used for interfacing the same GCI bus, the user internally connects the receive clock and sync signals to the SI RAM transmit section, using the CRTx bits. The user should then define the GCI frame routing and strobe select using the SI RAM.

When the receive and transmit section uses the same clock and sync signals, these sections should be programmed to the same configuration. Also, the L1TXDx pin in the I/O register should be programmed to be an open-drain output. To support the monitor and the C/I channels in GCI, those channels should be routed to one of the SMCs. To support the D channel when there is no possibility of collision, the user should clear the GRx bit corresponding to the SCC that supports the D channel in the SIMODE register.

**16.12.7.2.2 SCIT Programming.** For interfacing the GCI/SCIT bus, the SIMODE register must be programmed to the GCI/SCIT mode. The SI RAM is programmed to support a 96-bit frame length and the frame sync is programmed to the GCI sync pulse. Generally, the SCIT bus supports the D channel access collision mechanism. For this purpose, the user should program the receive and transmit sections to use the same clock and sync signals with the CRTx bits and program the GRx bits to transfer the D channel grant to the SCC that supports this channel. The received (grant) bit should be marked by programming the channel select bits of the SI RAM to 111 for an internal assertion of a strobe on this bit. This bit is sampled by the SI and transferred to the D channel SCC as the grant. The bit is generally bit 4 of the C/I in channel 2 of GCI, but any other bit can be selected using the SI RAM.

For example, assuming that SCC1 is connected to the D channel, SCC2 to the B1 channel, and SMC2 to the B2 channel, SMC1 is used to handle the C/I channels, and the D channel grant is on bit 4 of the C/I on SCIT channel 2, the initialization sequence is as follows:

1. Program the SI RAM. Write all entries that are not used with $0001, set the LST bit, and disable the routing function.

| ENTRY NO. | RAM WORD | | | | | | |
|---|---|---|---|---|---|---|---|
| | SWTR | SSEL | CSEL | CNT | BYT | LST | DESCRIPTION |
| 1 | 0 | 0000 | 010 | 0000 | 1 | 0 | 8 Bits SCC2 |
| 2 | 0 | 0000 | 110 | 0000 | 1 | 0 | 8 Bits SMC2 |
| 3 | 0 | 0000 | 101 | 0000 | 1 | 0 | 8 Bits SMC1 |
| 4 | 0 | 0000 | 001 | 0001 | 0 | 0 | 2 Bits SCC1 |
| 5 | 0 | 0000 | 101 | 0101 | 0 | 0 | 6 Bits SMC1 |
| 6 | 0 | 0000 | 000 | 0110 | 1 | 0 | Skip 7 Bytes |
| 7 | 0 | 0000 | 000 | 0001 | 0 | 0 | Skip 2 Bits |
| 8 | 0 | 0000 | 111 | 0000 | 0 | 1 | D Grant Bit |

**NOTE**

Since GCI requires the same routing for both receive and transmit, an exact duplicate of the above entries should be written to both the receive and transmit sections of the SI RAM beginning at addresses 0 and 128, respectively.

2. SIMODE = $800080E0. Only TDMa is used; SMC1 and SMC2 are connected. SCIT mode is used in this example.

**NOTE**

If SCIT mode is not used, delete the last three entries of the SI RAM and set the LST bit in the new last entry.

3. SICR = $000040C0. SCC2 and SCC1 are connected to the TSA. SCC1 supports the grant mechanism since it is on the D channel.
4. PAODR bit 9 = 1. Configures L1TXDa to an open-drain output.
5. PAPAR bits 9, 8, and 7= 1. Configures L1TXDa, L1RXDa, and L1RCLKa.
6. PADIR bits 9 and 8 = 1. PADIR bit 7 = 0. Configures L1TXDa, L1RXDa, and L1RCLKa.
7. If the 1× GCI data clock is required, set PBPAR bit 20 and PBDIR bit 20, which configures L1CLKOa as an output.
8. PCPAR bit 4 = 1. Configures L1RSYNCa.

9. SIGMR = $04. Enable TDMa (one static TDM).
10. SICMR is not used.
11. SISTR and SIRP do not need to be read, but can be used for debugging information once the channels are enabled.
12. Enable the SCC1 for HDLC operation (to handle the LAPD protocol of the D channel), set SCC2 and SMC2 as preferred, and enable SMC1 for SCIT operation.

### 16.12.8 NMSI Configuration

The SI supports an NMSI mode for each of the SCCs and SMCs. The decision of whether to connect a SCC to the NMSI is made in the SICR and the decision of whether to connect a SMC to the NMSI is made in the SIMODE register. An SCC or SMC can be connected to the NMSI, regardless of the other channels connected to a TDM channel. The user should note, however, that 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 SCCs and SMCs to connect and where to connect them should be made by consulting the MPC821 pinout.

The clocks that are provided to the SCCs and SMCs are derived from twelve sources—four internal baud rate generators and eight external CLK pins. Refer to Figure 16-64 for details. There are two main advantages to the bank-of-clocks approach. First, an SCC or SMC is not forced to choose it’s clock from a predefined pin or baud rate generator that allows flexibility in the pinout mapping strategy. Second, if a group of SCC receivers and transmitters need the same clock rate, they can share the same pin. This configuration leaves additional pins for other functions and minimizes potential skew between multiple clock sources.

The four baud rate generators also make their clocks available to external logic, regardless of whether the baud rate generators are being used by an SCC or SMC. Notice that the BRGOx pins are multiplexed with other functions thus, all BRGOx pins may not always be available. Also notice that BRGO3 has the flexibility to be output on both port A 12 and port B 16. Refer to the pinout description in **Section 2 External Signals** for more details.

There are a few restrictions in the bank-of-clocks mapping. First, only eight of the twelve sources can be connected to any given SCC receiver or transmitter. Second, the SMC transmitter must have the same clock source as the receiver when connected to the NMSI pins. Once the clock source is selected, the clock is given an internal name. For the SCCs, the name is RCLKx and TCLKx and for the SMCs, the name is simply SMCLKx. 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 any pins on the MPC821.

**NOTE**

The internal RCLKx and TCLKx can be used as inputs to the DPLL unit, which is inside the SCC. Thus, the RCLKx and TCLKx signals are not always required to reflect the actual bit rate on the line.

The exact pins available to each SCC and SMC in the NMSI mode are illustrated in Figure 16-64. The SCC1 in NMSI mode has it's own set of modem control pins:

- TXD1
- RXD1
- TCLK1 ←BRG1–BRG4, CLK1–CLK4
- RCLK1 ← BRG1–BRG4, CLK1–CLK4
- $\overline{\text{RTS1}}$
- $\overline{\text{CTS1}}$
- $\overline{\text{CD1}}$

The SCC2 in NMSI mode has it's own set of modem control pins:

- TXD2
- RXD2
- TCLK2 ← BRG1–BRG4, CLK1–CLK4
- RCLK2 ← BRG1–BRG4, CLK1–CLK4
- $\overline{\text{RTS2}}$
- $\overline{\text{CTS2}}$
- $\overline{\text{CD2}}$

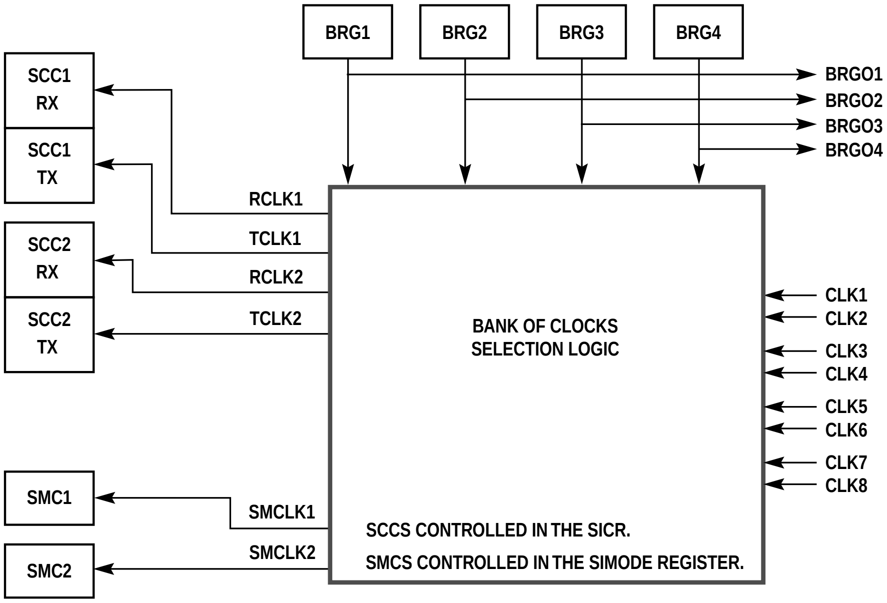
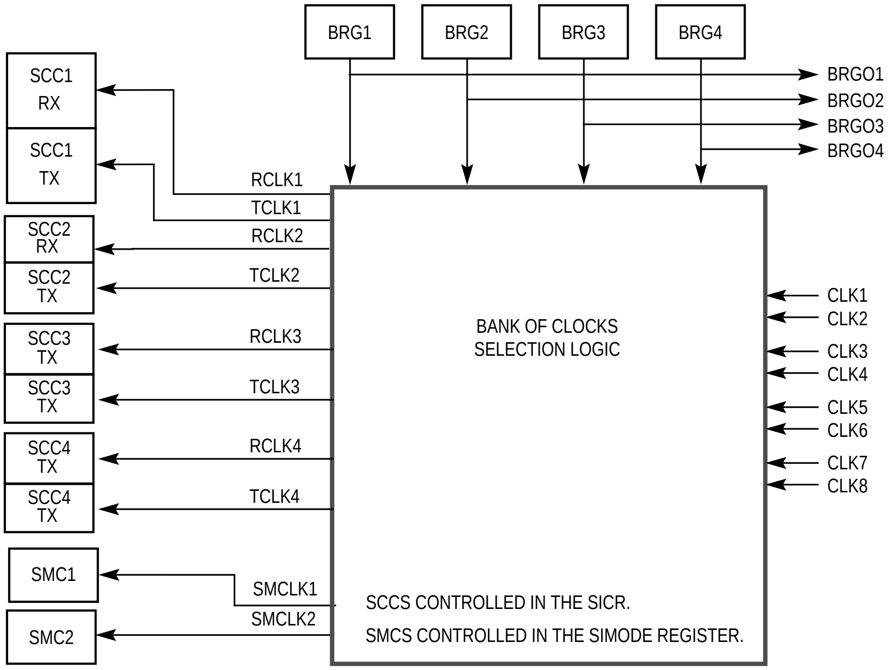


**Figure 16-64. Bank of Clocks**



The SMC1 in NMSI mode has it’s own set of modem control pins:

- SMTXD1
- SMRXD1
- SMCLK1 ← BRG1–BRG4, CLK1–CLK4
- SMSYN1 (used only in the totally transparent protocol)

The SMC2 in NMSI mode has it’s own set of modem control pins:

- SMTXD2
- SMRXD2
- SMCLK2 ← BRG1–BRG4, CLK5–CLK8
- SMSYN2 (used only in the totally transparent protocol)

Any SCC or SMC that requires fewer pins than those listed can use that pin for another function or configure that pin as a parallel I/O pin.

## 16.13 BAUD RATE GENERATORS

The CPM contains four, independent, identical baud rate generators (BRGs) that can be used with the SCCs and SMCs. The clocks produced in the BRG are sent to the bank-of-clocks selection logic, where they can be routed to the SCCs and/or SMCs. The bank-of-clocks logic is described in more detail in **Section 16.12.8 NMSI Configuration.** In addition, the output of the BRG can be routed to a pin to be used externally. The important features of the BRGs are as follows:

- Four independent and identical BRGs
- On-the-fly changes allowed
- Each BRG can be routed to one or more SCCs or SMCs
- A 16× 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 (BRGO1)

The BRG block diagram is illustrated in the figure below.

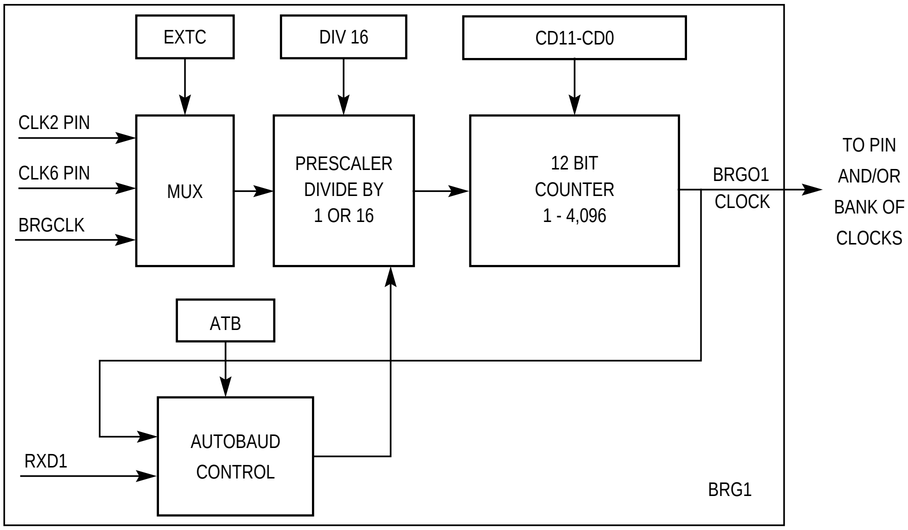


**Figure 16-65. Baud Rate Generator Block Diagram**

The clock input to the prescaler can be selected by the EXTC bits to come from one of three sources—BRGCLK, CLK2, or CLK6. The BRGCLK is generated in the MPC821 clock synthesizer specifically for the four BRGs, SPI, and I$^2$C internal BRG. Alternatively, the user can choose the CLK2 or CLK6 pins to be the clock source. An external pin allows flexible baud rate frequency generation, regardless of the system frequency. Additionally, the CLK2 or CLK6 pins allow a single external frequency to become the input clock for multiple BRGs. The clock signals on the CLK2 and CLK6 pins are not synchronized internally prior to being used by the BRG.

Next, the BRG provides a divide-by-16 option before the clock reaches the prescaler. This option is chosen by the DIV16 bit. The clock is then divided in the prescaler by up to 4,096. This input clock divide ratio can be programmed on-the-fly. However, two on-the-fly BRG changes should not occur within a shorter time than the period of at least two BRG input clocks.

The output of the prescaler is sent internally to the bank of clocks and can also be output externally on the BRGOx pins of either the port A or port B parallel I/O. One BRGOx pin (BRGO4–BRGO1) is an output from the corresponding BRG. If the BRG divides the clock by an even value, the transitions of the BRGO pin always occurs on the falling edge of the input clock to the BRG. If the BRG is programmed to an odd value, the transitions alternates between the falling and rising edges of the input clock. Additionally, the output of the BRG can be sent to the autobaud control block.

### 16.13.1 Autobaud Support

In the autobaud process, a UART deduces the baud rate of its received character stream by examining the pattern received and timing information of that pattern. The MPC821 BRGs have a built-in autobaud control function that automatically measures the length of a start bit and modifies the baud rate accordingly. If the ATB bit in the BRG is set, the autobaud control block starts searching for a low level on the corresponding RXDx input line (RXD2–RXD1) which it assumes is the beginning of a start bit and begins counting the start bit length. During this time, the BRG output clock toggles for 16 BRG clock cycles at the BRG input clock rate and then stops with the BRGO output clock in the low state.

After the RXDx line changes back to the high level, the autobaud control block rewrites the CD and DIV16 bits in the BRG configuration register to the divide ratio it found. Due to measurement error that occurs at high baud rates, this divide rate written by the autobaud controller may not be the precise, final baud rate the user prefers (56,600 could be the resulting baud rate, rather than 57,600). Thus, an interrupt is provided to the user in the UART SCC event register to signify that the BRG configuration register was rewritten by the autobaud controller. On recognition of this interrupt, the user should rewrite the BRG configuration register with the preferred value. It is recommended that this be done as quickly as possible (even prior to the first character being fully received) to ensure that all characters are recognized correctly by the UART.

Once a full character is received, the user can check in the software to see if the received character matches a predefined value (such as “a” or “A”). Software should then check for other characters (such at “t” or “T”) and program the SCC to the preferred parity mode. Changes in the parity mode can be accomplished in the UART protocol-specific mode register (PSMR).

**NOTE**

The SCC associated with this BRG must be programmed to UART mode and must have the TDCR and RDCR bits in the general SCC mode register set to the 16× option for the autobaud function to operate correctly. Input frequencies such as 1.8432 MHz, 3.68 MHz, 7.36 MHz, and 14.72 MHz should be used. For autobaud to operate successfully, the SCC performing the autobaud function must be connected to the BRG for that SCC. For instance, the SCC2 must be clocked by BRG2 to successfully perform the autobaud function. Also, for the SCC to correctly 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 set the GSMR with the ATB=0 and enable the BRG Rx clock to the highest frequency. Immediately prior to the start of the autobaud process (after device initialization), set the ATB bit equal to 1.

### 16.13.2 Baud Rate Generator Configuration Register

Each BRG cofiguration register (BRGC) is a 24-bit, memory-mapped, read/write register that is cleared at reset. A reset disables the BRG and puts the BRGO output clock to the high level. The BRGC can be written at any time with no need to disable the SCCs or the external devices that are connected to the BRGO output clock. The BRG changes occurs at the end of the next BRG clock cycle (no spikes occurs on the BRGO output clock). The BRGC allows on-the-fly changes, but two on-the-fly changes should not occur within a shorter time than the period of at least two BRG input clocks.

**BRGC REGISTER**

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | | | | | | | | RST | EN |
| 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 | 9F0 (BRGC1), 9F4 (BRGC2), 9F8 (BRGC3), 9FC (BRGC4) | | | | | | | | | | | | | | | |
| **BITS** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| FIELD | EXTC | | ATB | CD | | | | | | | | | | | | DIV16 |
| RESET | 0 | 0 | 0 | 0 | 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 | 9F2 (BRGC1), 9F6 (BRGC2), 9FA (BRGC3), 9FE (BRGC4) | | | | | | | | | | | | | | | |

Bits 8–13—Reserved

RST—Reset BRG

This bit performs a software reset of the BRG identical to that of an external reset. A reset disables the BRG and sets the BRGO output clock. This can only be seen externally if the BRGO function is enabled to reach the corresponding port B parallel I/O pin.

0 = Enable the BRG.
1 = Reset the BRG (software reset).

EN—Enable BRG Count

This bit is used to dynamically stop the BRG from counting, which may be useful for low-power modes.

0 = Stop all clocks to the BRG.
1 = Enable clocks to the BRG.

EXTC—External Clock Source

The EXTC bits select the BRG input clock from the internal BRGCLK or one of three external pins.

00 = The BRG input clock comes from the BRGCLK (internal clock generated by the clock synthesizer in the SIU).
01 = The BRG input clock comes from the CLK2 pin.
10 = The BRG input clock comes from the CLK6 pin.
11 = Reserved.

ATB—Autobaud

When set, this bit selects autobaud operation of the BRG on the corresponding RXDx pin.

0 = Normal operation of the BRG.
1 = When RXDx goes low, the BRG determines the length of the start bit and synchronizes the BRG to the actual baud rate.

**NOTE**

This bit must remain clear (0) until the SCC receives the three Rx clocks. Then the user must set this bit to one to obtain the correct baud rate. Once the baud rate is obtained and locked, it is indicated by setting the AB bit in the UART event register.

CD—Clock Divider

The clock divider bits (CD0–CD11) and the prescaler determine the BRG output clock rate. CD0–CD11 are used to preset a 12-bit counter that is decremented at the prescaler output rate, but the counter is inaccessible to the user. When the counter reaches zero, it is reloaded from the clock divider bits. Thus, a value of $FFF in CD0–CD11 produces the minimum clock rate (divide by 4,096), and a value of $0000 produces the maximum clock rate (divide by 1). Even when dividing by an odd number, the counter ensures a 50% duty-cycle by asserting the terminal count once on clock low and next on clock high. The terminal count signals counter expiration and toggles the clock.

DIV16—BRG Clock Prescaler Divide by 16

The BRG clock prescaler bit selects a divide-by-1 or divide-by-16 prescaler for the clock divider input.

### 16.13.3 UART Baud Rate Examples

For synchronous communication using the internal baud rate generator, the BRGO output clock must never be faster than the system frequency divided by 2. Thus, with a 25 MHz system frequency, the maximum BRGO output clock rate is 12.5 MHz.

The user should program the UART to 16× over sampling (RDCR and TDCR bits in the general SCC mode register) when using the SCC as a UART. On the MPC821, 8× and 32× options are also available. Assuming 16× over sampling is chosen in the UART, a data rate of 25 MHz ÷ 16 = 1.5625 Mbits/second is the maximum possible UART speed.

Putting this together, the following formula for calculating the bit rate based on a particular BRG configuration for a UART:

> **async baud rate = (BRGCLK or CLK2 or CLK6) ÷ (clock divider + 1) ÷ (1 or 16 depending on the DIV16 bit) ÷ (8 or 16 or 32 according to the RDCR and TDCR bits in the general SCC mode register).**

The following table lists typical bit rates of asynchronous communication. Notice that for this mode, the internal clock rate is assumed to be 16× the baud rate.

### Table 16-22. Typical Baud Rates of Asynchronous Communication

| BAUD RATES | MPC821 SYSTEM FREQUENCY (MHZ) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 20 | | | 25 | | | 24.5760 | | |
| | DIV16 | DIV | ACTUAL FREQUENCY | DIV16 | DIV | ACTUAL FREQUENCY | DIV16 | DIV | ACTUAL FREQUENCY |
| 50 | 1 | 1561 | 50.02 | 1 | 1952 | 50 | 1 | 1919 | 50 |
| 75 | 1 | 1040 | 75.05 | 1 | 1301 | 75 | 1 | 1279 | 75 |
| 150 | 1 | 520 | 149.954 | 1 | 650 | 150 | 1 | 639 | 150 |
| 300 | 1 | 259 | 300.48 | 1 | 324 | 300.5 | 1 | 319 | 300 |
| 600 | 0 | 2082 | 600.09 | 0 | 2603 | 600 | 0 | 2559 | 600 |
| 1200 | 0 | 1040 | 1200.7 | 0 | 1301 | 1200 | 0 | 1279 | 1200 |
| 2400 | 0 | 520 | 2399.2 | 0 | 650 | 2400.1 | 0 | 639 | 2400 |
| 4800 | 0 | 259 | 4807.7 | 0 | 324 | 4807.69 | 0 | 319 | 4800 |
| 9600 | 0 | 129 | 9615.4 | 0 | 162 | 9585.9 | 0 | 159 | 9600 |
| 19200 | 0 | 64 | 19231 | 0 | 80 | 19290 | 0 | 79 | 19200 |
| 38400 | 0 | 32 | 37879 | 0 | 40 | 38109 | 0 | 39 | 38400 |
| 57600 | 0 | 21 | 56818 | 0 | 26 | 57870 | 0 | 26 | 56889 |
| 115200 | 0 | 10 | 113636 | 0 | 13 | 111607 | 0 | 12 | 118154 |

NOTE: All values are decimal.

For synchronous communication, the internal clock is identical to the baud rate output. To get the preferred rate, the user can select the appropriate system clock according to the following equation:

**sync baud rate = (BRGCLK or CLK2 or CLK6) ÷ (clock divider + 1) ÷**
**(1 or 16 according to the DIV16 bit)**

To get the rate of 64 kbps, the system clock can be 24.96 MHz, DIV16 = 0, and the clock divider = 389.

## 16.14 SERIAL COMMUNICATION CONTROLLERS

The following is a list of the SCCs' important features:

- Implements HDLC/SDLC, HDLC bus, asynchronous HDLC, BISYNC, synchronous start/stop, asynchronous start/stop (UART), AppleTalk/LocalTalk, and totally transparent protocols
- Supports full 10-Mbps Ethernet/IEEE 802.3
- Additional protocols supported through Motorola-supplied RAM microcodes: profibus, signaling system#7 (SS7)
- 2-Mbps HDLC, HDLC bus, and/or transparent data rates supported on all two SCCs simultaneously (full duplex)
- 10-Mbps Ethernet (half duplex) on SCC1 and 2 Mbps on the other SCC supported simultaneously (full duplex)
- A single HDLC or transparent channel can be supported at 8 Mbps (full duplex)
- SCC clocking rates up to 12.5 MHz at 25 MHz
- DPLL circuitry for clock recovery with NRZ, NRZI, FM0, FM1, Manchester, and Differential Manchester (also known as Differential Bi-phase-L)
- SCC clocks can be derived from a baud rate generator, an external pin, or DPLL; data clock can be as high as 3.125 MHz with a 25 MHz clock
- Supports automatic control of the RTS, $\overline{\text{CTS}}$, and CD modem signals
- Multibuffer data structure for receive and transmit (up to 512 BDs can be partitioned in any way)
- Deep FIFOs (SCC1 has 32-byte Rx and Tx FIFOs; SCC2 has 16-byte Rx and Tx FIFOs)
- Transmit-on-demand feature decreases time-to-frame transmission
- Low FIFO latency option for transmit and receive in character-oriented and totally transparent protocols
- Frame preamble options
- Full-duplex operation
- Fully transparent option for receiver/transmitter while another protocol executes on the transmitter/receiver
- Echo and local loopback modes for testing

**NOTE**

The performance figures listed in the key features assume a 25-MHz system clock.

### 16.14.1 Overview

The MPC821 has two SCCs that can be configured independently to implement different protocols. Together, they can be used to implement bridging functions, routers, gateways, and interface with a wide variety of standard WANs and LANs, and proprietary networks. The SCCs have many physical interface options such as interfacing to TDM busses, ISDN busses, and standard modem interfaces. Refer to **Section 16.12 Serial Interface with Time-Slot Assigner** for more information.

On the MPC821, the SCC does not include the physical interface, but it is the logic which formats and manipulates the data obtained from the physical interface. That is why the SI section is described separately. The choice of protocol is independent of the choice of physical interface.

The SCC is described in terms of the protocol that it is chosen to run. When an SCC is programmed to a certain protocol, it implements a certain level of functionality associated with that protocol. For most protocols, this corresponds to portions of the link layer (layer 2 of the seven-layer ISO model). Many functions of the SCC are common to all of the protocols. These functions are described in the SCC description. Following that, the specific implementation details that make one protocol different from the others are discussed, beginning with the UART protocol. Thus, the reader should read from this point to the UART protocol and then skip to the particular protocol preferred. Since the SCCs use similar data structures across all protocols, the reader's learning time decreases dramatically after understanding the first protocol.

Each SCC supports a number of protocols—Ethernet, HDLC/SDLC, HDLC bus, BISYNC, IrDA, SCC2, asynchronous or synchronous start/stop (UART), totally transparent operation, and AppleTalk/LocalTalk. Although the selected protocol usually applies to both the SCC transmitter and receiver, the SCCs have an option of running one-half of the SCC with transparent operation while the other half runs the standard protocol.

Each of the internal clocks (RCLK, TCLK) for each SCC can be programmed with either an external or internal source. The internal clocks originate from one of four baud rate generators or one of four external clock pins. These clocks can be as fast as a 1:2 ratio of the system clock (12.5 MHz). However, the SCC’s ability to support a sustained bitstream depends on the protocol as well as other factors. Associated with each SCC is a digital phase-locked loop (DPLL) for external clock recovery. The clock recovery options include NRZ, NRZI, FM0, FM1, Manchester, and Differential Manchester. The DPLL can be configured to NRZ operation to pass the clocks and data to or from the SCCs without modifying them.

Each SCC can be connected to it’s own set of pins on the MPC821. This configuration is called the NMSI and is described in **Section 16.12 Serial Interface with Time-Slot Assigner**. In this configuration, each SCC can support the standard modem interface signals (RTS, $\overline{\text{CTS}}$, and CD) through the port C pins and the CPM interrupt controller (CPIC). Additional handshake signals can be supported with additional parallel I/O lines. The SCC block diagram is illustrated in Figure 16-66.

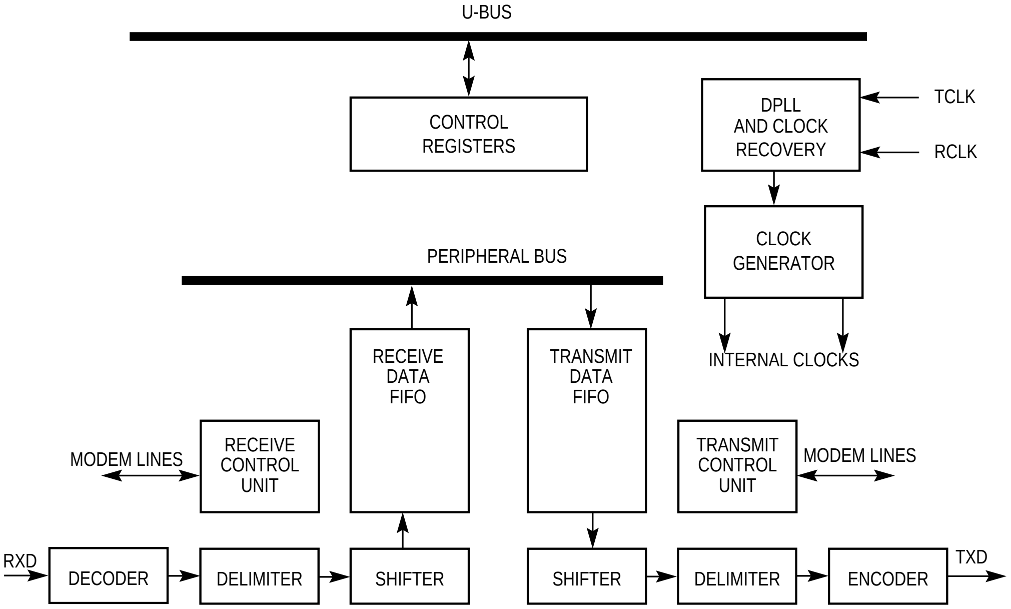


**Figure 16-66. SCC Block Diagram**

## 16.14.2 General SCC Mode Register

Each SCC contains a general SCC mode register (GSMR) that defines all the options common to each SCC, regardless of the protocol. Some of the GSMR operations are described in later sections. The GSMR is a read/write register that is cleared at reset and since it is 64 bits in length, it is accessed as GSMR_L and GSMR_H. GSMR_L contains the first (low-order) 32 bits of the GSMR and GSMR_H contains the last 32 bits.

### 16.14.2.1 GENERAL SCC MODE REGISTER HIGH.

GSMR_H REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | | | | | | | IRP | RES | GDE |
| 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 | A04 (GSMR_H1), A24 (GSMR_H2) | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | TCRC | | REVD | TRX | TTX | CDP | CTSP | CDS | CTSS | TFL | RFW | TXSY | SYNL | | RTSM | RSYN |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | A06 (GSMR_H1), A26 (GSMR_H2) | | | | | | | | | | | | | | | |

Bits 0–12—Reserved

IRP—InfraRed Rx Polarity

This bit determines the polarity of the received signal when the SCC employes IrDA encoding/decoding.

0 = Active high polarity–an active high pulse is decoded as ‘0’.
1 = Active low polarity–an active low pulse is decoded as ‘0’.

Bit 14—Reserved

GDE—Glitch Detect Enable

This bit determines whether the SCC will search for glitches on the external receive and transmit serial clock lines provided. If this feature is enabled, the presence of a glitch is reported in the SCC event register. Whether or not GDE is set, the SCC always attempts to clean up the clocks that it uses internally, via a Schmitt trigger on the input lines.

0 = No glitch detection is performed. This option should be chosen if the external serial clock exceeds the limits of the glitch detection logic (6.25 MHz assuming a 25 MHz system clock). This option should also be chosen if the SCC clock is provided from one of the internal baud rate generators. Lastly, this option should be chosen if external clocks are used and it is more important to minimize power consumption than to watch for glitches.
1 = Glitch detection is performed with a maskable interrupt generated in the SCC event register.

TCRC—Transparent CRC (Valid for a Totally Transparent Channel Only)

These bits select the type of frame checking that is provided on the 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 actual decision to send the frame check is made in the Tx BD. Thus, it is not required to send a frame check in transparent mode. If a frame check is not used, the user can simply ignore the frame check errors that are generated on the receiver.

00 = 16-bit CCITT CRC (HDLC). (X16 + X12 + X5 + 1).
01 = CRC16 (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.

REVD—Reverse Data (valid for a totally transparent channel only)

0 = Normal operation.
1 = When set, this bit causes the totally transparent channels on this SCC (either the receiver, transmitter, or both, as defined by TTX and TRX) to reverse the bit order, transmitting the MSB of each octet first. Refer to **Section 16.14.21.11 BISYNC Mode Register** for the method of reversing the bit order in the BISYNC protocol.

TRX—Transparent Receiver

The MPC821 SCCs offer totally transparent operation. However, to increase flexibility, totally transparent operation is not configured with the MODE bits, but with the TTX and TRX bits. This gives the user the opportunity to implement unique applications, such as an SCC transmitter configured to UART and the receiver configured to totally transparent operation. To do this, set MODE = UART, TTX = 0, and TRX = 1.

0 = Normal operation.
1 = The receiver operates in totally transparent mode, regardless of the protocol selected for the transmitter in the MODE bits.

**NOTE**

Full-duplex totally transparent operation for an SCC is obtained by setting both TTX and TRX. An SCC cannot operate with Ethernet on it's transmitter while transparent operation is on it's receiver or erratic behavior occurs. In other words, if the GSMR MODE = Ethernet, TTX must equal TRX or erratic operation results.

TTX—Transparent Transmitter

The MPC821 SCCs offer totally transparent operation. However, to increase flexibility, totally transparent operation is not configured with the MODE bits, but with the TTX and TRX bits. This gives the user the opportunity to implement unique applications, such as an SCC receiver configured to HDLC and a transmitter configured to totally transparent operation. To do this, set MODE = HDLC, TTX = 1, and TRX = 0.

0 = Normal operation.
1 = The transmitter operates in totally transparent mode, regardless of the protocol selected for the receiver in the MODE bits.

**NOTE**

Full-duplex totally transparent operation for an SCC is obtained by setting both TTX and TRX. An SCC cannot operate with Ethernet on it's receiver while transparent operation is on it's transmitter or erratic behavior occurs. In other words, if the GSMR MODE = Ethernet, TTX must equal TRX or erratic operation results.

CDP—CD Pulse

0 = Normal operation (envelope mode). The CD pin should envelope the frame and to negate CD while receiving causes a CD lost error.
1 = Pulse mode. Once the CD pin is asserted, synchronization has been achieved, and further transitions of CD have no effect on reception.

**NOTE**

This bit must be set if this SCC is used in the TSA.

CTSP—$\overline{\text{CTS}}$ Pulse

0 = Normal operation (envelope mode). The $\overline{\text{CTS}}$ pin should envelope the frame and to negate $\overline{\text{CTS}}$ while transmitting causes a CTS lost error.
1 = Pulse mode. Once the $\overline{\text{CTS}}$ pin is asserted, synchronization has been achieved, and further transitions of $\overline{\text{CTS}}$ have no effect on transmission.

CDS—CD Sampling

0 = The CD input is assumed to be asynchronous with the data. It is internally synchronized by the SCC and then data is received.
1 = The CD input is assumed to be synchronous with the data, giving faster operation. In this mode, CD must transition while the receive clock is in the low state. As soon as CD is low, data is received. This mode is especially useful when connecting MPC821 s in transparent mode since it allows the RTS pin of one MPC821 to be directly connected to the CD pin of the other MPC821.

CTSS—$\overline{\text{CTS}}$ Sampling

0 = The $\overline{\text{CTS}}$ input is assumed to be asynchronous with the data. It is internally synchronized by the SCC and data is then transmitted after several serial clock delays.
1 = The $\overline{\text{CTS}}$ input is assumed to be synchronous with the data, giving faster operation. In this mode, $\overline{\text{CTS}}$ must transition while the transmit clock is in the low state. As soon as $\overline{\text{CTS}}$ is low, data immediately begins transmission. This mode is especially useful when connecting MPC821 in transparent mode since it allows the RTS pin of one MPC821 to be directly connected to the $\overline{\text{CTS}}$ pin of the other MPC821.

TFL—Transmit FIFO Length

0 = Normal operation. The transmit FIFO is 32 bytes for SCC1 and 16 bytes for the other SCCs.
1 = The transmit FIFO is 1 byte and can be used with character-oriented protocols, such as UART, to ensure a minimum FIFO latency at the expense of performance.

RFW—Rx FIFO Width

0 = Rx FIFO is 32 bits wide for maximum performance. Data is not normally written to receive buffers until at least 32 bits are received. This configuration is required for HDLC-type protocols and Ethernet, but it is recommended for high-performance transparent modes. In this mode, the receive FIFO is 32 bytes for SCC1 and 16 bytes for the other SCCs.
1 = Low-latency operation. The Rx FIFO is 8 bits wide and the receive FIFO is a quarter it’s normal size (8 bytes for SCC1 and 4 bytes for the other SCCs). This allows data to be written to the data buffer when a character is received, instead of waiting to receive 32 bits. This configuration must be chosen for character-oriented protocols, such as UART and BISYNC. It can also be used for low-performance,

low-latency, totally transparent operation, if preferred. It must not be used with HDLC, HDLC Bus, AppleTalk, or Ethernet because erratic behavior can occur.

TXSY—Transmitter Synchronized to the Receiver

The TXSY bit is specifically intended for X.21 applications where the transmitted data must begin an exact multiple of 8-bit periods after the received data arrives.

0 = No synchronization between receiver and transmitter (default).
1 = The transmit bitstream is synchronized to the receiver. Additionally, if RSYN = 1, then transmission in the totally transparent mode does not occur until the receiver has synchronized with the bitstream and the $\overline{\text{CTS}}$ signal is asserted to the SCC. Assuming $\overline{\text{CTS}}$ is already asserted, transmission begins eight clocks after the receiver starts getting data.

SYNL—Sync Length (BISYNC and transparent mode only)

These bits determine the operation of an SCC receiver that is configured for BISYNC or totally transparent operation only. See the data synchronization register definition in the BISYNC and totally transparent descriptions for more information.

00 = The sync pattern in the DSR is not used. An external sync signal is used instead (CD pin asserted).
01 = 4-bit sync. The receiver synchronizes on a 4-bit sync pattern stored in the DSR. This character and additional syncs can be programmed to be stripped using the SYNC character in the parameter RAM. The transmitter transmits the entire contents of the DSR prior to each frame.
10 = 8-bit sync. This option should be chosen along with the BISYNC protocol to implement mono-sync. The receiver synchronizes on an 8-bit sync pattern stored in the DSR. The transmitter transmits the entire contents of the DSR prior to each frame.
11 = 16-bit sync. Also called BISYNC. The receiver synchronizes on a 16-bit sync pattern stored in the DSR. The transmitter transmits the DSR prior to each frame.

RTSM—RTS Mode

This bit can be changed on-the-fly.

0 = Send idles between frames as defined by the protocol and the tend bit. RTS is negated between frames (default).
1 = Send flags/syncs between frames according to the protocol. RTS is always asserted whenever the SCC is enabled.

RSYN—Receive Synchronization Timing (valid for a totally transparent channel only)

0 = Normal operation.
1 = If CDS = 1, then the CD pin should be asserted on the second bit of the receive frame, rather than the first.

### 16.14.2.2 GENERAL SCC MODE REGISTER LOW.

GSMR_L REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | SIR | EDGE | | TCI | TSNC | | RINV | TINV | TPL | | | TPP | | TEND | TDCR | |
| 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 | A00 (GSMR_L1), A20 (GSMR_L2) | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | RDCR | | RENC | | | TENC | | | DIAG | | ENR | ENT | MODE | | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | A02 (GSMR_L1), A22 (GSMR_L2) | | | | | | | | | | | | | | | |

SIR—Serial Infra-Red Encoding

This bit should be set to one to activate the serial infra-red coder/encoder. Available on SCC2 only.

EDGE—Clock Edge

The EDGE bits determine the clock edge used by the DPLL to adjust the receive sample point because of a jitter in the signal that is received. The selection of the EDGE bits is ignored in the UART protocol or the x1 mode of the RDCR bits.

00 = Both the positive and negative edges are used for changing the sample point (default).
01 = Positive edge. Only the positive edge of the received signal is used for changing the sample point.
10 = Negative edge. Only the negative edge of the received signal is used for changing the sample point.
11 = No adjustment is made to the sample points.

TCI—Transmit Clock Invert

0 = Normal operation.
1 = The internal transmit clock (TCLK) is inverted by the SCC before it is used. This option allows the SCC to clock data out one-half clock earlier on the rising edge of TCLK rather than the falling edge. In this mode, 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 transmit clock can be latched by the external receiver one clock cycle later on the next rising edge of the same transmit clock. This option is recommended for Ethernet, HDLC, or transparent operation when the clock rates are high (above 8 MHz) to improve data setup time for the external receiver.

TSNC—Transmit Sense

This bit indicates the amount of time the internal sense signal stays active after the last transition on the RXD pin, indicating that the line is free. For instance, these bits can be used in the AppleTalk protocol to avoid the spurious $\overline{\text{CS}}$-changed interrupt that would otherwise occur during the frame sync sequence preceding the opening flags.

If RDCR is configured to 1× mode, the delay is the greater of the two numbers listed. If RDCR is configured to 8×, 16×, or 32× mode, the delay is the lesser of the two numbers listed.

00 = Infinite—carrier sense is always active (default).
01 = 14- or 6.5-bit times as determined by the RDCR bits.
10 = 4- or 1.5-bit times as determined by the RDCR bits (normally for AppleTalk).
11 = 3- or 1-bit times as determined by the RDCR bits.

RINV—DPLL Receive Input Invert Data

0 = No invert.
1 = Invert the data before it is sent to the on-chip DPLL for reception. This setting is used to produce FM1 from FM0, NRZI space from NRZI mark. It can also be used in regular NRZ mode to invert the datastream.

**NOTE**

This bit must be zero in HDLC bus mode.

TINV—DPLL Transmit Input Invert Data

0 = No invert.
1 = Invert the data before it is sent to the on-chip DPLL for transmission. This setting is used to produce FM1 from FM0, NRZI space from NRZI mark. It can also be used in regular NRZ mode to invert the datastream.

**NOTE**

This bit must be zero in HDLC bus mode.

In T1 applications, setting TINV and Tend creates a continuously inverted HDLC datastream.

TPL—Tx Preamble Length

The TPL bits determine 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.

TPP—Tx Preamble Pattern

The TPP bits determine what, if any, bit pattern should precede the start of each transmit frame. The preamble pattern is sent prior to the first flag/sync of the frame. TPP is ignored if the SCC is programmed to UART mode. The length of the preamble is programmed in TPL. The preamble pattern is typically transmitted to a receiving station that uses a DPLL for clock recovery. The receiving DPLL uses the regular pattern of the preamble to help it lock onto the received signal in a short, predictable time period.

00 = All zeros.
01 = Repeating 10s. Select this setting for Ethernet operation.
10 = Repeating 01s.
11 = All ones. Select this setting for LocalTalk operation.

Tend—Transmitter Frame Ending

This bit is specifically intended for the NMSI transmitter encoding of the DPLL. Tend determines whether the TXD line should idle in a high state or in an encoded ones state (which may be either high or low). It may, however, be used with other encodings besides NMSI.

0 = Default operation. The TXD line is only encoded when data is transmitted (including the preamble and opening and closing flags/syncs). When no data is available to transmit, the line is driven high.
1 = The TXD line is always encoded (even when idles are transmitted).

TDCR—Transmit Divide Clock Rate

The TDCR bits determine the divider rate of the transmitter. If the DPLL is not used, the 1× value should be chosen, except in asynchronous UART mode where 8×, 16×, or 32× must be chosen. The user should program TDCR to equal RDCR in most applications. If the DPLL is used in the application, the selection of TDCR depends on the encoding. NRZI usually requires 1×; whereas, FM0/FM1, Manchester, and Differential Manchester allow 8×, 16×, or 32×. The 8× option allows highest speed; whereas, the 32× option provides the greatest resolution.

TDCR is usually equal to RDCR to allow the same clock frequency source to control both the transmitter and receiver.

00 = 1× clock mode. Only NRZ or NRZI encodings are allowed.
01 = 8× clock mode.
10 = 16× clock mode. Normally chosen for UART and AppleTalk.
11 = 32× clock mode.

RDCR—Receive DPLL Clock Rate

The RDCR bits determine the divider rate of the receive DPLL. If the DPLL is not used, the 1× value should be chosen, except in asynchronous UART mode where 8×, 16×, or 32× must be chosen. The user should program RDCR to equal TDCR in most applications.

If the DPLL is used in the application, the selection of RDCR depends on the encoding. NRZI usually requires 1×; whereas, FM0, FM1, Manchester, and Differential Manchester allow 8×, 16×, or 32×. The 8× option allows highest speed; whereas, the 32× option provides the greatest resolution.

00 = 1× clock mode. Only NRZ or NRZI decodings are allowed.
01 = 8× clock mode.
10 = 16× clock mode. Normally chosen for UART and AppleTalk.
11 = 32× clock mode.

RENC—Receiver Decoding Method

Select NRZ if the DPLL is not used. The user should program RENC to equal TENC in most applications. However, do not use this internal DPLL for Ethernet mode.

000 = NRZ (default setting if DPLL is not used).
001 = NRZI Mark (set RINV also for NRZI space).
010 = FM0 (set RINV also for FM1).
011 = Reserved.
100 = Manchester.
101 = Reserved.
110 = Differential Manchester (Differential Bi-phase-L).
111 = Reserved.

TENC—Transmitter Encoding Method

Select NRZ if the DPLL is not used. The user should program TENC to equal RENC in most applications. However, do not use this internal DPLL for Ethernet mode.

000 = NRZ (default setting if DPLL is not used).
001 = NRZI Mark (set TINV also for NRZI space).
010 = FM0 (set TINV also for FM1).
011 = Reserved.
100 = Manchester.
101 = Reserved.
110 = Differential Manchester (Differential Bi-phase-L).
111 = Reserved.

DIAG—Diagnostic Mode

In typical operation mode, the SCC operates normally. The data received enters the RXD pin and the transmit data is shifted out through the TXD pin. The SCC uses the modem signals (CD and $\overline{\text{CTS}}$) to automatically enable and disable transmission and reception. These timings are shown in **Section 16.14.11 SCC Timing Control**.

00 = Normal operation ($\overline{\text{CTS}}$ and CD signals under automatic control)

In local loopback mode, the transmitter output is internally connected to the receiver input, while the receiver and the transmitter operate normally. The value on the RXD pin is ignored. Data can be programmed to appear on the TXD pin or the TXD pin can remain high by programming the port A register. The RTS line can also be programmed to be disabled in the appropriate parallel I/O register. In TDM modes, the L1TXDx and L1RQx lines can be programmed to be either asserted normally or to remain inactive by programming the serial interface mode register (SIMODE).

When using local loopback mode, the clock source for the transmitter and receiver must be the same. Thus, the same baud rate generator can be used for both transmitter and receiver or the same external CLKx pin can be used for both transmitter or receiver. Separate CLKx pins can be used with the transmitter and receiver as long as the CLKx pins are connected to the same external clock signal source.

01 = Local loopback mode

**NOTE**

If external loopback is preferred, the DIAG bits should be selected for normal operation and an external connection should be made between the TXD and RXD pins. Clocks can be generated internally by a baud rate generator or generated externally. The user can physically connect the appropriate control signals (RTS connected to CD, and $\overline{\text{CTS}}$ grounded) or the port C register can be used to cause the CD and $\overline{\text{CTS}}$ pins to be permanently asserted to the SCC.

In automatic echo mode, the channel automatically retransmits the received data on a bit-by-bit basis using whatever receive clock is provided. The receiver operates normally and receives data if CD is asserted. The transmitter simply transmits received data.
In this mode, the $\overline{\text{CTS}}$ line is ignored. The echo function can also be accomplished in the software by receiving buffers from an SCC, linking them to Tx BDs, and then transmitting them back out of that SCC.

10 = Automatic echo mode

In loopback/echo mode, loopback and echo operation occur simultaneously. The CD and $\overline{\text{CTS}}$ pins are ignored. Refer to the loopback bit description for clocking requirements.

11 = Loopback and echo mode

ENR—Enable Receive

This bit enables the receiver hardware state machine for this SCC. When ENR is cleared, the receiver is disabled and any data in the receive FIFO is lost. If ENR is cleared during reception, the receiver aborts the current character. ENR may be set or cleared regardless of whether the serial clocks are present. Refer to **Section 16.14.14 Disabling the SCCs On-the-Fly** for a description of the proper methods to disable and reenable an SCC.

**NOTE**

> The SCC provides other tools to control reception besides the ENR bit. They are the ENTER HUNT MODE command, CLOSE Rx BD command, and E-bit in the Rx BD.

ENT—Enable Transmit

This bit enables the transmitter hardware state machine for this SCC. When ENT is cleared, the transmitter is disabled. If ENT is cleared during transmission, the transmitter aborts the current character and the TXD pin returns to the idle state. Data already in the transmit shift register is not transmitted. ENT can be set or cleared, regardless of whether serial clocks are present. Refer to **Section 16.14.14 Disabling the SCCs On-the-Fly** for a description of the proper methods to disable and reenable an SCC.

**NOTE**

> The SCC provides other tools to control transmission besides the ENT bit. They are the STOP TRANSMIT command, GRACEFUL STOP TRANSMIT command, RESTART TRANSMIT command, freeze option in UART mode, $\overline{\text{CTS}}$ flow control option in UART mode, and ready (R) bit in the Tx BD.

MODE—Channel Protocol Mode

0000 = HDLC.
0001 = Reserved.
0010 = AppleTalk/LocalTalk.
0011 = SS7. Reserved for RAM microcode.
0100 = UART.
0101 = Profibus. Reserved for RAM microcode.
0110 = Async HDLC.
0111 = V.14. Reserved for RAM microcode.
1000 = BISYNC.
1001 = DDCMP. Reserved for RAM microcode.
1010 = Reserved.
1011 = Reserved.
1100 = Ethernet for SCC1.
11xx = Reserved.

### 16.14.3 SCC Protocol-Specific Mode Register

The functionality of the SCC varies according to the protocol selected by the MODE bits in the GSMR. Each of the four SCCs has an additional 16-bit, memory-mapped, read/write protocol-specific mode register (PSMR) that configures them specifically for a chosen mode. A detailed description each of the PSMR bits is contained within each specific protocol. The PSMRs are cleared at reset.

### 16.14.4 SCC Data Synchronization Register

Each of the four SCC has a 16-bit, memory-mapped, read/write data synchronization register (DSR) that specifies the pattern used in the frame synchronization procedure of the synchronous protocols. In the UART protocol, it is used to configure fractional stop bit transmission. In the BISYNC and totally transparent protocol, it should be programmed with the preferred SYNC pattern. In the Ethernet protocol, it should be programmed with $D555. At reset, it defaults to $7E7E (two HDLC flags), so it does not need to be written for HDLC mode. When the DSR is used to send out SYNCs (such as in BISYNC or transparent mode), the contents of the DSR are always transmitted LSB first.

DSR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | SYN2 | | | | | | | | SYN1 | | | | | | | |
| RESET | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| R/W | R/W | R/W | 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 | A0E (DSR1), A2E (DSR2) | | | | | | | | | | | | | | | |

### 16.14.5 SCC Transmit-on-Demand Register

If no frame is currently being transmitted by an SCC, the RISC controller periodically polls the R-bit of the next Tx BD to see if the user has requested a new frame/buffer to be transmitted. This polling algorithm depends on the SCC configuration, but occurs every 8 to 32 serial transmit clocks. The user, however, has the option to request that the RISC begin processing the new frame/buffer immediately, without waiting until the normal polling time. To obtain immediate processing, the transmit-on-demand (TOD) bit in the transmit-on-demand register (TODR) is set after the R-bit in the Tx BD is set.

This feature, which decreases the transmission latency of the transmit buffer/frame, is particularly useful in LAN-type protocols where maximum interframe GAP times are limited by the protocol specification. Since the transmit-on-demand feature gives a high priority to the specified Tx BD, it can conceivably affect the servicing of the other SCC FIFOs. Therefore, it is recommended that the transmit-on-demand feature only be used when a high-priority Tx BD has been prepared and transmission on this SCC has not occurred for a certain period of time.

The TOD bit does not need to be set if a new Tx BD is added to the circular queue, but other Tx BDs in that queue have not fully completed transmission. In that case, the new Tx BD is processed immediately following the completion of the older Tx BDs. The first bit of the frame will typically be clocked out 5-6 bit times after TOD has been set to 1.

TODR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | TOD | RESERVED | | | | | | | | | | | | | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | | | | | | | | | | | | | | | | |
| ADDR | A0C (TODR1), A2C (TODR2) | | | | | | | | | | | | | | | |

TOD—Transmit on Demand

0 = Normal operation.

1 = The RISC gives a high priority to the current Tx BD and does not wait for the normal polling time to check that the Tx BD R-bit has been set. It begins transmitting the frame. This bit will be cleared automatically after one serial clock.

Bits 1–15—Reserved

These bits should be written with zeros.

### 16.14.6 SCC Buffer Descriptors

Data associated with each SCC channel is stored in buffers. Each buffer is referenced by a buffer descriptor that can be located anywhere in internal memory. The MPC821 internal memory has space for 224 BDs to be shared between the two SCCs, SMCs, SPI and I$^2$C that are used. However, the allocation of BDs to the transmitter or receiver of a serial channel is user-defined. 100 BDs for the SCC1 receiver or 20 BDs for the SCC1 transmitter can be selected.

The BD table forms a circular queue with a programmable length. The user can program the start address of each channel BD table in the internal memory. The user is allowed to allocate the parameter area of an unused channel to the other used channels as BD tables or as actual buffers. See Figure Figure 16-67 for details.

The format of the BDs is the same for each SCC mode of operation and for transmit and receive. The first word in each BD contains a status and control word, that determines the BD table length. Only this first field (containing the status and control bits) differs for each protocol. The second word determines the data length referenced to this BD and the last two words in the BD contain the 32-bit address pointer that points to the actual buffer in memory.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | STATUS AND CONTROL | | | | | | | | | | | | | | | |
| OFFSET + 2 | DATA LENGTH | | | | | | | | | | | | | | | |
| OFFSET + 4 | HIGH-ORDER DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| OFFSET + 6 | LOW-ORDER DATA BUFFER POINTER | | | | | | | | | | | | | | | |

For frame-oriented protocols, a message can reside in as many buffers as necessary (transmit or receive). Each buffer has a maximum length of (64K–1) bytes. The CP does not assume that all buffers of a single frame are currently linked to the BD table. It does assume, however, that the unlinked buffers are provided by the CPU core in time to be transmitted or received. Failure to do so results in an error condition being reported by the CP. An underrun error is reported in the case of transmit and a busy error is reported in the case of receive.

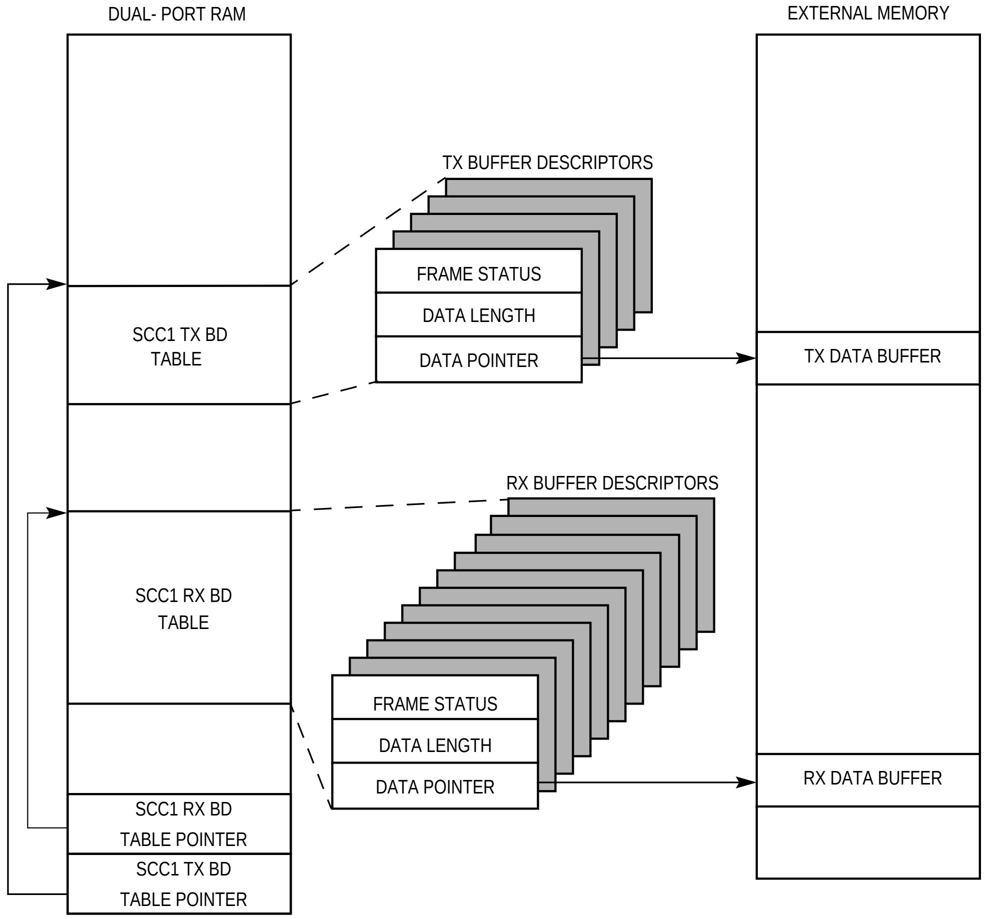


**Figure 16-67. SCC Memory Structure**

All protocols can have their buffer descriptors point to data buffers that are located in internal dual-port RAM. Typically, however, due to the internal RAM being used for buffer descriptors, it is customary for the data buffers to be located in external RAM, especially if the data buffers have a large size. In all cases, the internal U-Bus is used to transfer the data to the data buffer.

The CP processes the Tx BDs in a straightforward fashion. Once the transmit side of an SCC is enabled, it starts with the first BD in that SCC transmit table. Once the CP detects that the Tx BD R-bit is set, it begins processing the buffer. The CP detects that the BD is ready either by polling the R-bit periodically or by the user writing to the TODR. Once the data from the BD has been placed in the transmit FIFO, the CP moves on to the next BD, again waiting for that BD R-bit to be set. Thus, the CP does no look-ahead BD processing, nor does it skip over BDs that are not ready. When the CP sees the wrap (W) bit set in a BD, it goes back to the beginning of the BD table after processing of the BD is complete. After using a BD, the CP normally sets the R-bit to not-ready, thus, the CP does not use a BD twice until the BD has been confirmed by the CPU core. The one exception to this rule is that the MPC821 supports an option for repeated transmission, called the continuous mode, whereby the R-bit is left in the ready position. This is available in some protocols.

The CP uses the Rx BDs in a similar fashion. Once the receive side of an SCC is enabled, it starts with the first BD in that SCC Rx BD table. Once data arrives from the serial line into the SCC, the CP performs certain required protocol processing on the data and moves the resultant data to the data buffer pointed to by the first BD. Use of a BD is complete when there is no more room left in the buffer or when certain events occur, such as detection of an error or an end-of-frame. Whatever the reason, the buffer is then said to be closed and additional data is stored using the next BD. Whenever the CP needs to begin using a BD because new data is arriving, it checks the E-bit of that BD. If the current BD is not empty, it reports a busy error. However, it does not move from the current BD until it is empty. When the CP sees the W-bit set in a BD, it goes back to the beginning of the BD table after processing of the BD is complete and after using a BD, the CP sets the E-bit to not-empty and never uses a BD twice until the BD has been processed by the CPU core. The one exception to this rule is that the MPC821 supports an option for repeated reception, called the continuous mode, whereby the E-bit is left in the empty position. This is available in some protocols.

### 16.14.7 SCC Parameter RAM

Each SCC parameter RAM area begins at the same offset from each SCC base area. The protocol-specific portions of the SCC parameter RAM are discussed in the specific protocol descriptions and the part of it that is the same for all SCC protocols is shown in Table 16-23.

Certain parameter RAM values need to be initialized by the user before the SCC is enabled and other values are initialized/written by the CP. Once initialized, most parameter RAM values do not need to be accessed in the user software since most of the activity is centered around the transmit and Rx BDs, not the parameter RAM. However, if the parameter RAM is accessed by the user, the following regulations should be noted. The parameter RAM can be read at any time. The parameter time values related to the SCC transmitter can only be written whenever the transmitter is disabled after a STOP TRANSMIT and before a RESTART TRANSMIT command, or after the buffer/frame completes transmission as a result of a GRACEFUL STOP TRANSMIT command and before a RESTART TRANSMIT command. The parameter RAM values related to the SCC receiver can only be written when the receiver is disabled. Refer to **Section 16.14.14 Disabling the SCCs On-the-Fly** for more information.

Table 16-23. SCC Parameter RAM Common to All Protocols

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SCC Base + 00 | **RBASE** | Half-word | Rx BD Base Address |
| SCC Base + 02 | **TBASE** | Half-word | Tx BD Base Address |
| SCC Base + 04 | **RFCR** | Byte | Rx Function Code |
| SCC Base + 05 | **TFCR** | Byte | Tx Function Code |
| SCC Base + 06 | **MRBLR** | Half-word | Maximum Receive Buffer Length |
| SCC Base + 08 | RSTATE | Word | Rx Internal State |
| SCC Base + 0C |  | Word | Rx Internal Data Pointer |
| SCC Base + 10 | RBPTR | Half-word | Rx BD Pointer |
| SCC Base + 12 |  | Half-word | Rx Internal Byte Count |
| SCC Base + 14 |  | Word | Rx Temp |
| SCC Base + 18 | TSTATE | Word | Tx Internal State |
| SCC Base + 1C |  | Word | Tx Internal Data Pointer |
| SCC Base + 20 | TBPTR | Half-word | Tx BD Pointer |
| SCC Base + 22 |  | Half-word | Tx Internal Byte Count |
| SCC Base + 24 |  | Word | Tx Temp |
| SCC Base + 28 | RCRC | Word | Temp Receive CRC |
| SCC Base + 2C | TCRC | Word | Temp Transmit CRC |
| SCC Base + 30 |  |  | First Word of Protocol-Specific Area |
| SCC Base + xx |  |  | Last Word of Protocol-Specific Area |

NOTE: Items in bold must be initialized by the user.
SCC base = IMMR +0x1C00 (SCC1) or 0x1D00 (SCC2).

**16.14.7.1 BUFFER DESCRIPTOR TABLE POINTER.** The RBASE and TBASE entries define the starting location in the dual-port RAM for the set of BDs for receive and transmit functions of the SCC. This provides a great deal of flexibility in how BDs for an SCC are partitioned. By selecting RBASE and TBASE entries for all SCCs and by setting the W-bit in the last BD in each BD list, the user can select how many BDs to allocate for the transmit and receive side of every SCC. The user must initialize these entries before enabling the corresponding channel. Furthermore, the user should not configure BD tables of two enabled SCCs to overlap or erratic operation occurs.

**NOTE**

RBASE and TBASE should contain a value that is divisible by eight.

**16.14.7.2 SCC FUNCTION CODE REGISTERS.** There are eight separate function code registers for the four SCC channels—four for receive data buffers (RFCRx) and four for transmit data buffers (TFCRx). The FC entry contains the value that the user wants to appear on the address type pins AT1–AT3 when the associated SDMA channel accesses memory. It also controls the byte-ordering convention to be used in the transfers.

### Receive Function Code Register

RFCRx REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| RESET | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | SCC BASE + 00 | | | | | | | |

Bits 0–2—Reserved

BO—Byte Ordering

The user should set this bit field to select the required byte ordering of the data buffer. If this bit field is modified on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next buffer descriptor.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

### Transmit Function Code Register

TFCRx REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| RESET | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | SCC BASE + 02 | | | | | | | |

Bits 0–2—Reserved

BO—Byte Ordering

The user should set this bit field to select the required byte ordering of the data buffer. If this bit field is modified on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next buffer descriptor.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

**16.14.7.3 MAXIMUM RECEIVE BUFFER LENGTH REGISTER.** Each SCC has one maximum receive buffer length register (MRBLR) to define the receive buffer length for that SCC. MRBLR defines the maximum number of bytes that the MPC821 writes to a receive buffer on that SCC before moving to the next buffer. The MPC821 can write fewer bytes to the buffer than MRBLR if a condition, such as an error or end-of-frame occurs, but it never writes more bytes than the MRBLR value. It follows then, that buffers supplied by the user for the MPC821 to use should always be of MRBLR size (or greater) in length.

The transmit buffers for an SCC are not affected in any way by the value programmed into MRBLR. Transmit buffers may be individually chosen to have varying lengths, as needed. The number of bytes to be transmitted is chosen by programming the data length field in the Tx BD.

NOTE

MRBLR is not intended to be changed dynamically while an SCC is operating. However, if it is modified in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back), then a dynamic change in receive buffer length can be successfully achieved. This takes place when the CP moves control to the next Rx BD in the table. Thus, a change to MRBLR does not have an immediate effect. To guarantee the exact Rx BD on which the change occurs, the user should change MRBLR only while the SCC receiver is disabled. The MRBLR value should be greater than zero for all modes. For Ethernet and HDLC the MRBLR should be evenly divisible by 4. In totally transparent mode, MRBLR should also be divisible by 4, unless the receive FIFO width (RFW) bit in the GSMR is set to 8 bits.

**16.14.7.4 RECEIVER BD POINTER.** The receiver BD pointer (RBPTR) for each SCC channel points to the next BD 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 this pointer to the value programmed in the RBASE entry. Although RBPTR need never be written by the user in most applications, it can be modified by the user when the receiver is disabled or when the user is sure that no receive buffer is currently in use.

**16.14.7.5 TRANSMITTER BD POINTER.** The transmitter BD pointer (TBPTR) for each SCC channel points to the next BD that the transmitter transfers data from when it is in idle state or to the current BD during frame transmission. After a reset or when the end of the BD table is reached, the CP initializes this pointer to the value programmed in the TBASE entry. Although TBPTR need never be written by the user in most applications, it can be modified by the user when the transmitter is disabled or when the user is sure that no transmit buffer is currently in use (after a STOP TRANSMIT or GRACEFUL STOP TRANSMIT command is issued and the frame completes it’s transmission).

**16.14.7.6 OTHER GENERAL PARAMETERS.** These parameters do not need to be accessed by the user in normal operation, and are listed only because they provide helpful information for experienced users and for debugging purposes. Additional parameters are listed in Table 16-23. The Rx and Tx internal data pointers are updated by the SDMA channels to show the next address in the buffer to be accessed. The Tx internal byte count is a down-count value that is initialized with the Tx BD data length and decremented with every byte read by the SDMA channels. The Rx internal byte count is a down-count value that is initialized with the MRBLR value and decremented with every byte written by the SDMA channels.

NOTE

To extract data from a partially full receive buffer, the CLOSE Rx BD command may be used.

The Rx internal state, Tx internal state, Rx temp, Tx temp, and reserved areas are only for RISC use.

### 16.14.8 Interrupts from the SCCs

Interrupt handling for each of the SCC channels is configured on a global (per channel) basis in the CPM interrupt pending register, CPM interrupt mask register, and CPM in service register. Within each of these registers, one bit is used to either mask, enable, or report the presence of an interrupt in an SCC channel. The interrupt priority between the four SCCs is programmable in the CP interrupt configuration register. An SCC interrupt can be caused by a number of events. To allow interrupt handling for these (SCC-specific) events, further event registers are provided within the SCCs.

A number of events can cause the SCC to interrupt the processor and the events differ slightly depending on the protocol selected. For a detailed description of the events see the specific protocol paragraphs. These events are handled independently for each channel by the SCC event and mask registers. Events that can cause interrupts due to the $\overline{\text{CTS}}$ and CD modem lines are described in **Section 16.19.9 Port C Pin Functions**.

**16.14.8.1 SCC EVENT REGISTER.** The 16-bit SCC event (SCCE) register is used to report events recognized by any of the SCCs. On recognition of an event, the SCC sets it's corresponding bit in the SCC event register regardless of the corresponding mask bit. To the user it appears as a memory-mapped register that can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. This register is cleared at reset.

**16.14.8.2 SCC MASK REGISTER.** The 16-bit, read/write SCC mask (SCCM) register allows the user to either enable or disable interrupt generation by the CP for specific events in each SCC channel. Notice that an interrupt is only generated if the SCC interrupts in this channel are enabled in the CPM interrupt mask register.

If a bit in the SCCM register is zero, the CP does not proceed with it's usual interrupt handling whenever that event occurs. Anytime a bit in the SCCM register is set, a 1 in the corresponding bit in the SCCE register sets the SCC event bit in the CPM interrupt pending register. The bit position of the SCCM register is identical to that of the SCCE register.

**16.14.8.3 SCC STATUS REGISTER.** The 8-bit, read/write SCC status (SCCS) register allows the user to monitor real-time status conditions (flags, idle, and data carrier sense) on the RXD line. It does not show the real-time status of the $\overline{\text{CTS}}$ and CD pins. Their real-time status is available in the port C parallel I/O.

### 16.14.9 SCC Initialization

The SCCs require a number of registers and parameters to be configured after a power-on reset. The following outline is the proper sequence for initializing the SCCs, regardless of the protocol used.

1. Write the parallel I/O ports to configure and connect the I/O pins to the SCCs.
2. The SDCR should be initialized such that the SDMA arbitration ID is 10 binary.
3. Write the port C registers to configure the $\overline{\text{CTS}}$ and CD pins to be parallel I/O with interrupt capability or to be direct connections to the SCC (if modem support is needed).
4. If the TSA is used, the SI must be configured. If the SCC is used in the NMSI mode, the SICR must still be initialized.
5. Write the GSMR, but do not write the ENT or ENR bits yet.
6. Write the PSMR.
7. Write the DSR.
8. Initialize the required values for this SCC in it’s parameter RAM.
9. Clear out any current events in the SCCE register, if preferred.
10. Write the SCCM register to enable the interrupts in the SCCE register.
11. Write the CICR to configure the SCC interrupt priority.
12. Clear out any current interrupts in the CIPR, if preferred.
13. Write the CIMR to enable interrupts to the CP interrupt controller.
14. Set the ENT and ENR bits in the GSMR.

The buffer descriptors can have their ready/empty bits set at any time. Notice that the CPCR does not need to be accessed following a power-on reset. An SCC should be disabled and reenabled after any dynamic change in its parallel I/O ports or serial channel physical interface configuration. A full reset using the RST bit in the CPCR is a comprehensive reset that can also be used.

### 16.14.10 SCC Interrupt Handling

The following describes what usually occurs within an SCC interrupt handler:

1. Once an interrupt occurs, the SCCE register should be read by the user to see which sources have caused interrupts. The SCCE bits that are going to be “handled” in this interrupt handler would normally be cleared at this time.
2. Process the Tx BDs to reuse them if the TX bit or TXE bit was set in the SCCE register. If the transmit speed is fast or the interrupt delay is long, more than one transmit buffer may have been sent by the SCC. Thus, it is important to check more than just one Tx BD during the interrupt handler. One common practice is to process all Tx BDs in the interrupt handler until one is found with its R-bit set.

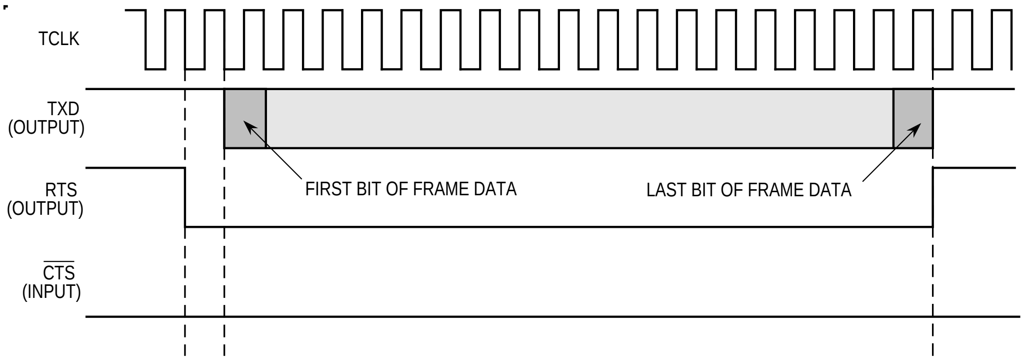
3. Extract data from the Rx BD if the RX, RXB, or RXF bit is set in the SCCE register. If the receive speed is fast or the interrupt delay is long, more than one receive buffer may have been received by the SCC. Thus, it is important to check more than just one Rx BD during the interrupt handler. One common practice is to process all Rx BDs in the interrupt handler until one is found with it's E-bit set.
4. Clear the SCCx bit in the CISR.
5. Execute the RTE instruction.

### 16.14.11 SCC Timing Control

When the DIAG bits of the GSMR are programmed to normal operation, the CD and $\overline{\text{CTS}}$ lines are automatically controlled by the SCC. The TCI bit in the GSMR is assumed to be cleared which implies a normal transmit clock operation.

**16.14.11.1 SYNCHRONOUS PROTOCOLS.** The RTS pin is asserted when the SCC data is loaded into the transmit FIFO and a falling transmit clock occurs. At this point, the SCC begins transmitting the data, once the appropriate conditions occur on the $\overline{\text{CTS}}$ pin. In all cases, the first bit of data is the start of the opening flag, sync pattern, or preamble (if a preamble was programmed to be sent prior to the frame).

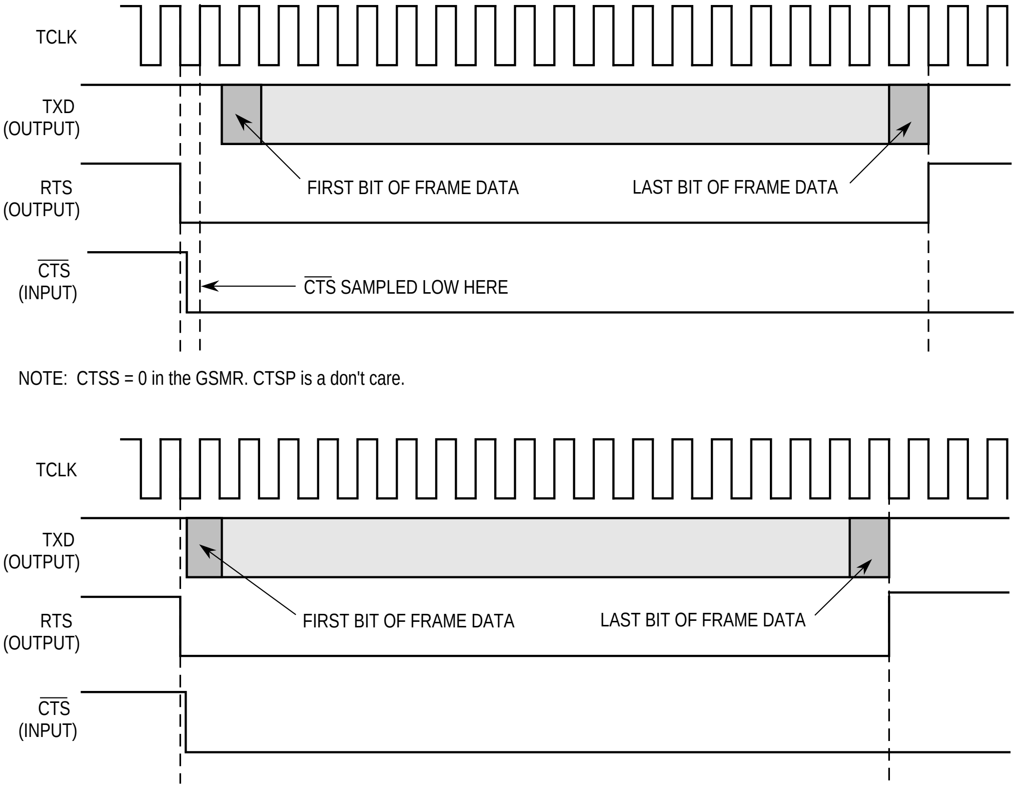
Figure 16-68 illustrates that the delay between RTS and data is 0 bit times, regardless of the CTSS bit in the GSMR. This operation assumes that the $\overline{\text{CTS}}$ pin is already asserted to the SCC or that the $\overline{\text{CTS}}$ pin is reprogrammed to be a parallel I/O line, in which case the $\overline{\text{CTS}}$ signal to the SCC is always asserted. RTS is negated one clock after the last bit in the frame.



NOTE: A frame includes opening and closing flags and syncs, if present in the protocol.

**Figure 16-68. Output Delays from RTS Asserted for Synchronous Protocols**

If the $\overline{\text{CTS}}$ pin is not already asserted when the RTS pin is asserted, then the delays to the first bit of data depend on when $\overline{\text{CTS}}$ is asserted. Figure 16-69 illustrates that the delay between $\overline{\text{CTS}}$ and the data can be approximately 0.5- to 1-bit time or 0 bit times, depending on the CTSS bit in the GSMR.



NOTE: CTSS = 0 in the GSMR. CTSP is a don't care.

NOTE: CTSS = 1 in the GSMR. CTSP is a don't care.

**Figure 16-69. Output Delays from $\overline{\text{CTS}}$ Asserted for Synchronous Protocols**

If the $\overline{\text{CTS}}$ pin is programmed to envelope the data, the $\overline{\text{CTS}}$ pin must remain asserted during frame transmission or a CTS lost error occurs. The negation of the $\overline{\text{CTS}}$ pin forces the $\overline{\text{RTS}}$ pin high and the transmit data to the idle state. If the CTSS bit in the GSMR is zero, the $\overline{\text{CTS}}$ pin must be sampled by the SCC before a CTS lost is recognized. Otherwise, the negation of $\overline{\text{CTS}}$ immediately causes the CTS lost condition. Refer to Figure 16-70 for details.

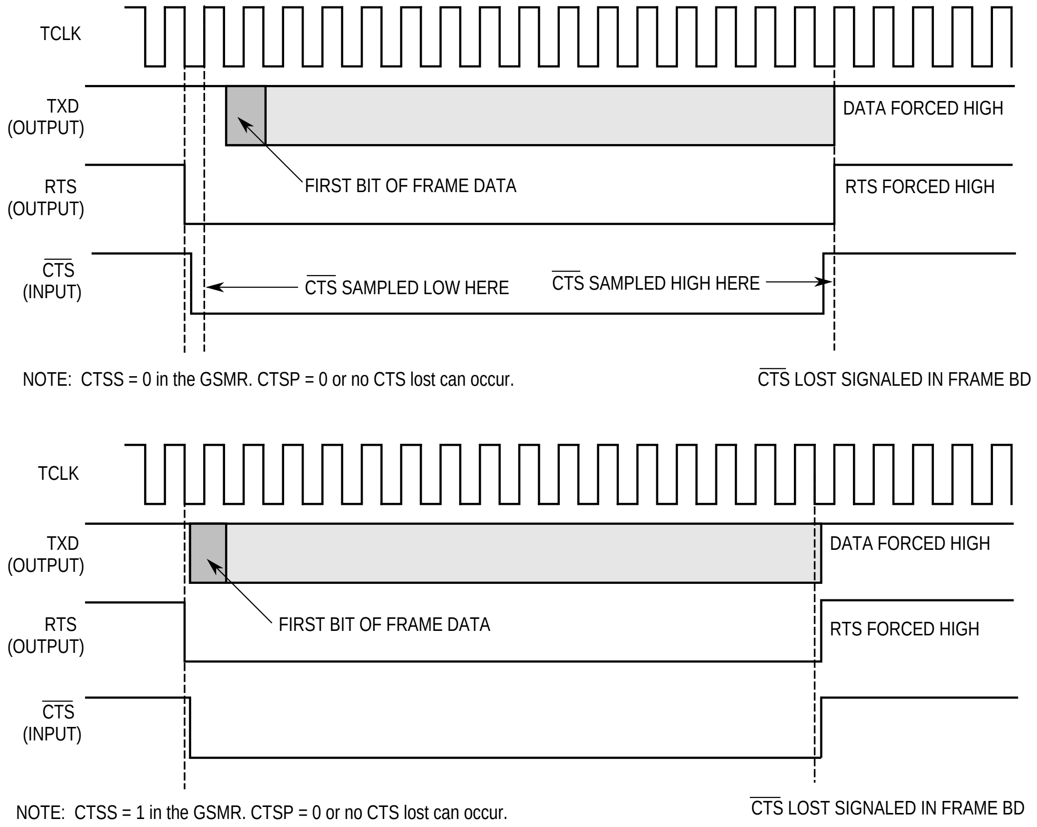
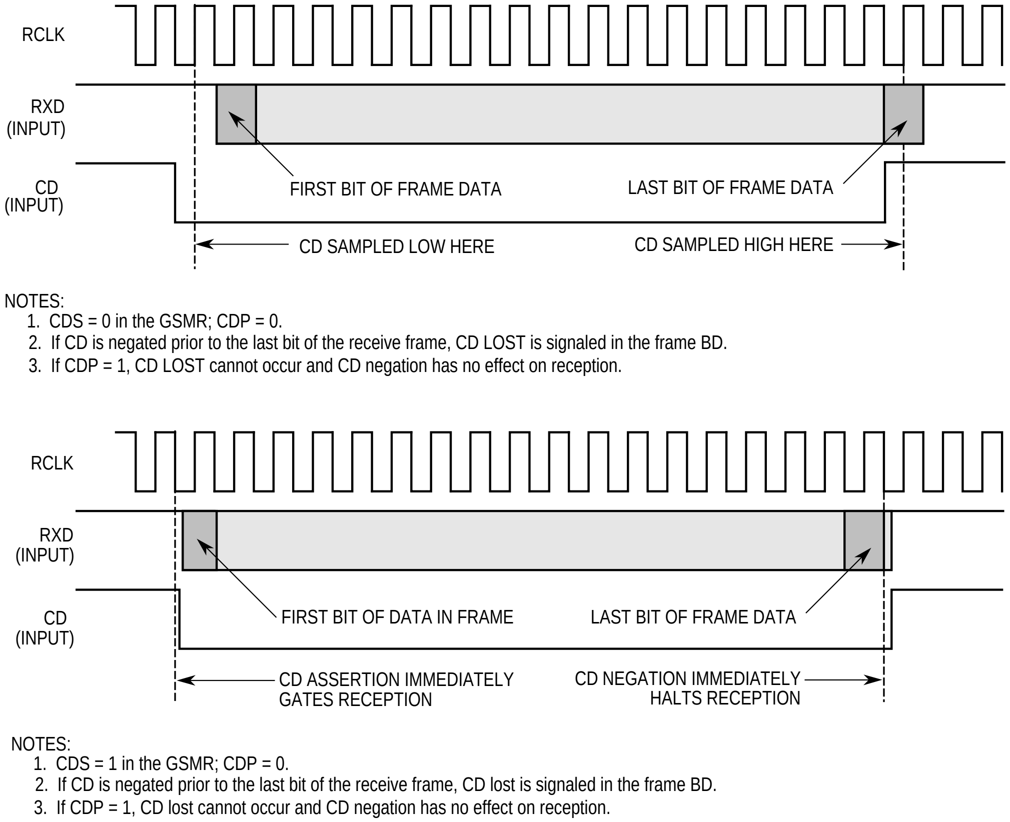


**Figure 16-70. $\overline{\text{CTS}}$ Lost in Synchronous Protocols**

**NOTE**

If the CTSS bit in the GSMR is set, all $\overline{\text{CTS}}$ transitions must occur while the transmit clock is low.

Reception delays are determined by the CD pin as illustrated in Figure 16-71. If the CDS bit in the GSMR is zero, then the CD pin is sampled on the rising receive clock edge prior to data being received. If the CDS bit in the GSMR is 1, then the CD pin transitions cause data to be immediately gated into the receiver.



NOTES:
1. CDS = 0 in the GSMR; CDP = 0.
2. If CD is negated prior to the last bit of the receive frame, CD LOST is signaled in the frame BD.
3. If CDP = 1, CD LOST cannot occur and CD negation has no effect on reception.

NOTES:
1. CDS = 1 in the GSMR; CDP = 0.
2. If CD is negated prior to the last bit of the receive frame, CD lost is signaled in the frame BD.
3. If CDP = 1, CD lost cannot occur and CD negation has no effect on reception.

**Figure 16-71. Using CD to Control Reception of Synchronous Protocols**

If the CD pin is programmed to envelope the data, the CD pin must remain asserted during frame transmission or a CD lost error occurs. The negation of the CD pin terminates reception. If the CDS bit in the GSMR is zero, the CD pin must be sampled by the SCC before a CD lost is recognized. Otherwise, the negation of CD immediately causes the CD lost condition.

### NOTE

> If the CDS bit in the GSMR is set, all CD transitions must occur while the receive clock is low.

**16.14.11.2 ASYNCHRONOUS PROTOCOLS.** The RTS pin is asserted when the SCC data is loaded into the transmit FIFO and a falling transmit clock occurs. The CD and $\overline{\text{CTS}}$ pins can be used to control reception and transmission in the same manner as the synchronous protocols. The first bit of data transmission in an asynchronous protocol is the start bit of the first character. In addition, the UART protocol has an option for $\overline{\text{CTS}}$ flow control as described in **Section 16.14.16 UART Controller**.

If $\overline{\text{CTS}}$ is already asserted when RTS is asserted, transmission begins in two additional bit times. If $\overline{\text{CTS}}$ is not already asserted when RTS is asserted and CTSS = 0, then transmission begins in three additional bit times. If $\overline{\text{CTS}}$ is not already asserted when RTS is asserted and CTSS = 1, then transmission begins in two additional bit times.

## 16.14.12 Digital Phase-Locked Loop

**16.14.12.1 DATA ENCODING.** Each SCC contains a digital phase-locked loop (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. Examples of the different encoding methods are illustrated in Figure 16-72.

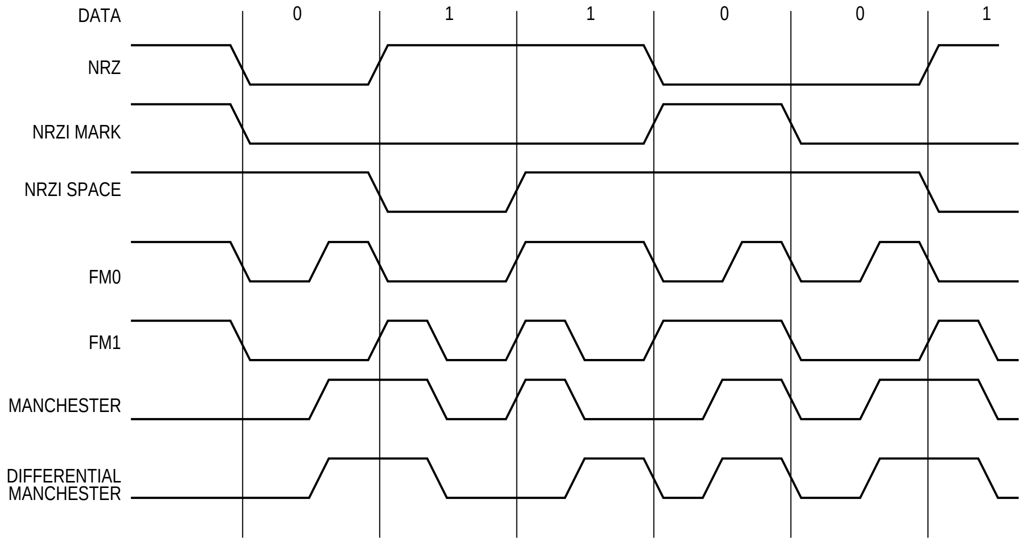


**Figure 16-72. DPLL Encoding Examples**

If it is not preferable to use the DPLL, the user can choose the NRZ coding in the GSMR. The definition of the encodings are as follows:

- NRZ—A one is represented by a high level for the duration of the bit and a zero is represented by a low level.
- NRZI Mark—A one is represented by no transition at all. A zero is represented by a transition at the beginning of the bit (the level present in the preceding bit is reversed).
- NRZI Space—A one is represented by a transition at the beginning of the bit (the level present in the preceding bit is reversed). A zero is represented by no transition at all.
- FM0—A one is represented by a transition only at the beginning of the bit. A zero is represented by a transition at the beginning of the bit and another transition at the center of the bit.

- FM1—A one is represented by a transition at the beginning of the bit and another transition at the center of the bit. A zero is represented by a transition only at the beginning of the bit.
- Manchester—A one is represented by a high to low transition at the center of the bit. A zero is represented by a low to high transition at the center of the bit. In both cases there may be a transition at the beginning of the bit to set up the level required to make the correct center transition.
- Differential Manchester—A one is represented by a transition at the center of the bit with the opposite direction from the transition at the center of the preceding bit. A zero is represented by a transition at the center of the bit with the same polarity from the transition at the center of the preceding bit.

**16.14.12.2 DPLL OPERATION.** Each SCC channel includes a DPLL to recover clock information from a received datastream. For applications that provide a direct clock source to the SCC, the DPLL can be bypassed as programmed in the GSMR. However, the DPLL must not be used when an SCC is programmed to Ethernet, but it is optional for other protocols. The DPLL receive block diagram is illustrated in Figure 16-73 and the transmit block diagram in Figure 16-74.

The DPLL is either driven by an external clock or one of the baud rate generator outputs. This clock should be approximately 8×, 16×, or 32× times the data rate, depending on the encoding/decoding preferred. The DPLL uses this clock, along with the datastream, to construct a data clock that can be used as the SCC receive and/or transmit clock. In all modes, the DPLL uses the input clock to determine the nominal bit time.

At the beginning of operation, the DPLL is in search mode. In this mode, the first transition resets the internal DPLL counter and begins DPLL operation. While the counter is counting, the DPLL watches the incoming datastream for transitions. Whenever a transition is detected, the DPLL makes a count adjustment to produce an output clock that tracks the incoming bits.

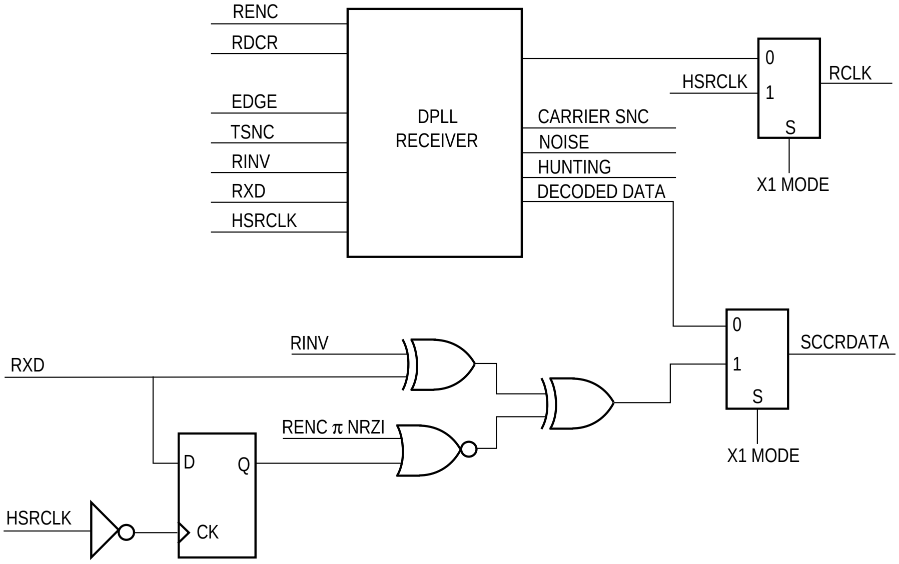


Figure 16-73. DPLL Receive Block Diagram

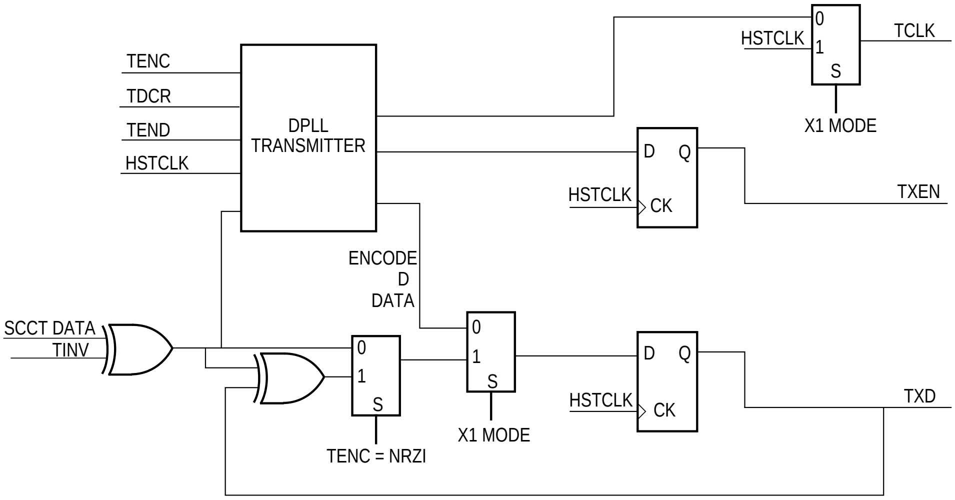


Figure 16-74. DPLL Transmit Block Diagram

The DPLL provides a carrier-sense signal that indicates when there are data transfers on the RXD line. It is asserted as soon as a transition is detected on the RXD line and negated after a programmable number of clocks have been detected with no transitions, using the TSNC bits in the GSMR.

To prevent the DPLL from locking on the wrong edges and to provide a fast synchronization, the DPLL should receive a preamble pattern prior to the data. In some protocols, the preceding flags or syncs are used. However, some protocols require a special pattern, such as alternating ones and zeros. For the case of transmission, the SCC has an option to generate preamble patterns as programmed in the TPP and TPL bits of the GSMR.

**Table 16-24. Preamble Requirement**

| DECODING METHOD | PREAMBLE PATTERN | MAX PREAMBLE LENGTH REQUIRED |
|---|---|---|
| NRZI Mark | All zeros | 8-bits |
| NRZI Space | All ones | 8-bits |
| FM0 | All ones | 8-bits |
| FM1 | All zeros | 8-bits |
| Manchester | Repeating 10's | 8-bits |
| Differential Manchester | All ones | 8-bits |

In addition, the DPLL can be used to invert the datastream on receive or transmit. This feature is available in all encodings, including the standard NRZ data format. The DPLL also offers a choice on the transmitter during idle of whether to force the TXD line to a high voltage or to continue encoding the data supplied to it. The DPLL is used for the UART encoding/decoding which gives the user the option to select the divide ratio used in the UART decoding process (8, 16, or 32). Usually, the 16× option is chosen.

The maximum data rate that can be supported with the DPLL is 3.125 MHz when working with a 25-MHz system clock, which assumes the 8× option is chosen: 25 MHz/8 = 3.125 MHz. Thus, the frequency applied to the CLKx pin or generated by an internal baud rate generator may be up to 25 MHz on a 25 MHz MPC821, if the DPLL 8×, 16×, or 32× options are used.

**NOTE**

The 1:2 ratio of the system clock to the serial clock does not apply when the DPLL is used to recover the clock in the 8×, 16×, or 32× modes. The synchronization actually occurs internally after the receive clock is generated by the DPLL, therefore, even the fastest DPLL clock generation (the 8× option) easily meets the required 1:2 ratio clocking limit.

**16.14.12.3 SERIAL INFRA-RED ENCODER/DECODER.** IRDA is a family of specifications intended to facilitate the interconnection of computers and peripherals using a directed half duplex serial infrared physical communications medium. The data link layer protocol is based on a preexisting standard asynchronous HDLC protocol. Refer to **Section 16.14.19 ASYNC HDLC Controller** for more details. A block of one end of the overall serial infra-red (SIR) link is illustrated in Figure 16-75.

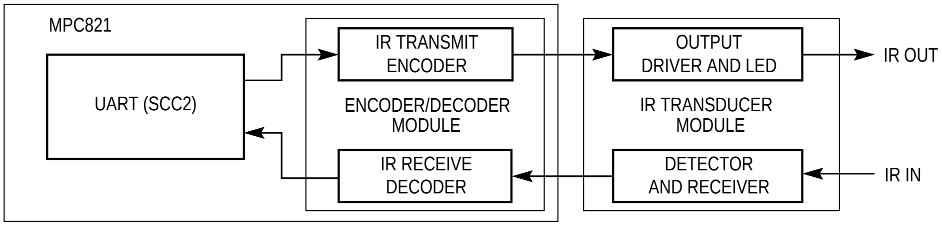


**Figure 16-75. Example of One End of SIR Link**

The signal between the UART and the encoder/decoder is a bitstream of pulses in a frame comprised of a start bit, 8 data bits, no parity bit, and ending with a stop bit, as illustrated in Figure 16-76(A). The signal between the encoder/decoder module and IR transducer module is illustrated in Figure 16-76(B). The electrical pulses between the IR transmit encoder and the output driver and LED are $^{3}/_{16}$ of a bit period in duration (or for the slower bit rates, as short as $^{3}/_{16}$ of the bit period for 115.2 kbits/second). The electrical pulses between the detector and receiver and the IR receive decoder are nominally the same duration as those between the IR transmit encoder, output driver, and LED.

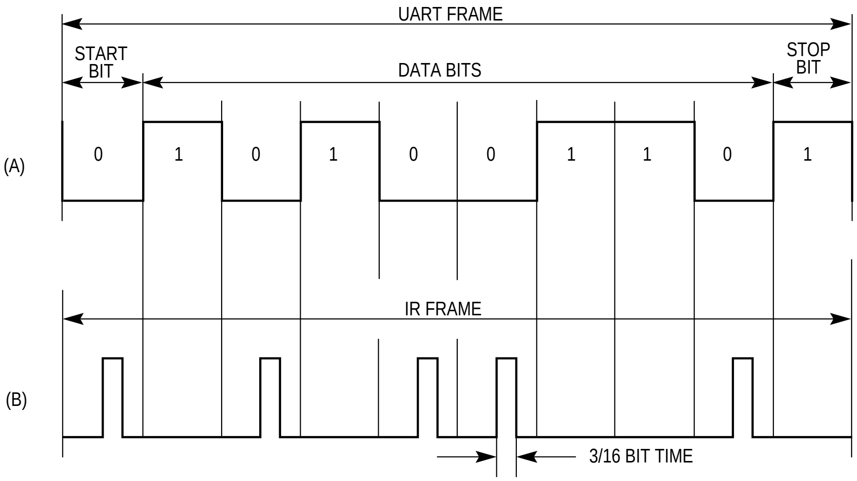


**Figure 16-76. UART and IR Frames**

The SIR encoding/decoding is supported only for SCC2. To activate it, the SIR bit in the GSMR should be set to 1 and the RDCR and TDCR fields in the GSMR should be configured for x16 clock operation.

### 16.14.13 Clock Glitch Detection

A clock glitch occurs when an input clock signal transitions between a one and zero state two times, within a small enough time period to violate the minimum high/low time specification of the input clock. Spikes are one type of glitch. Additionally, glitches can occur when excessive noise is present on a slowly rising/falling signal.

Glitched clocks can be a potential problem for many communications systems. Not only can they cause systems to experience errors, but they can cause errors without being detected by the system. Systems that supply an external clock to a serial channel are often susceptible to glitches from situations like noise, connecting/disconnecting the physical cable from the application board, or excessive ringing on the clock lines.

The SCCs on the MPC821 have a special circuit designed to detect glitches that can occur in the system. The glitch circuit is designed to detect glitches that could cause the SCC to transition to the wrong state. This status information can be used to alert the system of a problem at the physical layer. The glitch detect circuit is not a specification test. Thus, if the user develops a circuit that does not meet the input clocking specifications for the SCCs, erroneous data can be received/transmitted that is not indicated by the glitch detection logic. Conversely, if a glitch indication is signaled, it does not guarantee that erroneous data was received/transmitted. Regardless of whether the DPLL is used, the received clock is passed through a noise filter that eliminates any noise spikes that affect a single sample. This sampling is enabled with the GDE bit of the GSMR.

If a spike is detected, a maskable receive or transmit glitched clock interrupt is generated in the event register of the SCC channel. Although the user can choose to reset the SCC receiver or transmitter or continue operation, the statistics on clock glitches should be kept for later evaluation. The glitched status indication can also be used as a debugging aid during the early phases of prototype testing.

### 16.14.14 Disabling the SCCs On-the-Fly

If an SCC is not needed for a while, it can be disabled and reenabled later. In this case, a sequence of operation is followed.These sequences ensure that any buffers in use is properly closed and that new data is transferred to/from a new buffer. Such a sequence is required if the parameters that must be changed are not allowed to be changed dynamically. If the register or bit description states that dynamic changes are allowed, the following sequences are not required and the register or bit may be changed immediately. In all other cases, the sequence should be used. For instance, the internal baud rate generators allow on-the-fly changes, but the DPLL-related bits in the GSMR do not.

**NOTE**

The modification of parameter RAM does not require a full disabling of the SCC. Refer to the parameter RAM description for details on when parameter RAM values may be modified. If the user prefers to disable all SCCs, SMCs, SPI, and the I$^2$C, the CPCR can be used to reset the entire CP with a single command.

**16.14.14.1 SCC TRANSMITTER FULL SEQUENCE.** For the SCC transmitter, the full disable and enable sequence is as follows.

1. Issue the STOP TRANSMIT command. This command is recommended if the SCC is currently in the process of transmitting data because it stops transmission in an orderly way. If the SCC is not transmitting (no Tx BDs are ready or the GRACEFUL STOP TRANSMIT command has been issued and completed), then the STOP TRANSMIT command is not required. Furthermore, if the TBPTR is overwritten by the user or the INIT TX PARAMETERS command is executed, this command is not required.
2. Clear the ENT bit in the GSMR. This disables the SCC transmitter and puts it in a reset state.
3. Make modifications. The user can make modifications to the SCC transmit parameters, including the parameter RAM. If the user wants to switch protocols or restore the SCC transmit parameters to their initial state, the INIT TX PARAMETERS command can be issued.
4. Issue the RESTART TRANSMIT command. This command is required if the INIT TX PARAMETERS command was not issued in step 3.
5. Set the ENT bit in the GSMR. Transmission begins using the Tx BD that the TBPTR points to as soon as the Tx BD R-bit is set.

**16.14.14.2 SCC TRANSMITTER SHORTCUT SEQUENCE.** A shorter sequence is possible if the user prefers to reinitialize the transmit parameters to the state they had after reset. This sequence is as follows.

1. Clear the ENT bit in the GSMR.
2. Issue the INIT TX PARAMETERS command. Any additional modifications may now be made.
3. Set the ENT bit in the GSMR.

**16.14.14.3 SCC RECEIVER FULL SEQUENCE.** The full disable and enable sequence for the receiver is as follows.

1. Clear the ENR bit in the GSMR. Reception is aborted immediately. This disables the receiver of the SCC and puts it in a reset state.
2. Make modifications. The user can make modifications to the SCC receive parameters, including the parameter RAM. If the user prefers to switch protocols or restore the SCC receive parameters to their initial state, the INIT RX PARAMETERS command can be issued.
3. Issue the ENTER HUNT MODE command. This command is required if the INIT RX PARAMETERS command was not issued in step 2.
4. Set the ENR bit in the GSMR. Reception begins immediately using the Rx BD that the RBPTR points to if the Rx BD E-bit is set.

**16.14.14.4 SCC RECEIVER SHORTCUT SEQUENCE.** A shorter sequence is possible if the user prefers to reinitialize the receive parameters to the state they had after reset. This sequence is as follows.

1. Clear the ENR bit in the GSMR.
2. INIT RX PARAMETERS command. Any additional modifications may now be made.
3. Set the ENR bit in the GSMR.

**16.14.14.5 SWITCHING PROTOCOLS.** Sometimes the user wants to switch the protocol that the SCC is executing (UART to HDLC) without resetting the board or affecting any other SCC. This can be accomplished by using only one command and a short number of steps.

1. Clear the ENT and ENR bits in the GSMR.
2. INIT TX AND RX PARAMETERS command. This one command initializes both transmit and receive parameters. Any additional modifications can be made in the GSMR to change the protocol.
3. Set the ENT and ENR bits in the GSMR. The SCC is enabled with the new protocol.

### 16.14.15 Saving Power

When the ENT and ENR bits of an SCC are cleared, that the SCC consumes a minimal amount of power.

### 16.14.16 UART Controller

Many applications need a simple method of communicating low-speed data between equipment. The universal asynchronous receiver transmitter (UART) protocol is the de-facto standard for such communications. The term asynchronous is used because it is not necessary to send clocking information with the data that is sent. UART links are typically 38400 baud or less in speed and are character oriented (the smallest unit of data that can be correctly received or transmitted is a character). Typical applications of asynchronous links are connections between terminals and computer equipment. Even in applications where synchronous communications are required, the UART is often used for a local debugging port to run board debugger software. The character format of the UART protocol is illustrated in Figure 16-77.

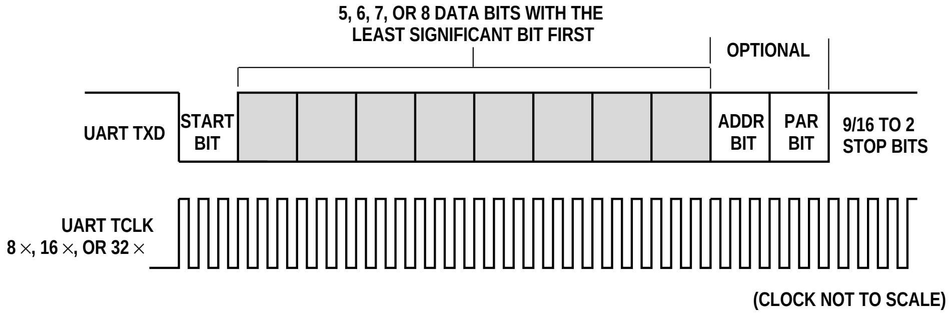


Figure 16-77. UART Character Format

Since the transmitter and receiver operate asynchronously, there is no need to connect transmit and receive clocks. Instead, the receiver oversamples the incoming datastream (usually by a factor of 16) and uses some of these samples to determine the bit value. Traditionally, the middle three samples of the 16 samples are used. Two UARTs can communicate using a system like this if parameters, such as the parity scheme and character length, are the same for both transmitter and receiver.

When data is not transmitted in the UART protocol, a continuous stream of ones is transmitted, called the idle condition. Since the start bit is always a zero, the receiver can detect when real data is once again present on the line. UART also specifies an all-zeros character called a break, which is used to abort a character transfer sequence.

Many different protocols have been defined using asynchronous characters, but the most popular of these is the RS-232 standard. RS-232 specifies standard baud rates, handshaking protocols, and mechanical/electrical details. Another popular standard using the same character format is RS-485, which defines a balanced line system allowing longer cables than RS-232 links. Synchronous protocols like HDLC are sometimes defined to run over asynchronous links. Other protocols like Profibus extend the UART protocol to include LAN-oriented features such as token passing.

All the standards provide handshaking signals, but some systems require just three physical lines—transmit data, receive data, and ground. Many proprietary standards have been built around the asynchronous character frame, and some even implement a multidrop configuration. In multidrop systems, more than two stations can be present on a network, each one having a specific address. Frames made up of many characters may be broadcast, with the first character acting as a destination address. To allow this, the UART frame is extended by one bit to distinguish between an address character and the normal data characters. Additionally, a synchronous form of the UART protocol exists where start and stop bits are still present, but a clock is provided with each bit, so the oversampling technique is not required. This mode is called “isochronous” operation or, more often, synchronous UART.

By appropriately setting the GSMR, any of the SCC channels can be configured to function as a UART. The UART provides standard serial I/O using asynchronous character-oriented (start-stop) protocols with RS-232C-type lines. The UART can be used to communicate with any existing RS-232-type device and to provides a port for serial communication to other microprocessors and terminals (either locally or via modems). It includes facilities for communication using standard asynchronous bit rates and protocols. The UART device supports a multidrop mode for master/slave operation with wake-up capability on both idle line and address bit. The UART also supports a synchronous mode of operation where a clock must be provided with each bit received.The UART transmits data from memory (either internal or external) to the TXD line and receives data from the RXD line into memory. In a synchronous UART mode, the clock must also be supplied. It may be generated internally or externally. Modem lines are supported via the port C pins. The UART consists of separate transmit and receive sections whose operations are asynchronous with the CPU core.

**16.14.16.1 FEATURES.** The following is a list of the UART’s important features:

- Flexible message-oriented data structure
- Implements synchronous and asynchronous UART
- Multidrop operation
- Receiver wake-up on idle line or address mode
- Eight control character comparison
- Two address comparison
- Maintenance of four 16-bit error counters
- Received break character length indication
- Programmable data length (5–8 bits)
- Programmable 1 to 2 stop bits in transmission
- Capable of reception without a stop bit
- Programmable fractional stop bit length
- Even/odd/force/no parity generation
- Even/odd/force/no parity check

- Frame error, noise error, break, and idle detection
- Transmit preamble and break sequences
- Freeze transmission option with low-latency stop

**16.14.16.2 NORMAL ASYNCHRONOUS MODE.** In normal asynchronous mode, the receive shift register receives the incoming data on the RXDx pin. The length and format of the UART character is defined by the control bits in the UART mode register and the order of reception is as follows:

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 times faster then the baud rate and samples each bit of the incoming data three times around its center. The value of the bit is determined by the majority of those samples and if they do not all agree, a noise indication counter is incremented. When a complete byte has been clocked in, the contents of the shift register are transferred to the UART receive data register. If there is an error in this character, the appropriate error bits is set by the CP.

The UART can receive fractional stop bits. The next character's start bit may begin any time after the three middle samples have been taken. The UART transmit shift register transmits the outgoing data on the TXDx pin. Data is clocked synchronously with the transmit clock, which may have either an internal or external source. The order of bit transmission is LSB first. Only the data portion of the UART frame is actually stored in the data buffers. The start and stop bits are always generated and stripped by the UART controller. The parity bit can also be generated in transmission and checked during reception. Although parity is not stored in the data buffer, it's value can be inferred from the reporting mechanism in the data buffer (character with parity errors are identified). Similarly, the optional address bit is not stored in the transmit or receive data buffer, but is implied from the buffer descriptor itself. Parity is generated and checked for the address bit, when present. The RFW bit in the GSMR must be set for an 8-bit receive FIFO for the UART receiver.

**16.14.16.3 SYNCHRONOUS MODE.** In synchronous mode, the UART controller uses the 1× data clock for timing. The receive shift register receives the incoming data on the RXD pin synchronously to the clock. The length and format of the serial word in bits are defined by the control bits in the UART mode register in the same manner as for asynchronous mode. When a complete byte has been clocked in, the contents of the shift register are transferred to the UART receive data register. If there is an error in this character, then the appropriate error bits is set by the CP.

The UART transmit shift register transmits the outgoing data on the TXD pin. Data is clocked synchronously with the transmit clock, which may have either an internal or external source. The RFW bit in the GSMR must be set for an 8-bit receive FIFO for the UART receiver.

**16.14.16.4 UART MEMORY MAP.** When configured to operate in UART mode, the MPC821 overlays the structure listed in Table 16-23 with the UART-specific parameters described in Table 16-25.

### Table 16-25. UART-Specific Parameters

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SCC Base + 30 | RES | Word | Reserved |
| SCC Base + 34 | RES | Word | Reserved |
| SCC Base + 38 | **MAX_IDL** | Half-word | MaximumIdle Characters |
| SCC Base + 3A | IDLC | Half-word | Temporary idle Counter |
| SCC Base + 3C | **BRKCR** | Half-word | Break Count Register (Transmit) |
| SCC Base + 3E | **PAREC** | Half-word | Receive Parity Error Counter |
| SCC Base + 40 | **FRMEC** | Half-word | Receive Framing Error Counter |
| SCC Base + 42 | **NOSEC** | Half-word | Receive Noise Counter |
| SCC Base + 44 | **BRKEC** | Half-word | Receive Break Condition Counter |
| SCC Base + 46 | BRKLN | Half-word | Last Received Break Length |
| SCC Base + 48 | **UADDR1** | Half-word | UART Address Character 1 |
| SCC Base + 4A | **UADDR2** | Half-word | UART Address Character 2 |
| SCC Base + 4C | RTEMP | Half-word | Temp Storage |
| SCC Base + 4E | **TOSEQ** | Half-word | Transmit Out-of-Sequence Character |
| SCC Base + 50 | **CHARACTER1** | Half-word | Control Character 1 |
| SCC Base + 52 | **CHARACTER2** | Half-word | Control Character 2 |
| SCC Base + 54 | **CHARACTER3** | Half-word | Control Character 3 |
| SCC Base + 56 | **CHARACTER4** | Half-word | Control Character 4 |
| SCC Base + 58 | **CHARACTER5** | Half-word | Control Character 5 |
| SCC Base + 5A | **CHARACTER6** | Half-word | Control Character 6 |
| SCC Base + 5C | **CHARACTER7** | Half-word | Control Character 7 |
| SCC Base + 5E | **CHARACTER8** | Half-word | Control Character 8 |
| SCC Base + 60 | **RCCM** | Half-word | Receive Control Character Mask |
| SCC Base + 62 | RCCR | Half-word | Receive Control Character Register |
| SCC Base + 64 | RLBC | Half-word | Receive Last Break Character |

NOTE: Items in bold must be initialized by the user.
SCC base = IMMR + 1C00 (SCC1) or 1D00 (SCC2).

- MAX_IDL—Once a character is received on the line, the UART controller begins counting any idle characters received. If a MAX_IDL number of idle characters is received before the next data character is received, an idle timeout occurs and the buffer is closed. This, in turn, can produce an interrupt request to the CPU core to receive the data from the buffer. Thus, MAX_IDL provides a convenient way to demarcate frames in the UART mode. To disable the MAX_IDL feature, simply program it to $0000.

  A character of idle is calculated as the following number of bit times: 1 + data length (5, 6, 7, 8, or 9) + 1 (if parity bit is used) + number of stop bits (1 or 2). Example: for 8 data bits, no parity, and 1 stop bit, the character length is 10 bits.

- IDLC—This value is used by the RISC to store the current idle counter value in the MAX_IDL timeout process. IDLC is a down-counter and it does not need to be initialized or accessed by the user.
- BRKCR—The UART controller sends a break character sequence whenever a STOP TRANSMIT command is given. The number of break characters sent by the UART controller is determined by the value in BRKCR. In the case of 8 data bits, no parity, 1 stop bit, and 1 start bit, each break character is 10 bits in length and consists of all zeros.
- PAREC, FRMEC, NOSEC, and BRKEC—These 16-bit (modulo–$2^{16}$) counters are initialized by the user. When the associated condition occurs, they are incremented by the RISC controller.

  — PAREC counts received parity errors.
  — FRMEC counts received characters with framing errors.
  — NOSEC counts received characters with noise errors (one of the three samples was different).
  — BRKEC counts the number of break conditions that occurs on the line.

  Notice that one break condition may last for hundreds of bit times, yet this counter is incremented only once during that period.

- BRKLN—This value is used to store the length of the last break character that is received and is the length in characters of the break. For example, if the receive pin is low for 20 bit times, BRKLN shows the value $0010. BRKLN is accurate to within one character unit of bits. For 8 data bits, no parity, 1 stop bit, and 1 start bit, BRKLN is accurate to within 10 bits.
- UADDR1, UADDR2—In the multidrop mode, the UART controller provides automatic address recognition for two addresses. In this case, the user programs the lower order bytes of UADDR1 and UADDR2 with the two preferred addresses.
- TOSEQ—This value is used to transmit out-of-sequence characters (XOFF and XON) in the transmit stream. Using this field, the preferred characters can be inserted into the transmit FIFO without affecting any transmit buffer that might currently be in progress.
- CHARACTER1–8—These characters define the receive control characters on which interrupts can be generated.

- RCCM—This value is used to mask the comparison of CHARACTER1–8 so that classes of control characters can be defined. A 1 enables the bit comparison and a zero masks it.
- RCCR—This value is used to hold the value of any control character that is not written to the data buffer.
- RLBC—This entry is used in synchronous UART, when the RZS bit is set in the PSMR and contains the actual pattern of the last break character. By counting the zeros in this entry, the CPU core can measure the break length to a bit resolution. The user reads RLBC by counting the number of zeros written, starting at bit 15 down to the point where the first one is written. Therefore, RLBC = 001xxxxxxxxxxxxx (binary) indicates two zeros and RLBC = 1xxxxxxxxxxxxxxx (binary) indicates no zeros.

**16.14.16.5 UART PROGRAMMING MODEL.** An SCC configured as a UART uses the same data structure as the other modes and it supports multibuffer operation. The UART can be programmed to perform address comparison whereby messages not destined for a given programmable address are discarded. Also, the user can program the UART to accept or reject control characters. If a control character is rejected, an interrupt can be generated. The receive character can be accepted using a receive character mask value. The UART enables the user to transmit break and preamble sequences. Overrun, parity, noise, and framing errors are reported via the BD table and/or error counters. An indication of the status of the line (idle) is reported through the status register and a maskable interrupt is generated upon a status change. In it's simplest form, the UART functions in a character-oriented environment. Each character is transmitted with accompanied stop bits and parity (as configured by the user) and received into separate 1-byte buffers. Reception of each buffer can generate a maskable interrupt.

Many applications want to take advantage of the message-oriented capabilities supported by the UART by using linked buffers (in either receive or transmit). In this case, data is handled in a message-oriented environment which means that users can work on entire messages rather than operating on a character-by-character basis. A message may span several linked buffers. For example, before handling the input data, a terminal driver may want to wait until an end-of-line character has been typed by the user rather than being interrupted when a character is received.

As another example, when transmitting ASCII files, the data be transferred as messages ending on the end-of-line character. Each message could be both transmitted and received as a linked list of buffers without any intervention from the CPU, which makes it easy to program and saves processor overhead. On the receive side, the user can define up to eight control characters and each control character can be configured to designate the end of a message or generate a maskable interrupt without being stored in the data buffer. The latter option is useful when flow control characters such as XON or XOFF need to alert the CPU, yet do not belong to the received message.

**16.14.16.6 COMMAND SET.** The following transmit and receive commands are issued to the CPCR.

#### 16.14.16.6.1 Transmit Commands.

**STOP TRANSMIT**

After a hardware or software reset and the enabling of the channel in the SCC mode register, the channel is in the transmit enable mode and starts polling the first BD in the table every 8 transmit clocks (immediately if the TOD bit in the TODR is set).

This command disables the transmission of characters on the transmit channel. If this command is received by the UART controller during message transmission, the transmission is aborted. The UART completes the transmission of all data already transferred to it’s FIFO and stops transmitting it. The TBPTR is not advanced.

The UART transmitter transmits a programmable number of break sequences and start to transmit idles. The number of break sequences (which can be zero) should be written to the break count register before this command is given to the UART controller.

**GRACEFUL STOP TRANSMIT**

This command is used to stop transmission in an orderly way rather than abruptly, as performed by the regular STOP TRANSMIT command. It stops transmission after the current buffer has completed transmission, or immediately if there is no buffer being transmitted. The GRA bit in the SCCE register is set once transmission stops. Then the UART transmit parameters, including BDs, can be modified. The TBPTR points to the next Tx BD in the table. Transmission begins once the R-bit of the next BD is set and the RESTART TRANSMIT command is issued.

**RESTART TRANSMIT**

This command enables the transmission of characters on the transmit channel. This command is expected by the UART controller after disabling the channel in it’s SCC mode register, after a STOP TRANSMIT command, after a GRACEFUL STOP TRANSMIT command, or after a transmitter error (underrun or CTS lost). The UART controller resumes transmission from the current TBPTR in the channel’s Tx BD table.

**INIT TX PARAMETERS**

This command initializes all transmit parameters in the serial channel’s parameter RAM to their reset state. This command should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS command can also be used to reset both transmit and receive parameters.

#### 16.14.16.6.2 Receive Commands.

**ENTER HUNT MODE**

After the hardware or software is reset and the channel is enabled in the SCC mode register, the channel is in receive enable mode and uses the first BD in the table.

This command forces the UART controller to close the current Rx BD if it is being used and enter the hunt mode. The UART controller resumes reception to the next BD if a message is in progress.

In the multidrop hunt mode, the UART controller continually scans the input datastream for the address character. When not in multidrop mode, the UART controller waits for the idle sequence (one character of idle) without losing any data that was in the receive FIFO when this command was executed.

**CLOSE Rx BD**

This command forces the SCC to close the Rx BD if it is currently being used and uses the next BD for any subsequently received data. If the SCC is not in the process of receiving data, no action is taken by this command.

**NOTE**

The CLOSE Rx BD command in UART mode does the same job as the ENTER HUNT MODE command, except for one distinction. The CLOSE Rx BD does not require that a character of idle be present on the line for reception to continue.

**INIT RX PARAMETERS**

This command initializes all receive parameters in this serial channel's parameter RAM to their reset state. This command should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the receive and transmit parameters.

**16.14.16.7 UART ADDRESS RECOGNITION.** In multidrop systems, more than two stations can be present on a network and each one can have a specific address. Figure 16-78 illustrates two examples of this configuration. Frames made up of many characters can be broadcast, with the first character acting as a destination address. To achieve this, the UART frame is extended by one bit, called the address bit, to distinguish between an address character and the normal data characters.

The UART can be configured to operate in a multidrop environment that supports the following two modes:

- Automatic Multidrop Mode**—**The UART controller automatically checks the incoming address character and accepts the data following it, but only if the address matches one of two preset values.
- Nonautomatic Multidrop Mode**—**The UART controller receives all characters. An address character is always written to a new buffer (it can be followed by data characters).

Each UART controller has two 16-bit address registers to support address recognition, UADDR1, and UADDR2. The upper 8 bits of these registers should be written with zero; only the lower 8 bits are used. In the automatic mode, the incoming address is checked against UADDR1 and UADDR2. Upon an address match, the M-bit in the BD is set to indicate which address character was matched and the data following it is written to the data buffers.

**NOTE**

For characters less than 8 bits, the MSBs should be zero.

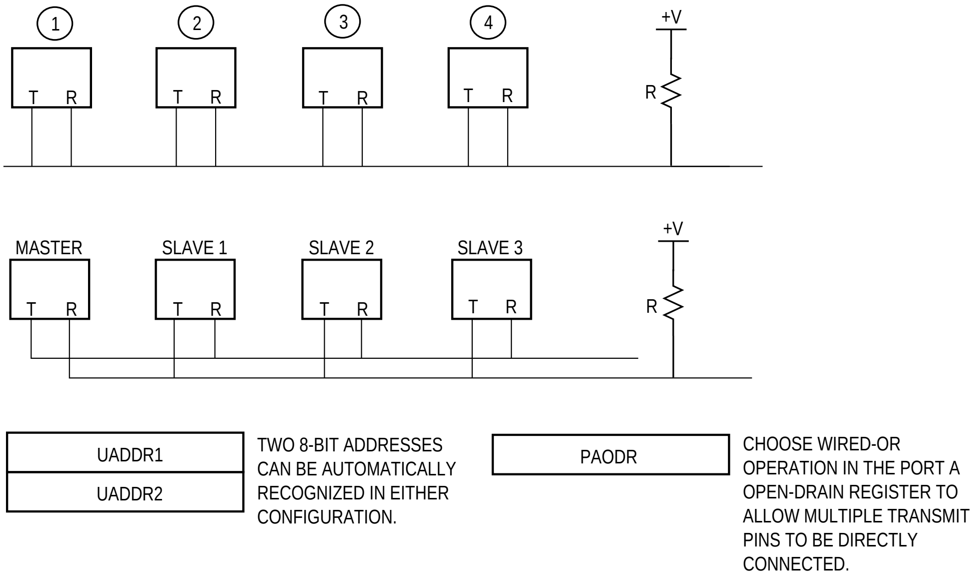

1
2
3
4
+V
T R
T R
T R
T R
R
MASTER
SLAVE 1
SLAVE 2
SLAVE 3
+V
T R
T R
T R
T R
R
UADDR1
UADDR2
TWO 8-BIT ADDRESSES CAN BE AUTOMATICALLY RECOGNIZED IN EITHER CONFIGURATION.
PAODR
CHOOSE WIRED-OR OPERATION IN THE PORT A OPEN-DRAIN REGISTER TO ALLOW MULTIPLE TRANSMIT PINS TO BE DIRECTLY CONNECTED.

**Figure 16-78. Two Configurations of UART Multidrop Operation**

**16.14.16.8 UART CONTROL CHARACTERS (RECEIVER).** The UART has the capability to recognize special control characters and can be used when the UART is in a message-oriented environment. Up to eight control characters can be defined by the user in the control characters table. Each character can either be written to the receive buffer (on which the buffer is closed and a new receive buffer taken) or rejected. If rejected, the character is written to the received control character register (RCCR) in internal RAM and a maskable interrupt is generated. This method is useful for notifying the user of the arrival of control characters (XOFF) that are not part of the received messages. The UART uses a table of 16-bit entries to support control character recognition and each entry consists of the control character, a valid bit, and a reject character bit.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | E | R | | | | | | | CHARACTER1 | | | | | | | |
| OFFSET + 2 | E | R | | | | | | | CHARACTER2 | | | | | | | |
| OFFSET + 4 | E | R | | | | | | | CHARACTER3 | | | | | | | |
| | | | | | | | | ⋮ | | | | | | | | |
| OFFSET + E | E | R | | | | | | | CHARACTER8 | | | | | | | |
| OFFSET + 10 | 1 | 1 | | | | | | | RCCM | | | | | | | |
| OFFSET + 12 | | | | | | | | | RCCR | | | | | | | |

E—End of Table

In tables with eight control characters, E is always zero.

0 = This entry is valid. The lower 8 bits are checked against the incoming character.
1 = This entry is invalid and must be the last entry in the control characters table.

R—Reject Character

0 = The character is not rejected, but is written into the receive buffer. The buffer is then closed and a new receive buffer is used if there is more data in the message. A maskable (I-bit in the Rx BD) interrupt is generated.
1 = If this character is recognized, it is not written to the receive buffer. Instead, it is written to the RCCR and a maskable interrupt is generated. The current buffer is not closed when a control character is received with R set.

CHARACTER1–8—Control Character Values

These fields define control characters that should be compared to the incoming character. For less than 8-bit characters, the MSB should be zero.

RCCM—Received Control Character Mask

The value in this register is used to mask the comparison of CHARACTER1–8. The lower eight bits of RCCM correspond to the lower eight bits of CHARACTER1–8 and are decoded as follows:

0 = Mask this bit in the comparison of the incoming character and CHARACTER1–8.
1 = The address comparison on this bit proceeds normally; no masking occurs.

Bits 0 and 1 of RCCM must be set or erratic operation can occur during the control character recognition process.

RCCR—Received Control Character Register

Upon a control character match for which the reject bit is set, the UART writes the control character into the RCCR and generates a maskable interrupt. The CPU core must process the interrupt and read the RCCR before a second control character arrives. Failure to do so results in the UART overwriting the first control character.

**16.14.16.9 WAKE-UP TIMER (RECEIVER).** By issuing the ENTER HUNT MODE command, the user can temporarily disable the UART receiver and remains inactive until an idle or address character is recognized (depending on the setting of the UM bits).

If the UART is still in the process of receiving a message that the user has already decided to discard, the user can abort it's reception by issuing the ENTER HUNT MODE command. The UART receiver is reenabled when the message is finished by detecting the idle line (one character of idle) or by the address bit of the next message, depending on the UM bits.

When the receiver is in sleep mode and a break sequence is received, the receiver increments the BRKEC counter and generates the BRK interrupt if it is enabled.

**16.14.16.10 BREAK SUPPORT (RECEIVER).** The UART offers very flexible break support for the receiver. Transmission of out-of-sequence characters is also supported by the UART and is normally used for the transmission of flow control characters like XON or XOFF. This procedure is performed using the TOSEQ entry in the UART parameter RAM.

The UART polls TOSEQ whenever the transmitter is enabled for UART operation. This includes during UART freeze operation or UART buffer transmission, and when no buffer is ready for transmission. The TOSEQ character is transmitted at a higher priority than the other characters (if any) in the transmit buffer, but does not preempt characters already in the transmit FIFO. This means that the XON or XOFF character may not be transmitted for eight character times (SCC1) or four character times (SCC2). To reduce this latency, the TFL bit in the GSMR should be set to decrease the FIFO size to one character prior to enabling the SCC transmitter.

TOSEQ

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8DSR | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | REA | I | CT | 0 | 0 | A | CHARSEND | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | SCC BASE + 4E | | | | | | | | | | | | | | | |

Bits 0–1—Don't Care. May be written with ones or zeros.

REA—Ready

This bit is set by the CPU core when the character is ready for transmission and remains 1 while the character is being transmitted. The CP clears this bit after transmission.

I—Interrupt

If set, the CPU core is interrupted when this character has been transmitted. The TX bit is set in the UART event register.

CT—Clear-to-Send Lost

This status bit indicates that the $\overline{\text{CTS}}$ signal was negated during transmission of this character. If this occurs, the CTS bit in the UART event register is also set. This bit operates only if the $\overline{\text{CTS}}$ line is monitored by the SCC, as determined by the DIAG bits.

**NOTE**

If the $\overline{\text{CTS}}$ signal was negated during transmission and the CP transmits this character in the middle of buffer transmission, the $\overline{\text{CTS}}$ signal could actually have been negated either during this character's transmission or during a buffer character's transmission. In this case, the CP sets the CT bit both here and in the Tx BD status word.

Bits 5–6—Should be written with zeros.

A—Address

When working in a multidrop configuration, the user should include the address bit in this position.

CHARSEND

This value contains the character to be transmitted. Any 5-, 6-, 7-, or 8-bit character value can be transmitted in accordance with the UART configuration. The character should comprise the LSBs of CHARSEND. This value may be modified only while the REA bit is cleared.

**16.14.16.11 SEND BREAK (TRANSMITTER).** A break is an all-zeros character without a stop bit(s) and a break is sent by issuing the STOP TRANSMIT command. The UART completes transmission of any outstanding data, sends a programmable number of break characters according to the break count register, and then reverts to idle or sends data if the RESTART TRANSMIT command was given before completion. At the completion of the break code, the transmitter sends at least one high bit before transmitting any data to guarantee recognition of a valid start bit.

The break characters do not preempt characters already in the transmit FIFO. This means that the break character may not be transmitted for eight (SCC1) or four character times (SCC2). To reduce this latency, the TFL bit in the GSMR should be set to decrease the FIFO size to one character prior to enabling the SCC transmitter.

**16.14.16.12 SENDING A PREAMBLE (TRANSMITTER).** A preamble sequence gives the programmer a convenient way of ensuring that the line goes idle before starting a new message. The preamble sequence length is constructed of consecutive ones of one character length. If the preamble bit in a BD is set, the SCC sends a preamble sequence before transmitting that data buffer. For example, for 8 data bits, no parity, 1 stop bit, and 1 start bit, a preamble of 10 ones is sent before the first character in the buffer.

**16.14.16.13 FRACTIONAL STOP BITS (TRANSMITTER).** The asynchronous UART transmitter can be programmed to transmit fractional stop bits. Four bits in the SCC data synchronization register (DSR) are used to program the length of the last stop bit transmitted. These DSR bits can be modified at any time. If two stop bits are transmitted, only the second one is affected. Idle characters are always transmitted as full-length characters.

DSR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | 0 | FSB | | | | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| RESET | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | A0E (DSR1), A2E (DSR2) | | | | | | | | | | | | | | | |

In normal UART mode with 16× oversampling, the FSB bits (1–4) in the DSR are decoded as follows:

1111 = Last Transmitted Stop Bit 16/16 (the default value after reset)
1110 = Last Transmitted Stop Bit 15/16
1101 = Last Transmitted Stop Bit 14/16
1100 = Last Transmitted Stop Bit 13/16
1011 = Last Transmitted Stop Bit 12/16
1010 = Last Transmitted Stop Bit 11/16
1001 = Last Transmitted Stop Bit 10/16
1000 = Last Transmitted Stop Bit 9/16
0xxx = Invalid. Do not use.

When the UART is configured for 32× oversampling, the FSB bits (1–4) in the DSR are decoded as follows:

1111 = Last Transmitted Stop Bit 32/32 (the default value after reset)
1110 = Last Transmitted Stop Bit 31/32
1101 = Last Transmitted Stop Bit 30/32
1100 = Last Transmitted Stop Bit 29/32
1011 = Last Transmitted Stop Bit 28/32
1010 = Last Transmitted Stop Bit 27/32
1001 = Last Transmitted Stop Bit 26/32
1000 = Last Transmitted Stop Bit 25/32
0111 = Last Transmitted Stop Bit 24/32
0110 = Last Transmitted Stop Bit 23/32
0101 = Last Transmitted Stop Bit 22/32
0100 = Last Transmitted Stop Bit 21/32
0011 = Last Transmitted Stop Bit 20/32
0010 = Last Transmitted Stop Bit 19/32
0001 = Last Transmitted Stop Bit 18/32
0000 = Last Transmitted Stop Bit 17/32

When the UART is configured for 8× oversampling, the FSB bits (1–4) in the DSR are decoded as follows:

1111 = Last Transmitted Stop Bit 8/8 (the default value after reset)
1110 = Last Transmitted Stop Bit 7/8
1101 = Last Transmitted Stop Bit 6/8
1100 = Last Transmitted Stop Bit 5/8
10xx = Invalid. Do not use.
01xx = Invalid. Do not use.
00xx = Invalid. Do not use.

The UART receiver can always receive fractional stop bits. The next character's start bit can begin at any time after the three middle samples of the stop bit have been taken.

**16.14.16.14 UART ERROR-HANDLING PROCEDURE.** The UART controller reports character reception and transmission error conditions via the channel BDs, the error counters, and the UART event register. The modem interface lines can be monitored by the port C pins.

#### 16.14.16.14.1 UART Transmission Error.

**CTS Lost During Character Transmission**

When this error occurs, the channel stops transmission after finishing transmission of the current character from the buffer. The channel then sets the CT bit in the Tx BD and generates the TX interrupt if it is not masked. The channel resumes transmission after the RESTART TRANSMIT command is issued and the $\overline{\text{CTS}}$ pin is asserted.

**NOTE**

The UART also offers an asynchronous flow control option using $\overline{\text{CTS}}$ that does not generate an error. Refer to the FLC bit in the PSMR description in **Section 16.14.16 UART Controller**.

#### 16.14.16.14.2 Reception Errors.

**Overrun Error**

Data is moved from the receive FIFO to the data buffer when the first byte is received into the FIFO. If a receiver FIFO overrun occurs, the channel writes the received character into the internal FIFO over the previously received character (the previous character is lost.) The channel writes the received character to the buffer, closes the buffer, sets the overrun error (OV) bit in the Rx BD, and generates the RX interrupt if it is enabled. In automatic multidrop mode, the receiver enters hunt mode immediately.

**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 the CD lost during character reception (CD) bit in the Rx BD, and generates the RX interrupt (if enabled). This error has the highest priority. The last character in the buffer is lost and other errors are not checked. In automatic multidrop mode, the receiver enters the hunt mode immediately.

**Parity Error**

When a parity error occurs, the channel writes the received character to the buffer, closes the buffer, sets the parity error (PR) bit in the Rx BD, and generates the RX interrupt (if enabled). The channel also increments the PAREC counter. In automatic multidrop mode, the receiver enters hunt mode immediately.

### Noise Error

Noise error is detected by the UART controller when the three samples taken on every bit are not identical. When this error occurs, the channel writes the received character to the buffer and proceeds normally, but increments the noise error (NOSEC).

**NOTE**

In the synchronous mode of the UART controller, this error cannot occur.

### Idle Sequence Receive

An idle is detected when one character consisting of all ones is received. When the UART is receiving data into a receive buffer and an idle is received, the channel counts the number of consecutive idle characters received. If the count reaches the value programmed into MAX_IDL, the buffer is closed and an RX interrupt is generated. If no receive buffer is open, this event does not generate an interrupt or any status information. The internal idle counter (IDLC) is reset every time a character is received.

**NOTE**

To completely disable the idle sequence function, set the MAX_IDL value to zero.

### Framing Error

A framing error is detected by the UART controller when a character is received with no stop bit. All framing errors are reported by the UART controller, regardless of the UART mode. When this error occurs, the channel writes the received character to the buffer, closes the buffer, sets the framing error (FR) bit in the BD, and generates the RX interrupt (if enabled). The channel also increments the FRMEC. When this error occurs, parity is not checked for this character. In automatic multidrop mode, the receiver enters the hunt mode immediately.

If the RZS bit is set in the UART mode register when the UART is in the synchronous mode (SYN is set), the receiver reports all framing errors, but continues reception with the assumption that the unexpected zero is really the start bit of the next character. If RZS is set, user software may not want to consider a reported UART framing error as a true UART framing error, unless two or more framing errors occur within a short period of time.

### Break Sequence

The UART offers very flexible break support for the receiver. When the first break sequence is received (one or more all-zero characters), the UART increments the BRKEC and issues the break start (BRKs) event in the UART event register, which can generate an interrupt (if enabled). The UART then measures the break length and, when the break sequence is complete, writes the length to the BRKLN register. After the first one is received, the UART also issues the break end (BRKe) event in the UART event register, which can generate an interrupt (if enabled). If the UART was in the process of receiving characters when the break was received, it also closes the receive buffer, sets the break sequence (BR) bit in the Rx BD, and writes the RX bit in the event register, which can generate an interrupt (if enabled).

If the RZS bit is set in the UART mode register when the UART is in the synchronous mode (SYN is set), then a break sequence is detected after only two successive break characters are received.

**16.14.16.15 UART MODE REGISTER.** Each PSMR is a 16-bit, memory-mapped, read/write register that controls SCC operation. When the SCC is configured as a UART, this register is called the UART mode register cleared at reset. Many of the PSMR bits can be modified on-the-fly (while the receiver and transmitter are enabled).

PSMR

| BITS | 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 | 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 | A08 (PSMR1), A28 (PSMR2) | | | | | | | | | | | | | | | |

FLC—Flow Control

0 = Normal operation. The GSMR and port C registers determine the mode of the $\overline{\text{CTS}}$ pin.

1 = Asynchronous flow control. When the $\overline{\text{CTS}}$ pin is negated, the transmitter stops transmitting at the end of the current character. If $\overline{\text{CTS}}$ is negated past the middle of the current character, the next full character can be sent and transmission is stopped. When $\overline{\text{CTS}}$ is asserted once more, transmission continues where it left off and no CTS lost error is reported. No characters except idles are transmitted while $\overline{\text{CTS}}$ is negated.

SL—Stop Length

The SL bit selects the number of stop bits transmitted by the UART. This bit may be modified on-the-fly. The receiver is always enabled for one stop bit unless the UART is in synchronous mode and the RZS bit is set. Fractional stop bits are configured in the DSR.

0 = One Stop Bit
1 = Two Stop Bits

CL—Character Length

The CL bits determine the number of data bits in the character, not including the optional parity or multidrop address bits. When a character of less than an 8-bits is used, the MSBs in memory are written as zeros and on transmission the MSBs in memory are a don't care. These bits can be modified on-the-fly.

| | |
|---|---|
| 00 = | 5 Data Bits |
| 01 = | 6 Data Bits |
| 10 = | 7 Data Bits |
| 11 = | 8 Data Bits |

UM—UART Mode

The UART mode bits select the protocol that is implemented over the ASYNC channel and these bits can be modified on-the-fly.

00 = Normal UART operation. Multidrop mode is disabled and an idle-line wake-up is selected. In the idle-line wake-up mode, the UART receiver is reenabled by receiving one character of all ones.

01 = Multidrop nonautomatic mode. In the multidrop mode, an additional address/data bit is transmitted with each character. The multidrop asynchronous modes are compatible with the MC68681 DUART, the MC68HC11 SCI, the DSP56000 SCI, and the Intel 8051 serial interface. The UART receiver is reenabled when the last data bit received in the character (the address bit) is a one. This means that the received character is an address that has to be processed by all inactive processors. The UART receives the address character and writes it to a new buffer. The CPU core then compares the written address with its own address to decide whether to ignore or process the following characters.

10 = Reserved.

11 = Multidrop automatic mode. In this mode, the CP automatically checks the address of the incoming address character using the UADDR1 and UADDR2 parameter RAM values and automatically accepts or discards the data that follows the address.

FRZ—Freeze Transmission

This bit allows the user to halt the UART transmitter and continue transmission from the same point at a later time.

0 = Normal operation. If the UART was previously frozen, the UART resumes transmission from the next character in the same buffer that was frozen.

1 = The UART completes transmission of any data already transferred to the UART FIFO (the number of characters depends on the TFL bit in the GSMR) and then freezes (stops transmitting data). After this bit is reset, transmission proceeds from the next character.

RZS—Receive Zero Stop Bits

The RZS bit configures the UART receiver to receive data without stop bits. This configuration is useful in V.14 applications where UART data is supplied synchronously and all stop bits of a particular character can be omitted for the purpose of cross-network rate adaptation. RZS should only be set if the SYN bit is also set.

0 = The receiver operates normally with at least one stop bit required between characters. A framing error is issued upon a missing stop bit and a break status is set if a character with all-zero data bits is received with a zero stop bit.

1 = The receiver continues reception if a missing stop bit is detected and if the stop bit is a zero, the next bit is considered the first data bit of the next character. A framing error is issued if a stop bit is missing, but a break status is only reported after back-to-back reception of two break characters without stop bits.

SYN—Synchronous Mode

0 = Normal asynchronous operation. Notice that the user normally programs the TENC and RENC bits in the GSMR to NRZ and must select either 8×, 16×, or 32× in the RDCR and TDCR bits of the GSMR (16× is the recommended value for most applications).

1 = Synchronous UART using 1× clock. Notice that the user normally programs the TENC and RENC bits in the GSMR to NRZ and must select the RDCR and TDCR bits in the GSMR to be 1× mode. A 1 bit is transferred with each clock and is synchronous to the clock. As with the other modes, the clock can be provided internally or externally. This mode is sometimes referred to as isochronous operation of a UART channel.

DRT—Disable Receiver While Transmitting

0 = Normal operation.

1 = While data is being transmitted by the SCC, the receiver is disabled and gated by the internal RTS signal. This is useful if the UART is configured onto a multidrop line and the user does not want to receive their own transmission.

**NOTE**

The user should set the preamble bit in the transmit buffer descriptor if the MPC821 is being used in multidrop UART mode.

Bit 10—Reserved

PEN—Parity Enable

0 = No Parity.

1 = Parity is enabled and determined by the parity mode bits.

RPM—Receiver Parity Mode

The RPM bits select the type of parity check to be performed by the receiver and can be modified on-the-fly.

00 = Odd Parity.
01 = Low Parity (always check for a zero in the parity bit position).
10 = Even Parity.
11 = High Parity (always check for a 1 in the parity bit position).

When odd parity is selected, the transmitter counts the number of ones in the data word. If the total number of ones is not an odd number, the parity bit is set to 1 and produces an odd number. If the receiver counts an even number of ones, an error in transmission has occurred. In the same manner, for even parity, an even number must result from the calculation performed at both ends of the line. In high/low parity (sometimes called mark/space parity), if the parity bit is not high/low, a parity error is reported.

**NOTE**

The receive parity errors cannot be disabled, but can be ignored.

TPM—Transmitter Parity Mode

The TPM bits select the type of parity to be performed for the transmitter and can be modified on-the-fly.

00 = Odd Parity.
01 = Force Low Parity (always send a zero in the parity bit position).
10 = Even Parity.
11 = Force High Parity (always send a 1 in the parity bit position).

**16.14.16.16 UART RECEIVE BUFFER DESCRIPTOR.** The CP reports information concerning the received data on a per-buffer basis via receive buffer descriptors (Rx BDs).

The CP closes the current buffer, generates a maskable interrupt, and starts to receive data into the next buffer after one of the following events occur:

1. A user-defined control character is received (when the reject bit = 0 in the control character table entry).
2. An error during message processing is detected.
3. A full receive buffer is detected.
4. A MAX_IDL number of consecutive idle characters is received.
5. The ENTER HUNT MODE command is issued.
6. The CLOSE Rx BD command is issued.
7. An address character is received while working in multidrop mode. The address character is written to the next buffer for a software comparison.

An example of the SCC UART Rx BD process is illustrated in Figure 16-79.

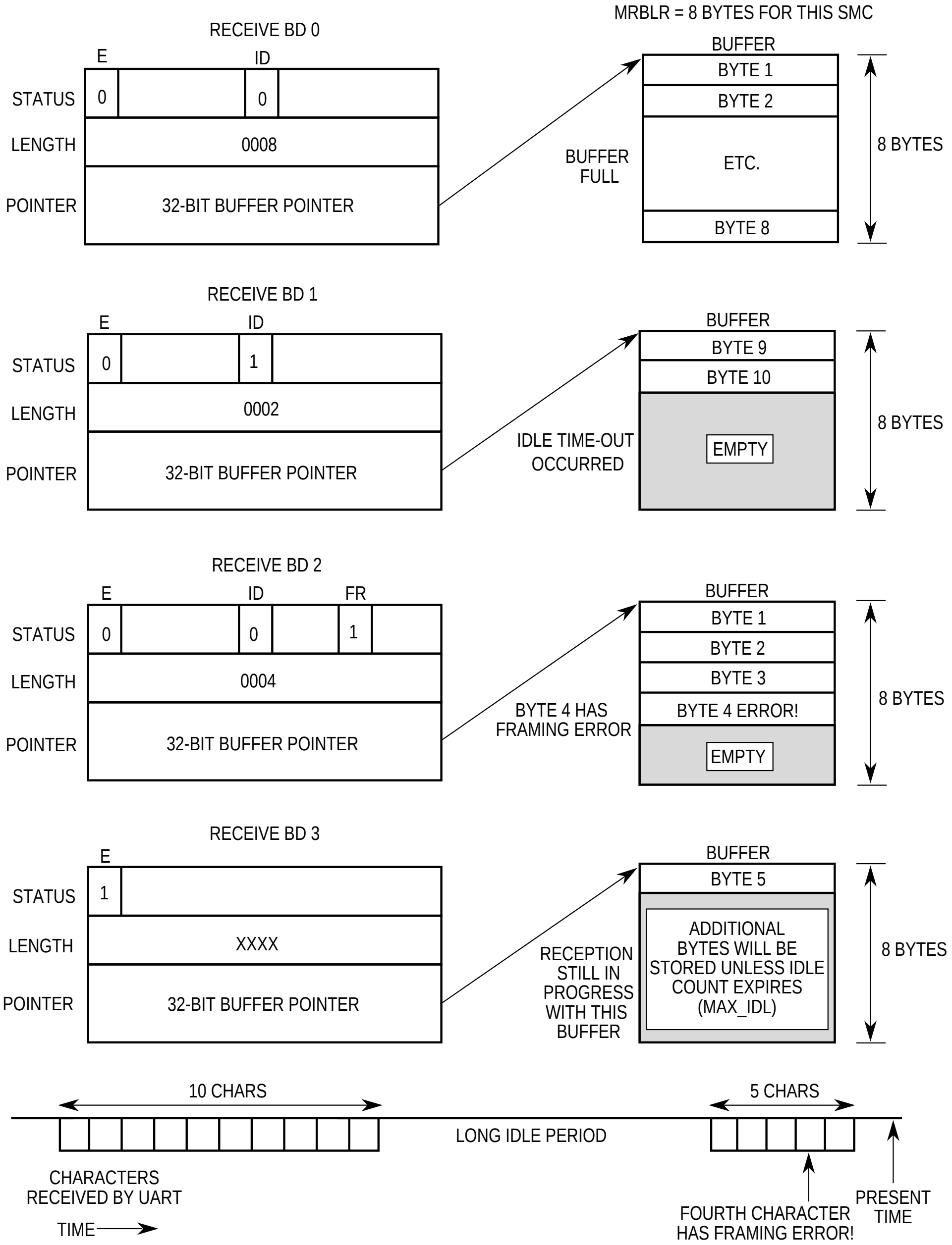


**Figure 16-79. SCC UART Rx BD Example**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **E** | RES | **W** | **I** | C | **A** | **CM** | ID | AM | RES | BR | FR | PR | RES | OV | CD |
| OFFSET + 2 | DATA LENGTH | | | | | | | | | | | | | | | |
| OFFSET + 4 | RX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

E—Empty

0 = The data buffer associated with this Rx BD has been filled with received data or data reception has been aborted due to an error condition. The CPU core is free to examine or write to any fields of this Rx BD. The CP does not use this BD again as long as the E-bit is zero.

1 = The data buffer associated with this BD is empty or reception is currently in progress. This Rx BD and it's associated receive buffer are owned by the CP. Once the E-bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Rx BD table.

1 = This is the last BD in the Rx BD table. After this buffer has been used, the CP receives incoming data into the first BD that RBASE points to in the table. The number of Rx BDs in this table are programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is filled.

1 = The RX bit in the UART event register is set when this buffer is completely filled by the CP, indicating the need for the CPU core to process the buffer. The RX bit can cause an interrupt if it is enabled.

C—Control Character

0 = This buffer does not contain a control character.

1 = This buffer contains a control character. The last byte in the buffer is one of the user-defined control characters.

A—Address

0 = The buffer only contains data.

1 = When working in nonautomatic multidrop mode, this bit indicates that the first byte of this buffer contains an address byte. The address comparison should be implemented in the software. In automatic multidrop mode, this bit indicates that the BD contains a message received immediately after an address is recognized in UADDR1 or UADDR2. This address is not written into the receive buffer.

CM—Continuous Mode

0 = Normal operation.
1 = The E-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically overwritten the next time CP accesses this BD. However, the E-bit is cleared if an error occurs during reception, regardless of the CM bit.

ID—Buffer Closed on Reception of Idles

The buffer is closed due to the reception of the programmable number of consecutive idle sequences (defined in MAX_IDL).

AM—Address Match

This bit only has meaning if the address bit is set and the automatic multidrop mode was selected in the UM bits. Following an address match, this bit defines which address character matched the user-defined address character, thus enabling the UART to receive data.

0 = The address matched the value in UADDR2.
1 = The address matched the value in UADDR1.

Bit 9—Reserved

BR—Break Received

A break sequence is received while receiving data into this buffer.

FR—Framing Error

A character with a framing error is received and located in the last byte of this buffer. A framing error is a character without a stop bit. A new receive buffer is used for further data reception.

PR—Parity Error

A character with a parity error is received and located in the last byte of this buffer. A new receive buffer is used for further data reception.

Bit 13—Reserved

OV—Overrun

A receiver overrun occurs during message reception.

CD—Carrier Detect Lost

The carrier detect signal is negated during message reception.

Data Length

Data length is the number of octets the CP writes into this BD data buffer. It is written by the CP once the BD is closed.

### NOTE

The actual amount of memory allocated for this buffer should be greater than or equal to the contents of the MRBLR.

Rx Data Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, can be even or odd. The buffer can reside in internal or external memory.

**16.14.16.17 UART TRANSMIT BUFFER DESCRIPTOR.** Data is presented to the CP for transmission on an SCC channel by arranging it in buffers referenced by the channel’s UART transmit buffer descriptor (Tx BD) table. The CP confirms transmission or indicates error conditions via the BDs to inform the processor that the buffers have been serviced.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **R** | RES | **W** | **I** | **CR** | **A** | **CM** | P | NS | RESERVED | | | | | | CT |
| **OFFSET + 2** | **DATA LENGTH** | | | | | | | | | | | | | | | |
| **OFFSET + 4** | **TX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or it’s associated data buffer. The CP clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which is prepared for transmission by the user, has not been transmitted yet or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Tx BD table.

1 = This is the last BD in the Tx BD table. After this buffer has been used, the CP will transmit data from the first BD that TBASE points to in the table. The number of Tx BDs in this table are programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = The TX bit in the UART event register is set when this buffer is serviced by the CP, which can cause an interrupt.

CR—Clear-to-Send Report

This bit allows a choice of either no delay between buffers transmitted in UART mode, a more accurate CTS lost error reporting, and three bits of idle between buffers.

0 = The buffer following this buffer is transmitted with no delay (assuming it is ready), but the CT bit may not be set in the correct Tx BD or may not be set at all in a CTS lost condition. Asynchronous flow control, however, continues to function normally.
1 = Normal CTS lost (CT bit) error reporting and three bits of idle occur between back-to-back buffers.

A—Address

This bit is only valid in multidrop mode (either automatic or nonautomatic).

0 = This buffer only contains data.
1 = Set by the CPU core, this bit indicates that this buffer contains address character(s). All the buffer data is transmitted as address characters.

CM—Continuous Mode

0 = Normal operation.
1 = The R-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this BD. However, the R-bit is cleared if an error occurs during transmission, regardless of the CM bit.

P—Preamble

0 = No preamble sequence is sent.
1 = The UART sends one character consisting of all ones before sending the data so that the other end can detect an idle line before the data. If this bit is set and the data length of this BD is zero, only a preamble is sent.

NS—No Stop Bit Transmitted

0 = Normal operation. Stop bits are sent with all characters in this buffer.
1 = The data in this buffer is sent without stop bits if the SYNC mode is selected by setting the SYN bit in the PSMR. If ASYNC is selected, the stop bit is SHAVED according to the value of the DSR.

Bits 9–14—Reserved

The following bit is written by the CP after it finishes transmitting the associated data buffer.

CT—CTS Lost

0 = The $\overline{\text{CTS}}$ signal remains asserted during transmission.
1 = The $\overline{\text{CTS}}$ signal is negated during transmission.

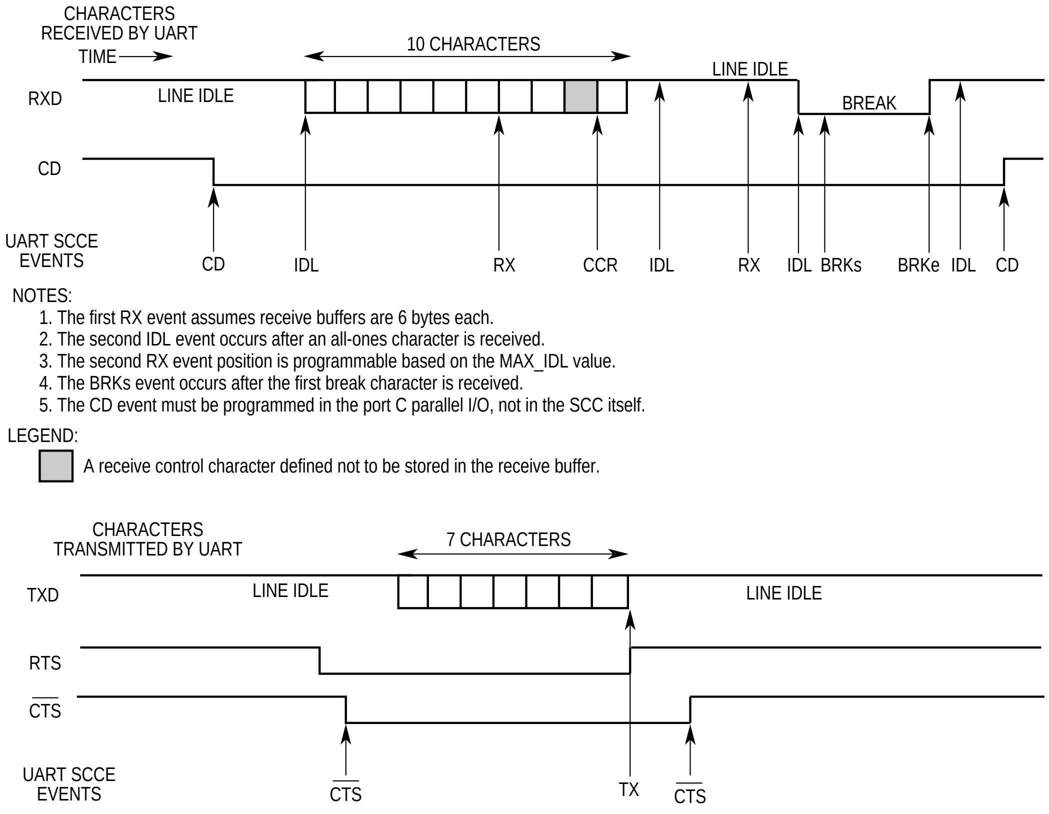
Data Length

The data length is the number of octets that the CP should transmit from this BD data buffer and it is never modified by the CP. Normally, this value should be greater than zero. The data length can be equal to zero with the P-bit set and only a preamble is sent.

Tx Data Buffer Pointer

The transmit buffer pointer, which always points to the first location of the associated data buffer, can be even or odd. The buffer can reside in internal or external memory.

**16.14.16.18 UART EVENT REGISTER.** The SCCE is called the UART event register when the SCC is operating as a UART. It is a 16-bit register used to report events recognized by the UART channel and to generate interrupts. On recognition of an event, the UART controller sets the corresponding bit in the UART event register. Interrupts generated by this register can be masked in the UART mask register. An example of interrupts that can be generated by the UART is illustrated in Figure 16-80.



NOTES:
1. The first RX event assumes receive buffers are 6 bytes each.
2. The second IDL event occurs after an all-ones character is received.
3. The second RX event position is programmable based on the MAX_IDL value.
4. The BRKs event occurs after the first break character is received.
5. The CD event must be programmed in the port C parallel I/O, not in the SCC itself.

LEGEND:

▨ A receive control character defined not to be stored in the receive buffer.

NOTES:
1. TX event assumes all seven characters were put into a single buffer and CR = 1 in the Tx BD.
2. The CTS event must be programmed in the port C parallel I/O, not in the SCC itself.

**Figure 16-80. UART Interrupt Events Example**

The UART event register is a memory-mapped register that can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset.

SCCE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | GLr | GLt | RES | AB | IDL | GRA | BRKe | BRKs | RES | CCR | BSY | TX | RX |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | A10 (SCCE1), A30 (SCCE2) | | | | | | | | | | | | | | | |

Bits 0–2—Reserved

These bits should be written with zeros.

GLr—Glitch on Rx

A clock glitch is detected by this SCC on the receive clock.

GLt—Glitch on Tx

A clock glitch is detected by this SCC on the transmit clock.

Bit 5—Reserved

This bit should be written as a zero.

AB—Auto Baud

An auto baud lock is detected. The CPU core should rewrite the baud rate generator with the precise divider value for the preferred baud rate. Refer to **Section 16.13 Baud Rate Generators** for more details.

IDL—Idle Sequence Status Changed

A change in the status of the serial line is detected on the UART channel. The real-time status of the line can be read in SCCS. Idle is entered when a character of all ones is received and it is exited when a single zero is received.

GRA—Graceful Stop Complete

A graceful stop, which is initiated by the GRACEFUL STOP TRANSMIT command, is now complete. This bit is set as soon as the transmitter has finished transmitting any buffer that is in progress when the command was issued. It is set immediately if no buffer is in progress when the command was issued.

BRKe—Break End

The end of a break sequence is detected. This indication is set no sooner than after an idle bit is received following a break sequence.

BRKs—Break Start

A break character is received and this is the first break of a break sequence. The user does not receive multiple BRKs events if a long break sequence is received.

Bit 11—Reserved

This bit should be written as a zero.

CCR—Control Character Received

A control character is received (with reject (R) character = 1) and stored in the receive control character register (RCCR).

BSY—Busy Condition

A character is received and discarded due to a lack of buffers. If the multidrop mode is selected, the receiver automatically enters hunt mode. Otherwise, reception continues as soon as an empty buffer is provided. The latest that an Rx BD can be made empty (have it's E-bit set) and still guarantee avoiding the busy condition is the middle of the stop bit of the first character to be stored in that buffer.

TX—Tx Buffer

A buffer is transmitted over the UART channel. If CR = 1 in the Tx BD, this bit is set no sooner than when the last stop bit of the last character in the buffer is first transmitted. If CR = 0, this bit is set after the last character is written to the transmit FIFO.

RX—Rx Buffer

A buffer is received over the UART channel. This event occurs no sooner than the middle of the first stop bit of the character that caused the buffer to be closed.

**16.14.16.19 UART MASK REGISTER.** The SCCM is referred to as the UART mask register when the SCC is operating as a UART. It is a 16-bit read/write register with the same bit formats as the UART event register. If a bit in the UART mask register is a 1, the corresponding interrupt in the event register is enabled and if it is zero, the corresponding interrupt in the event register is masked. This register is cleared at reset.

**16.14.16.20 SCC STATUS REGISTER.** The SCCS is an 8-bit read-only register that allows the user to monitor real-time status conditions on the RXD line. The real-time status of the $\overline{\text{CTS}}$ and CD pins are part of the port C parallel I/O.

SCCS REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | ID |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R | R |
| ADDR | A17 (SCCS1), A37 (SCCS2) | | | | | | | |

Bits 0–6—Reserved

ID—Idle Status

ID is set when the RXD pin has been a logic one for at least a full character time.

0 = The line is not currently idle.
1 = The line is currently idle.

**16.14.16.21 SCC UART EXAMPLE.** The following list is an initialization sequence for 9,600 baud, 8 data bits, no parity, and stop bit of an SCC UART operation assuming a 25 MHz system frequency. BRG1 and SCC2 are used. The UART is configured with the RTS2, $\overline{\text{CTS2}}$, and CD2 pins active. In addition, the $\overline{\text{CTS2}}$ pin is used as an automatic flow control signal.

1. Configure the port A pins to enable the TXD2 and RXD2 pins. Write PAPAR bits 13 and 12 with ones and then write the PADIR and PAODR bits 13 and 12 with zeros.
2. Configure the port C pins to enable RTS2, $\overline{\text{CTS2}}$, and CD2. Write PCPAR bit 14 with one and bits 9 and 8 with zeros, PCDIR bits 14, 9, and 8 with zeros, and PCSO bits 9 and 4 with ones.
3. Configure BRG1. Write BRGC1 with $010144. The DIV16 bit is not used and the divider is 162 (decimal). The resulting BRG1 clock is 16× the preferred bit rate of the UART.
4. Connect the BRG1 clock to SCC2 using the SI. Write the R2CS bits in the SICR to 000and the T2CS bits in the SICR to 000.
5. Write to the SDCR to initialize the SDMA Configuration Register.
6. Connect the SCC2 to the NMSI (its own set of pins). Clear the SC2 bit in the SICR.
7. Write RBASE and TBASE in the SCC parameter RAM to point to the Rx BD and Tx BD in the dual-port RAM. Assuming one Rx BD at the beginning of dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.
8. Write $0041 to the CPCR to execute the INIT RX and TX PARAMS command for SCC2.
9. Write RFCR with $15 and TFCR with $15 for normal operation.

10. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = $0010.
11. Write MAX_IDL with $0000 in the SCC UART-specific parameter RAM to disable the MAX_IDL functionality for this example.
12. Set BRKCR to $0001, so that if a STOP TRANSMIT command is issued, one break character is sent.
13. Clear PAREC, FRMEC, NOSEC, and BRKEC in the SCC UART-specific parameter RAM for clarity.
14. Clear UADDR1 and UADDR2. They are not used.
15. Clear TOSEQ. It is not used.
16. Write CHARACTER1–8 with $8000. They are not used.
17. Write RCCM with $C0FF. It is not used.
18. Initialize the Rx BD. Assume the Rx data buffer is at $00001000 in main memory. Write $B000 to Rx_BD_Status, $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
19. Initialize the Tx BD. Assume the Tx data buffer is at $00002000 in main memory and contains five 8-bit characters. Write $B000 to Tx_BD_Status, $0010 to Tx_BD_Length, and $00002000 to Tx_BD_Pointer.
20. Write $FFFF to the SCCE register to clear any previous events.
21. Write $0003 to the SCCM register to enable the TX and RX interrupts.
22. Write $20000000 to the CIMR to allow SCC2 to generate a system interrupt. The CICR should also be initialized.
23. Write $00000020 to the GSMR_H2 register to configure a small receive FIFO width.
24. Write $00028004 to the GSMR_L2 register to configure 16× sampling for transmit and receive, the $\overline{\text{CTS}}$ and CD pins to automatically control transmission and reception (DIG bits) and the UART mode. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled yet.
25. Set the PSMR2 register to $B000 to configure automatic flow control using the $\overline{\text{CTS}}$ pin, 8-bit characters, no parity, 1 stop bit, and asynchronous UART operation.
26. Write $00028034 to the GSMR_L2 register to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

**NOTE**

After 16 bytes are transmitted, the Tx BD is closed. Additionally, the receive buffer is closed after 16 bytes are received. Any additional receive data beyond 16 bytes causes a busy (out-of-buffers) condition since only one Rx BD is prepared.

**16.14.16.22 S-RECORDS PROGRAMMING EXAMPLE.** The following paragraphs contain an example of a downloading application that uses an SCC channel as a UART controller. The application performs S-record downloads and uploads between a host computer and an intelligent peripheral through a serial asynchronous line. The S-records are strings of ASCII characters that begin with ‘S’ and end in an end-of-line character. This characteristic is used to impose a message structure on the communication between the devices. Notice that each device can also transmit XON and XOFF characters for flow control, which do not form part of the program being uploaded or downloaded.

The UART mode register should be set as required, with the FRZ bit cleared and the ENT and ENR bits set. Receive buffers should be linked to the receive buffer table with the interrupt (I) bit set. For simplicity, assume that the line is not multidrop (no addresses are transmitted) and that each S-record fits into a single data buffer. Three characters should first be entered into the UART control character table:

- Line Feed—Both the E and R bits should be cleared. When an end-of-line character is received, the current buffer is closed (the next BD taken by the CP) and made available to the CPU core for processing. This buffer contains an entire S record that the processor can now check and copy to memory or disk as required.
- XOFF—E should be cleared and R should be set. Whenever the CPU core receives a control character received interrupt and the receive control character register contains XOFF, the software should immediately stop transmitting to the other station by setting the FRZ bit in the UART mode register. This prevents data from being lost by the other station when it runs out of receive buffers.
- XON—XON should be received after XOFF. E should be cleared and R should be set. The FRZ bit on the transmitter should now be cleared. The CP automatically resumes transmission of the serial line at the point at which it was previously stopped. Like XOFF, the XON character is not stored in the receive buffer.

To receive the S-records, the CPU core must only wait for the RX interrupt, indicating the reception of a complete S-record buffer. Transmission requires assembling S-records into data buffers and linking them to the transmit buffer table (transmission may be temporarily halted by the reception of an XOFF character). This scheme minimizes the number of interrupts received by the CPU core (one per S-record) and relieves it from the task of continually scanning for control characters.

## 16.14.17 HDLC Controller

Layer 2 of the seven-layer OSI model is the data link layer and one of the most common protocols in this layer is HDLC. In fact, many other common layer 2 protocols are based heavily on HDLC, particularly the framing structure of HDLC—SDLC, SS#7, AppleTalk, LAPB, and LAPD. The framing structure of HDLC is illustrated in Figure 16-81.

HDLC uses a zero insertion/deletion process (commonly known as bit-stuffing) to ensure that the bit pattern of the delimiter flag does not occur in the fields between flags. The HDLC frame is synchronous and therefore relies on the physical layer to provide a method of clocking and synchronizing the transmitter/receiver.

Since the layer 2 frame can be transmitted over a point-to-point link, a broadcast network, or packet and circuit switched systems, an address field is needed to carry the frame's destination address. The length of this field is commonly 0, 8, or 16 bits, depending on the data link layer protocol. For instance, SDLC and LAPB use an 8-bit address and SS#7 has no address field at all because it is always used in point-to-point signaling links. LAPD further divides its 16-bit address into different fields to specify various access points within one piece of equipment. It also defines a broadcast address. Some HDLC-type protocols also allow for extended addressing beyond 16 bits.

The 8- or 16-bit control field provides a flow control number and defines the frame type (control or data). The exact use and structure of this field depends upon the protocol using the frame. Data is transmitted in the data field, which can vary in length depending upon the protocol using the frame. Layer 3 frames are carried in this data field. Error control is implemented by appending a cyclic redundancy check (CRC) to the frame, which in most protocols is 16-bits long but can be as long as 32-bits. In HDLC, the LSB of each octet is transmitted first and the MSB of the CRC is transmitted first.

When the MODE bits of the GSMR select the HDLC mode, that SCC functions as an HDLC controller. When an SCC in HDLC mode is used with a nonmultiplexed modem interface, the SCC outputs are connected directly to the external pins. Modem signals can be supported through the port C pins. The receive and transmit clocks can be supplied from either the bank of baud rate generators, by the DPLL, or externally. The HDLC controller can also be connected to one of the two TDM channels of the serial interface and used with the TSA. The HDLC controller consists of separate transmit and receive sections whose operations are asynchronous with the CPU core and can either be synchronous or asynchronous with respect to the other SCCs. The user can allocate up to 196 BDs for receive and transmit tasks so that many frames can be transmitted or received without host intervention.

**16.14.17.1 FEATURES.** The following is a list of the HDLC’s important features:

- Flexible data buffers with multiple buffers per frame
- Separate interrupts for frames and buffers (receive and transmit)
- Received frames threshold to reduce interrupt overhead
- May be used with the SCC DPLL
- Four address comparison registers with mask
- Maintenance of five 16-bit error counters
- Flag/abort/idle generation/detection
- Zero insertion/deletion
- 16- or 32-bit CRC-CCITT generation/checking
- Detection of nonoctet aligned frames
- Detection of frames that are too long
- Programmable flags (0–15) between successive frames
- Automatic retransmission in case of collision

**16.14.17.2 HDLC CHANNEL FRAME TRANSMISSION PROCESSING.** The HDLC transmitter is designed to work with almost no intervention from the CPU core. When the CPU core enables one of the transmitters, it starts transmitting flags or idles as programmed in the HDLC mode register. The HDLC controller polls the first BD in the transmit channel BD table. When there is a frame to transmit, the HDLC controller fetches the data from memory and starts transmitting the frame (after first transmitting the user-specified minimum number of flags between frames). When the end of the current BD has been reached and the last buffer in the frame bit is set, the CRC (if selected) and closing flag are appended. In HDLC, the LSB of each octet and the MSB of the CRC are transmitted first. A typical HDLC frame is illustrated below in Figure 16-81.

| OPENING FLAG | ADDRESS | CONTROL | INFORMATION (OPTIONAL) | CRC | CLOSING FLAG |
|---|---|---|---|---|---|
| 8 BITS | 16 BITS | 8 BITS | 8N BITS | 16 BITS | 8 BITS |

**Figure 16-81. HDLC Framing Structure**

Following the transmission of the closing flag, the HDLC controller writes the frame status bits into the BD and clears the R-bit. When the end of the current BD has been reached and the last bit is not set (working in multibuffer mode), only the R-bit is cleared. In either mode, an interrupt can be issued if the I-bit in the Tx BD is set. The HDLC controller then proceeds to the next Tx BD in the table. In this way, the user can be interrupted after each buffer, a specific buffer, or each frame.

To rearrange the transmit queue before the CP has completed transmission of all buffers, issue the STOP TRANSMIT command. This technique can be useful for transmitting expedited data before previously linked buffers or for error situations. When receiving the STOP TRANSMIT command, the HDLC controller aborts the current frame being transmitted and starts transmitting idles or flags. When the HDLC controller is given the RESTART TRANSMIT command, it resumes transmission. To insert a high-priority frame without aborting the current frame, the GRACEFUL STOP TRANSMIT command can be issued. A special interrupt (GRA) can be generated in the event register when the current frame is complete.

**16.14.17.3 HDLC CHANNEL FRAME RECEPTION PROCESSING.** The HDLC receiver is designed to work with almost no intervention from the CPU core and can perform address recognition, CRC checking, and maximum frame length checking. The received frame is available to the user for performing any HDLC-based protocol.

When the CPU core enables one of the receivers, the receiver waits for an opening flag character and when it detects the first byte of the frame, the HDLC controller compares the frame address against the user-programmable addresses. The user has four 16-bit address registers and an address mask available for address matching. The HDLC controller compares the received address field to the user-defined values after masking with the address mask. The HDLC controller can also detect broadcast (all ones) address frames, if one address register is written with all ones.

If a match is detected, the HDLC controller fetches the next BD and if it is empty, it starts transferring the incoming frame to the BD associated data buffer. When the data buffer has been filled, the HDLC controller clears the E-bit in the BD and generates an interrupt if the I-bit in the BD is set. If the incoming frame exceeds the length of the data buffer, the HDLC controller fetches the next BD in the table and if it is empty, continues transferring the rest of the frame to this BD associated data buffer.

During this process, the HDLC controller checks for a frame that is too long. When the frame ends, the CRC field is checked against the recalculated value and written to the data buffer. The data length written to the last BD in the HDLC frame is the length of the entire frame. This enables HDLC protocols that “lose” frames to correctly recognize the frame-too-long condition. The HDLC controller then sets the last buffer in frame bit, writes the frame status bits into the BD, and clears the E-bit. The HDLC controller next generates a maskable interrupt, indicating that a frame has been received and is in memory. The HDLC controller then waits for a new frame. Back-to-back frames can be received with only a single shared flag between frames.

The user can configure the HDLC controller not to interrupt the CPU core until a certain number of frames have been received. This is configured in the received frames threshold (RFTHR) location of the parameter RAM. The user can combine this function with a timer to implement a timeout if less than the threshold number of frames are received.

**16.14.17.4 HDLC MEMORY MAP.** When configured to operate in HDLC mode, the MPC821 overlays the structure listed in Table 16-23 with the HDLC-specific parameters described in Table 16-26 below.

**Table 16-26. HDLC-Specific Parameters**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SCC Base + 30 | RES | Word | Reserved |
| SCC Base + 34 | **C_MASK** | Word | CRC Constant |
| SCC Base + 38 | **C_PRES** | Word | CRC Preset |
| SCC Base + 3C | **DISFC** | Half-word | Discard Frame Counter |
| SCC Base + 3E | **CRCEC** | Half-word | CRC Error Counter |
| SCC Base + 40 | **ABTSC** | Half-word | Abort Sequence Counter |
| SCC Base + 42 | **NMARC** | Half-word | Nonmatching Address Rx Counter |
| SCC Base + 44 | **RETRC** | Half-word | Frame Transmission Counter |
| SCC Base + 46 | **MFLR** | Half-word | Max Frame Length Register |
| SCC Base + 48 | MAX_cnt | Half-word | Max_Length Counter |
| SCC Base + 4A | **RFTHR** | Half-word | Received Frames Threshold |
| SCC Base + 4C | RFCNT | Half-word | Received Frames Count |
| SCC Base + 4E | **HMASK** | Half-word | User-Defined Frame Address Mask |
| SCC Base + 50 | **HADDR1** | Half-word | User-Defined Frame Address |

Table 16-26. HDLC-Specific Parameters (Continued)

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SCC Base + 52 | **HADDR2** | Half-word | User-Defined Frame Address |
| SCC Base + 54 | **HADDR3** | Half-word | User-Defined Frame Address |
| SCC Base + 56 | **HADDR4** | Half-word | User-Defined Frame Address |
| SCC Base + 58 | TMP | Half-word | Temp Storage |
| SCC Base + 5A | TMP_MB | Half-word | Temp Storage |

NOTE: Items in bold must be initialized by the user.
SCC base = IMMR + 1C00 (SCC1) or 1D00 (SCC2).

- C_MASK—For the 16-bit CRC-CCITT, C_MASK should be initialized with $0000F0B8. For the 32-bit CRC-CCITT, C_MASK should be initialized with $DEBB20E3.
- C_PRES—For the 16-bit CRC-CCITT, C_PRES should be initialized with $0000FFFF. For the 32-bit CRC-CCITT, C_PRES should be initialized with $FFFFFFFF.
- DISFC, CRCEC, ABTSC, NMARC, and RETRC—These 16-bit (modulo $2^{16}$) counters are maintained by the CP. They may be initialized by the user while the channel is disabled. The counters are as follows:
  - — DISFC–Discarded Frame Counter (error-free frames, but no free buffers).
  - — CRCEC–CRC Error Counter (includes frames not addressed to the user or frames received in the BSY condition, but does not include overrun errors).
  - — ABTSC–Abort Sequence Counter.
  - — NMARC–Nonmatching Address Received Counter (error-free frames only).
  - — RETRC–Frame Retransmission Counter (due to collision).
- MFLR—The HDLC controller checks the length of an incoming HDLC frame against the user-defined value given in this 16-bit register. If this limit is exceeded, the remainder of the incoming HDLC frame is discarded and the LG (Rx frame too long) bit is set in the last BD belonging to that frame. The HDLC controller waits until the end of the frame and then reports the frame status and length in the last Rx BD.

  The MFLR is defined as all the in-frame bytes between the opening and closing flags (address, control, data, and CRC). MAX_cnt is a temporary down-counter used to track the frame length.
- HMASK, HADDR1, HADDR2, HADDR3, and HADDR4—Each HDLC controller has five 16-bit registers for address recognition: one mask register and four address registers. The HDLC controller reads the frame address from the HDLC receiver, checks it against the four address register values, and then masks the result with the user-defined mask register. A one in the mask register represents a bit position for which address comparison should occur and a zero represents a masked bit position. Upon an address match, the address and the data following it are written into the data buffers. When the addresses are not matched and the frame is error-free, the nonmatching address received counter (NMARC) is incremented.

**NOTE**

For 8-bit addresses, mask out (clear) the eight high-order bits in the HMASK register. The eight low-order bits of HMASK and HADDRx should contain the address byte that immediately follows the opening flag. For, example, to recognize a frame that begins $7E (Flag), $68, $AA, using 16-bit address recognition, HADDRx should contain $AA68 and HMASK should contain $FFFF. Refer to Figure 16-82 for details.

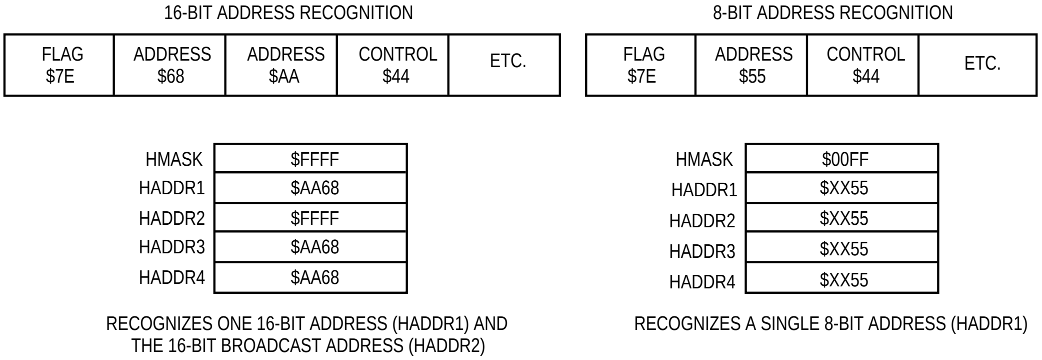


**Figure 16-82. HDLC Address Recognition Example**

- RFTHR—The received frame's threshold value is used to reduce the interrupt overhead that might otherwise occur when a series of short HDLC frames arrives, each causing an RXF interrupt. By setting the RFTHR value, the user limits the frequency of RXF interrupts which only occurs when the RFTHR value is reached. RFCNT is a down-counter used to implement this feature.

**NOTE**

The user should provide enough empty Rx BDs to receive the number of frames specified in RFTHR.

**16.14.17.5 PROGRAMMING MODEL.** The CPU core configures each SCC to operate in one of the protocols of the GSMR's MODE bits. The HDLC controller uses the same data structure as other modes. This data structure supports multibuffer operation and address comparisons. The receive errors (overrun, nonoctet aligned frame, CD lost, aborted frame, and CRC error) are reported through the Rx BD. The transmit errors (underrun and CTS lost) are reported through the Tx BD.

**16.14.17.6 COMMAND SET.** The following transmit and receive commands are issued to the CPCR.

#### 16.14.17.6.1 Transmit Commands.

**STOP TRANSMIT**

After the hardware or software is reset and the channel is enabled in the SCC mode register, the channel is in the transmit enable mode and starts polling the first BD in the table every 64 transmit clocks (immediately if the TOD bit in the TODR is set). The channel STOP TRANSMIT command disables the transmission of frames on the transmit channel. If this command is received by the HDLC controller during frame transmission, transmission is aborted after a maximum of 64 additional bits are transmitted and the transmit FIFO is flushed. The TBPTR is not advanced, no new BD is accessed, and no new frames are transmitted for this channel. The transmitter transmits an abort sequence consisting of 01111111 (if the command was given during frame transmission) and begins transmitting flags or idles, as indicated by the HDLC mode register.

**NOTE**

If the MFF bit in the PSMR is set, then it is possible for one or more small frames to be flushed from the transmit FIFO. The GRACEFUL STOP TRANSMIT command can be used to avoid this.

**GRACEFUL STOP TRANSMIT**

This command is used to stop transmission smoothly rather than abruptly, as performed by the regular STOP TRANSMIT command. It stops transmission after the current frame has finished transmitting or immediately if there is no frame being transmitted. The GRA bit in the SCCE is set once transmission has stopped. Then the HDLC transmit parameters (including BDs) can be modified. The TBPTR points to the next Tx BD in the table. Transmission begins once the R-bit of the next BD is set and the RESTART TRANSMIT command is issued.

**RESTART TRANSMIT**

This command enables the transmission of characters on the transmit channel. This command is expected by the HDLC controller after a STOP TRANSMIT command, after a STOP TRANSMIT command is issued and the channel in it's SCC mode register is disabled, after a GRACEFUL STOP TRANSMIT command, or after a transmitter error (underrun or CTS lost when no automatic frame retransmission is performed). The HDLC controller resumes transmission from the current TBPTR in the channel Tx BD table.

### INIT TX PARAMETERS

This command initializes all transmit parameters in this serial channel parameter RAM to their reset state. This command should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the transmit and receive parameters.

### 16.14.17.6.2 Receive Commands.

### ENTER HUNT MODE

After the hardware or software is reset and the channel is enabled in the SCC mode register, the channel is in the receive enable mode and uses the first BD in the table. The ENTER HUNT MODE command is generally used to force the HDLC receiver to abort reception of the current frame and enter the hunt mode. In the hunt mode, the HDLC controller continually scans the input datastream for the flag sequence. After receiving the command, the current receive buffer is closed and the CRC is reset. Further frame reception uses the next BD.

### CLOSE Rx BD

This command should not be used in the HDLC protocol.

### INIT RX PARAMETERS

This command initializes all the receive parameters in this serial channel parameter RAM to their reset state and should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the receive and transmit parameters.

**16.14.17.7 HDLC ERROR-HANDLING PROCEDURE.** The HDLC controller reports frame reception and transmission error conditions using the channel BDs, error counters, and HDLC event register.

### 16.14.17.7.1 Transmission Errors.

### Transmitter Underrun

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the underrun (U) bit in the BD, and generates the TXE interrupt if it is enabled. The channel resumes transmission after receiving the RESTART TRANSMIT command. The transmit FIFO size is 32 bytes on SCC1 and 16 bytes on SCC2.

### CTS Lost During Frame Transmission

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the CT bit in the BD, and generates the TXE interrupt if it is enabled. The channel resumes transmission after reception of the RESTART TRANSMIT command. If this error occurs on the first or second buffer of the frame and the RTE bit in the HDLC mode register is set, the channel retransmits the frame when the CTS line becomes active again. When working in an HDLC mode with collision possibility, to ensure the retransmission method functions properly, the first and second data buffers should contain more than 36 bytes of data (SCC1) and 20 bytes of data (SCC2) if multiple buffers per frame are used. The channel also increments the retransmission counter. This requirement does not apply to small frames consisting of a single buffer.

## 16.14.17.7.2 Reception Errors.

### Overrun Error

The HDLC controller maintains an internal FIFO for receiving data. The CP begins programming the SDMA channel (if the data buffer is in external memory) and updating the CRC when 8 or 32 bits (according to the RFW bit in the GSMR) are received in the FIFO. When a receive FIFO overrun occurs, the channel writes the received data byte to the internal FIFO over the previously received byte. The previous data byte and the frame status are lost. The channel closes the buffer with the overrun (OV) bit in the BD set and generates the RXF interrupt if it is enabled. The receiver then enters the hunt mode. Even if the overrun occurs during a frame whose address is not matched in the address recognition logic, an Rx BD with data length two is opened to report the overrun and the RXF interrupt is generated if it is enabled.

### CD Lost During Frame Reception

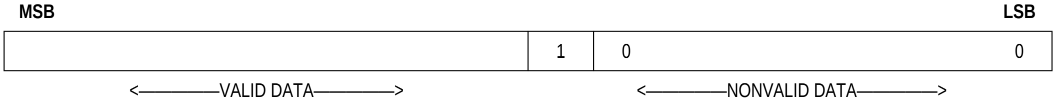
When this error occurs, the channel terminates frame reception, closes the buffer, sets the CD bit in the Rx BD, and generates the RXF interrupt if it is enabled. This error has the highest priority. The rest of the frame is lost and other errors are not checked in that frame. At this point, the receiver enters the hunt mode.

### Abort Sequence

An abort sequence is detected by the HDLC controller when seven or more consecutive ones are received. When this error occurs and the HDLC controller receives a frame, the channel closes the buffer by setting the AB bit in the Rx BD and generates the RXF interrupt (if enabled). The channel also increments the abort sequence counter. The CRC and nonoctet error status conditions are not checked on aborted frames. The receiver then enters hunt mode. When an abort is received, the user is given no indication that an HDLC controller is not currently receiving a frame.

### Nonoctet Aligned Frame

When this error occurs, the channel writes the received data to the data buffer, closes the buffer, sets the Rx nonoctet aligned frame (NO) bit in the Rx BD, and generates the RXF interrupt if it is enabled. The CRC error status should be disregarded on nonoctet frames. After a nonoctet aligned frame is received, the receiver enters hunt mode. An immediately back-to-back frame is still received. The nonoctet data may be derived from the last word in the data buffer as follows:



**NOTE**

If the data buffer swapping option is used (BO=0x in the RFCR), then the above diagram refers to the last byte of the data buffer, not the last word. In HDLC, the LSB of each octet is transmitted first and the MSB of the CRC is transmitted first.

### CRC Error

When this error occurs, the channel writes the received CRC to the data buffer, closes the buffer, sets the CR bit in the Rx BD, and generates the RXF interrupt if it is enabled. The channel also increments the CRC error counter. After receiving a frame with a CRC error, the receiver enters hunt mode. An immediately following back-to-back frame is still received. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

**16.14.17.8 HDLC MODE REGISTER.** Each HDLC mode register is a 16-bit, memory-mapped, read/write register that controls SCC operation. The term HDLC mode register refers to the PSMR of the SCC when that SCC is configured for HDLC. The HDLC mode register is cleared at reset.

PSMR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | NOF | | | | CRC | | RTE | RES | FSE | DRT | BUS | BRM | MFF | RESERVED | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | A08 (PSMR1), A28 (PSMR2) | | | | | | | | | | | | | | | |

NOF—Number of Flags

Minimum number of flags between or before frames (0 to 15 flags). If NOF = 0000, then no flags are inserted between the frames. Thus, the closing flag of one frame is immediately followed by the opening flag of the next frame in the case of back-to-back frames. These bits can be modified on-the-fly.

CRC—CRC Selection

00 = 16-Bit CCITT-CRC (HDLC). X16 + X12 + X5 + 1.
01 = Reserved.
10 = 32-Bit CCITT-CRC (Ethernet and HDLC). X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X1 +1.
11 = Reserved.

RTE—Retransmit Enable

0 = No retransmission.
1 = Automatic frame retransmission is enabled. This is particularly useful in the HDLC bus protocol and ISDN applications where multiple HDLC controllers can collide on a single channel. Notice that retransmission only occurs if the lost CTS occurs on the first or second buffer of the frame.

Bits 7, 13–15—Reserved

FSE—Flag Sharing Enable.

This bit is only valid if the RTSM bit is set in the GSMR. This bit can be modified on-the-fly.

0 = Normal operation.
1 = If NOF0–NOF3 = 0000, then a single shared flag is transmitted between back-to-back frames. Other values of NOF0–NOF3 are decremented by 1 when FSE is set. This is useful in signaling system #7 applications.

DRT—Disable Receiver While Transmitting

0 = Normal operation.
1 = While data is being transmitted by the SCC, the receiver is disabled, being gated by the internal $\overline{\text{RTS}}$ signal. This configuration is useful if the HDLC channel is configured onto a multidrop line and the user does not want to receive his own transmission.

BUS—HDLC Bus Mode

0 = Normal HDLC operation.
1 = HDLC bus operation selected. Refer to **Section 16.14.18 HDLC Bus Controller** for more details.

BRM—HDLC Bus RTS Mode

This bit is only valid if BUS = 1. Otherwise, it is ignored.

0 = Normal RTS operation during HDLC bus mode RTS is asserted on the first bit of the transmit frame and negated after the first collision bit is received.
1 = Special RTS operation during HDLC bus mode. RTS is delayed by one bit with respect to the normal case. This is useful when the HDLC bus protocol is simultaneously being run locally and transmitted over a long-distance transmission line. Data can be delayed by 1 bit before it is sent over the transmission line, thus RTS can be used to enable the transmission line buffers. The result is a clean signal level sent over the transmission line.

MFF—Multiple Frames in FIFO

0 = Normal operation. The transmit FIFO must never contain more than one HDLC frame. The CTS lost status is reported accurately on a per-frame basis. The receiver is not affected by this bit.
1 = The transmit FIFO can contain multiple frames, but lost CTS is not guaranteed to be reported on the exact buffer/frame it truly occurred on. This option, however, can improve the performance of HDLC transmissions in cases of small back-to-back frames or in cases where the user prefers to strongly limit the number of flags transmitted between frames. The receiver is not affected by this bit.

**16.14.17.9 HDLC RECEIVE BUFFER DESCRIPTOR.** The HDLC controller uses the Rx BD to report information about the received data for each buffer. An example of the Rx BD process is illustrated in Figure 16-83.

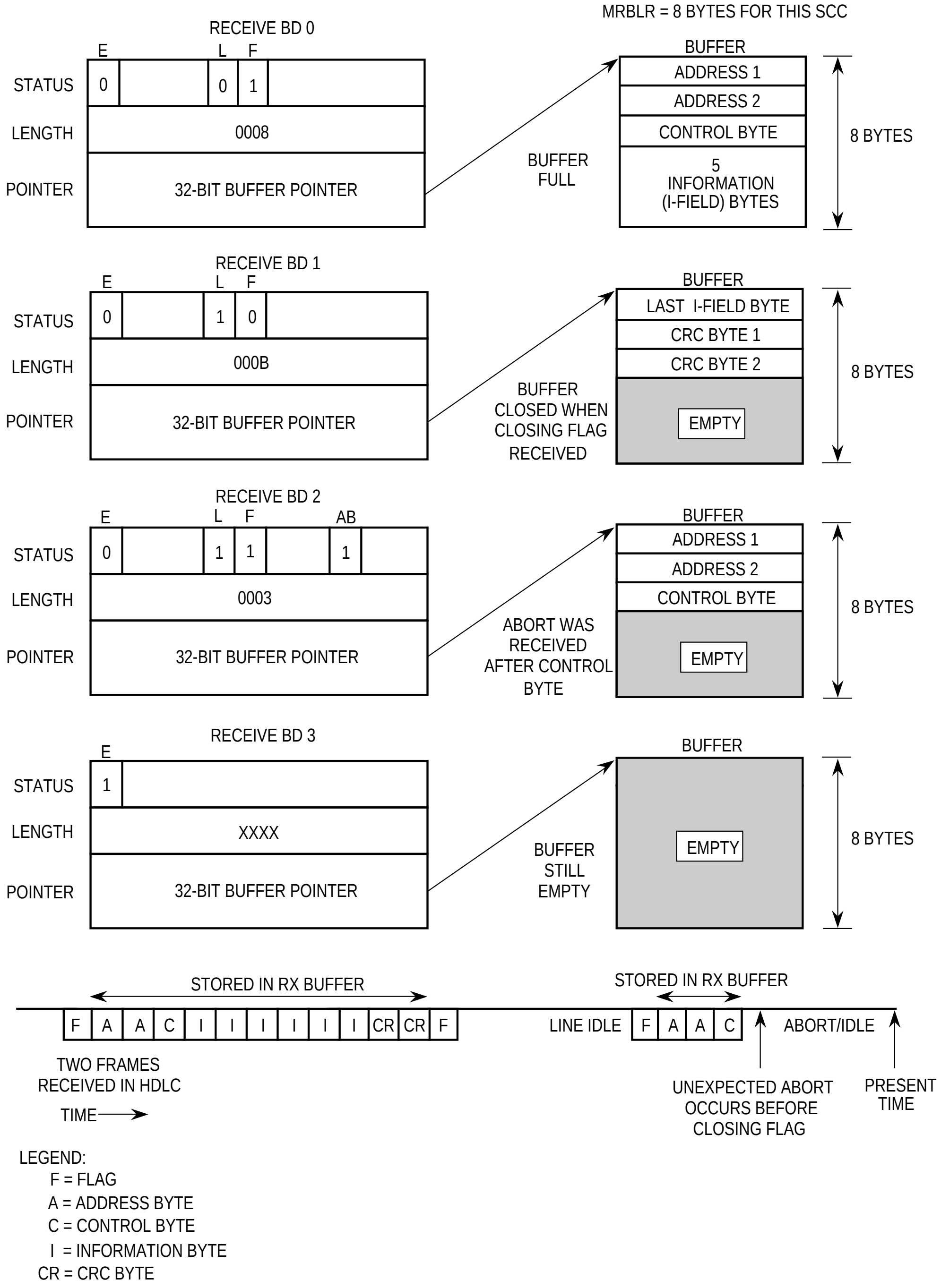


**Figure 16-83. HDLC Rx BD Example**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **E** | RES | **W** | **I** | L | F | **CM** | RES | DE | RES | LG | NO | AB | CR | OV | CD |
| **OFFSET + 2** | **DATA LENGTH** | | | | | | | | | | | | | | | |
| **OFFSET + 4** | **RX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

E—Empty

0 = The data buffer associated with this Rx BD has been filled with received data or data reception has been aborted because of an error condition. The CPU core is free to examine or write to any fields of this Rx BD. The CP does not use this BD as long as the E-bit is zero.

1 = The data buffer associated with this BD is empty or reception is currently in progress. This Rx BD and it’s associated receive buffer are owned by the CP. Once the E-bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last buffer descriptor in the Rx BD table.

1 = This is the last buffer descriptor in the Rx BD 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 Rx BDs in this table are programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = The RXB bit is not set after this buffer is used, but RXF operation remains unaffected.

1 = The RXB or RXF bit in the HDLC event register is set when the HDLC controller uses this buffer. These two bits can cause interrupts if they are enabled.

L—Last in Frame

This bit is set by the HDLC controller when this buffer is the last one in a frame. This implies the reception of a closing flag or reception of an error, in which case one or more of the CD, OV, AB, and LG bits are set. The HDLC controller writes the number of frame octets to the data length field.

0 = This buffer is not the last one in a frame.

1 = This buffer is the last one in a frame.

F—First in Frame

This bit is set by the HDLC controller when this buffer is the first in a frame.

0 = The buffer is not the first one in a frame.

1 = The buffer is the first one in a frame.

CM—Continuous Mode

0 = Normal operation.
1 = The E-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this BD. The E-bit is cleared if an error occurs during reception, regardless of the CM bit.

Bit 7—Reserved

DE—DPLL Error

This bit is set by the HDLC controller when a DPLL error occurs during the reception of this buffer. In decoding modes where a transition is promised every bit, the DE bit is set when a missing transition occurs.

Bit 9—Reserved

LG—Rx Frame Length Violation

A frame length greater than the maximum defined for this channel is recognized and only the maximum-allowed number of bytes (MFLR) is written to the data buffer. This event is not reported until the Rx BD is closed, the RXF bit is set, and the closing flag is received. The actual number of bytes received between flags is written to the data length field of this BD.

NO—Rx Nonoctet Aligned Frame

A frame that contains a number of bits approximately divisible by eight is received.

AB—Rx Abort Sequence

A minimum of seven consecutive ones are received during frame reception.

CR—Rx CRC Error

This frame contains a CRC error. The received CRC bytes are always written to the receive buffer.

OV—Overrun

A receiver overrun occurs during frame reception.

CD—Carrier Detect Lost

The carrier detect signal is negated during frame reception. This bit is only valid when working in the NMSI mode.

Data Length

Data length is the number of octets the CP writes into this BD data buffer. It is written by the CP once the BD is closed. When this BD is the last BD in the frame (L = 1), the data length contains the total number of frame octets, including 2 or 4 bytes for CRC. The actual amount of memory allocated for this buffer should be greater than or equal to the MRBLR.

Rx Data Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, resides in internal or external memory and must be divisible by four.

**16.14.17.10 HDLC TRANSMIT BUFFER DESCRIPTOR.** Data is presented to the HDLC controller for transmission on an SCC channel by arranging it in buffers referenced by the channel Tx BD table. The HDLC controller confirms transmission (or indicates error conditions) using the BDs to inform the CPU core that the buffers have been serviced.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **R** | RES | **W** | **I** | L | **TC** | **CM** | RES | RES | RES | RES | RES | RES | RES | UN | CT |
| OFFSET + 2 | **DATA LENGTH** | | | | | | | | | | | | | | | |
| OFFSET + 4 | **TX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or it's associated data buffer. The CP clears this bit after the buffer has been transmitted or an error condition is encountered.

1 = The data buffer, which has been prepared for transmission by the user, has not been transmitted or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Tx BD table.

1 = This is the last BD in the Tx BD table. After this buffer has been used, the CP transmit data from the first BD that TBASE points to in the table. The number of Tx BDs in this table are programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = Either TXB or TXE in the HDLC event register is set when this buffer is serviced by the HDLC controller. These bits can cause interrupts if they are enabled.

L—Last

0 = This is not the last buffer in the frame.

1 = This is the last buffer in the current frame.

TC—Tx CRC

This bit is valid only when the L-bit is set. Otherwise, it is ignored.

0 = Transmit the closing flag after the last data byte. This setting can be used to send a bad CRC after the data for testing purposes.

1 = Transmit the CRC sequence after the last data byte.

CM—Continuous Mode

0 = Normal operation.
1 = The R-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this BD. However, the R-bit is cleared if an error occurs during transmission, regardless of the CM bit.

P—Preamble

0 = No preamble sequence is sent.
1 = The UART sends one character of all ones before sending the data so that the other end detects an idle line before the data. If this bit is set and the data length of this BD is zero, only a preamble is sent.

NS—No Stop Bit Transmitted

0 = Normal operation. Stop bits are sent with all characters in this buffer.
1 = The data in this buffer is sent without stop bits if the SYNC mode is selected by setting the SYN bit in the PSMR. If ASYNC is selected, the stop bit is SHAVED according to the value of the DSR.

Bits 7–13—Reserved

The following status bits are written by the HDLC controller after it finishes transmitting the associated data buffer.

UN—Underrun

The HDLC controller encounters a transmitter underrun condition while transmitting the associated data buffer.

CT—CTS Lost

CTS in NMSI mode or layer 1 grant is lost in GCI mode during frame transmission. If data from more than one buffer is currently in the FIFO when this error occurs, this bit is set in the currently open Tx BD.

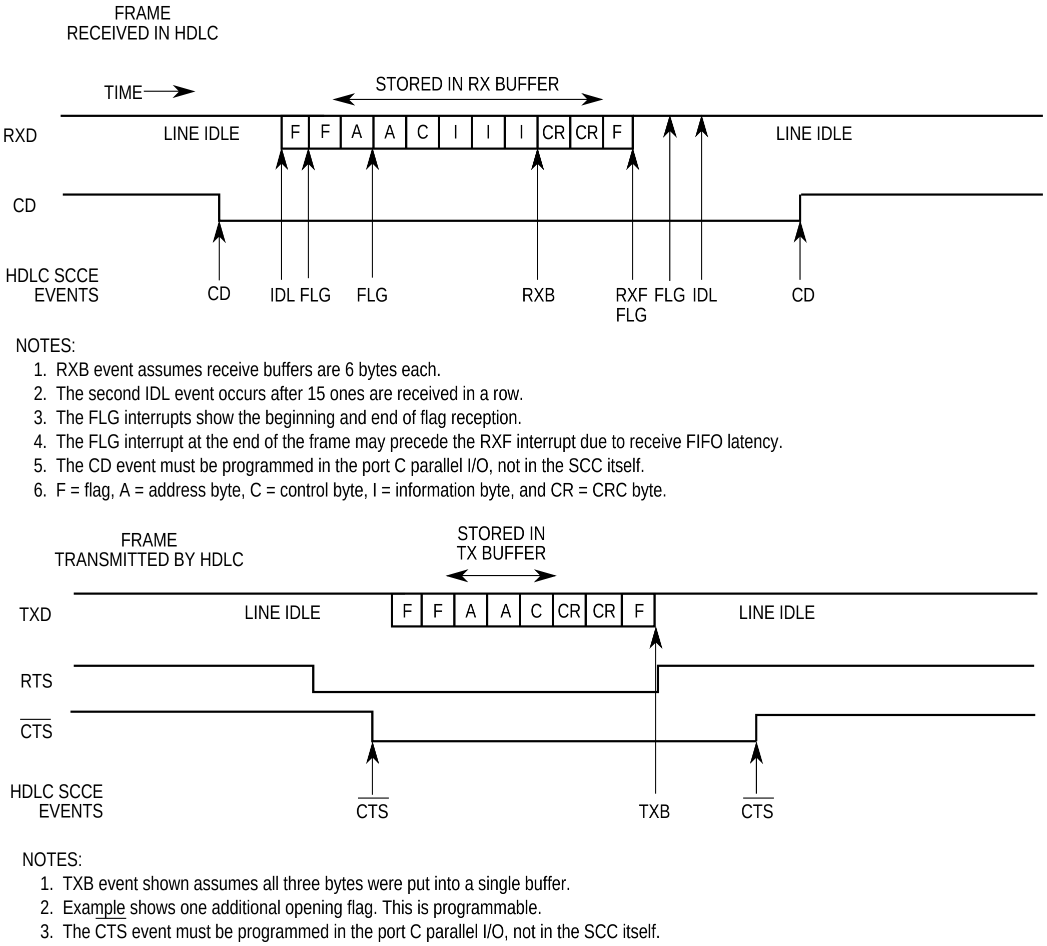
Data Length

The data length is the number of bytes the HDLC controller should transmit from this BD data buffer and it is never modified by the CP. The value of this field should be greater than zero.

Tx Data Buffer Pointer

The transmit buffer pointer, which contains the address of the associated data buffer, can either be even or odd. The buffer can reside in internal or external memory. This value is never modified by the CP.

**16.14.17.11 HDLC EVENT REGISTER.** The SCCE is called the HDLC event register when the SCC is operating as an HDLC controller. It is a 16-bit register used to report events recognized by the HDLC channel and to generate interrupts. On recognition of an event, the HDLC controller sets the corresponding bit in the HDLC event register. Interrupts generated by this register can be masked in the HDLC mask register. An example of interrupts that can be generated in the HDLC protocol is illustrated in Figure 16-84.



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 $\overline{CTS}$ event must be programmed in the port C parallel I/O, not in the SCC itself.

**Figure 16-84. HDLC Interrupt Event Example**

The HDLC event register is a memory-mapped register that can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset.

SCCE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | GLr | GLt | DCC | FLG | IDL | GRA | RES | RES | TXE | RXF | BSY | TXB | RXB |
| RESET | | | | | | | | | | | | | | | | |
| R/W | | | | | | | | | | | | | | | | |
| ADDR | A10 (SCCE1), A30 (SCCE2) | | | | | | | | | | | | | | | |

Bits 0–2—Reserved

These bits should be written with zeros.

GLr—Glitch on Rx

A clock glitch is detected by the SCC on the receive clock.

GLt—Glitch on Tx

A clock glitch is detected by the SCC on the transmit clock.

DCC—DPLL CS Changed

The carrier sense status generated by the DPLL has changed state. The real-time status can be found in SCCS. This is not the CD pin status that is reported in port C and is only valid when the DPLL is used.

FLG—Flag Status

The HDLC controller stops or starts receiving HDLC flags. The real-time status can be obtained in SCCS.

IDL—Idle Sequence Status Changed

A change in the status of the serial line is detected on the HDLC line. The real-time status of the line can be read in SCCS.

GRA—Graceful Stop Complete

A graceful stop, which was initiated by the GRACEFUL STOP TRANSMIT command, is now complete. This bit is set as soon as the transmitter finishes transmitting any frame that is in progress when the command was issued. It is set immediately if no frame is in progress when the command was issued.

Bits 9–10—Reserved

These bits should be written with zeros.

TXE—Tx Error

An error (CTS lost or underrun) occurs on the transmitter channel.

RXF—Rx Frame

A complete frame is received on the HDLC channel. This bit is set no sooner than two clocks after receipt of the last bit of the closing flag.

BSY—Busy Condition

A frame is received and discarded due to a lack of buffers.

TXB—Transmit Buffer

A buffer is transmitted on the HDLC channel. This bit is set no sooner than when the last bit of the closing flag begins it's transmission if the buffer is the last one in the frame. Otherwise, this bit is set after the last byte of the buffer is written to the transmit FIFO.

RXB—Receive Buffer

A buffer that is not a complete frame is received on the HDLC channel.

**16.14.17.12 HDLC MASK REGISTER.** The SCCM is referred to as the HDLC mask register when the SCC is operating as an HDLC controller. It is a 16-bit read/write register with the same bit formats as the HDLC event register. If a bit in the HDLC mask register is a 1, the corresponding interrupt in the event register is enabled. If the bit is zero, the corresponding interrupt in the event register is masked. This register is cleared upon reset.

**16.14.17.13 SCC STATUS REGISTER.** The SCCS is an 8-bit read-only register that allows the user to monitor real-time status conditions on the RXD line. The real-time status of the $\overline{\text{CTS}}$ and CD pins are part of the port C parallel I/O.

SCCS REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | FG | CS | ID |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | | RO | R | R |
| ADDR | A17 (SCCS1), A37 (SCCS2) | | | | | | | |

Bits 0–4—Reserved

FG—Flags

While FG is cleared, the most recently received 8 bits are examined every bit time to see if a flag is present. FG is set as soon as an HDLC flag ($7E) is received on the line. Once FG is set, it remains set at least 8 bit times, then the next eight received bits are examined. If another flag occurs, FG stays set for at least another eight bits. Otherwise, FG is cleared and the search begins again.

The examination of the line is made after the data has been decoded by the DPLL.

0 = HDLC flags are not currently being received.
1 = HDLC flags are currently being received.

CS—Carrier Sense (DPLL)

This bit shows the real-time carrier sense of the line as determined by the DPLL, if it is used.

0 = The DPLL does not sense a carrier.
1 = The DPLL senses a carrier.

ID—Idle Status

ID is set when the RXD pin is a logic one for 15 or more consecutive bit times; it is cleared after a single logic zero is received.

0 = The line is busy.
1 = The line is currently idle.

**16.14.17.14 SCC HDLC EXAMPLE #1.** The following list is an initialization sequence for an SCC HDLC channel assuming an external clock is provided. SCC2 is used. The HDLC controller is configured with the RTS2, $\overline{\mathrm{CTS2}}$, and CD2 pins active. The CLK3 pin is used for both the HDLC receiver and transmitter.

1. Configure the port A pins to enable the TXD2 and RXD2 pins. Write PAPAR bits 13 and 12 with ones and then write PADIR and PAODR bits 13 and 12 with zeros.
2. Configure the port C pins to enable RTS2, $\overline{\mathrm{CTS2}}$, and CD2. Write PCPAR bit 14 with one, and bits 9 and 8 with zeros, PCDIR bits 14, 9 and 8 with zeros, and PCSO bits 9 and 8 with ones.
3. Configure port A to enable the CLK3 pin. Write PAPAR bit 5 with a one and PADIR bit 5 with a zero.
4. Connect the CLK3 pin to SCC2 using the SI. Write the R2CS and T2CS bits in the SICR to 110.
5. Connect the SCC2 to the NMSI (its own set of pins) and clear the SC2 bit in the SICR.
6. Write $0001 to the SDCR to initialize the SDMA Configuration Register.
7. Write RBASE and TBASE in the SCC parameter RAM to point to the Rx BD and Tx BDs in the dual-port RAM. Assuming one Rx BD at the beginning of dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.
8. Program the CPCR to execute the INIT RX and TX PARAMS command for the SCC. This command causes the RBPTR and TBPTR parameters of the serial channel to be updated with the new values just programmed into RBASE and TBASE.
9. Write RFCR with $18 and TFCR with $18 for normal operation.
10. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 256 bytes, so MRBLR = $0100. The value 256 is chosen to allow an entire receive frame to fit into one receive buffer.
11. Write C_MASK with $0000F0B8 to comply with 16-bit CCITT-CRC.
12. Write C_PRES with $0000FFFF to comply with 16-bit CCITT-CRC.
13. Clear DISFC, CRCEC, ABTSC, NMARC, and RETRC for clarity.
14. Write MFLR with $0100 so the maximum frame size is 256 bytes.

15. Write RFTHR with $0001 to allow interrupts after each frame.
16. Write HMASK with $0000 to allow all addresses to be recognized.
17. Clear HADDR1, HADDR2, HADDR3, and HADDR4 for clarity.
18. Initialize the Rx BD. Assume the Rx data buffer is at $00001000 in main memory. Write $B000 to Rx_BD_Status, rite $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
19. Initialize the Tx BD. Assume the Tx data frame is at $00002000 in main memory and contains five 8-bit characters. Write $BC00 to Tx_BD_Status, $0005 to Tx_BD_Length, and $00002000 to Tx_BD_Pointer.
20. Write $FFFF to the SCCE register to clear any previous events.
21. Write $001A to the SCCM register to enable the TXE, RXF, and TXB interrupts.
22. Write $20000000 to the CIMR to allow SCC2 to generate a system interrupt. The CICR should also be initialized.
23. Write $00000000 to the GSMR_H2 register to enable normal behavior of the $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ pins and idles between frames (as opposed to flags).
24. Write $00000000 to the GSMR_L2 register to configure the $\overline{\text{CTS}}$ and CD pins to automatically control transmission and reception (DIAG bits) and the HDLC mode. Normal operation of the transmit clock is used (TCI is cleared). Notice that the transmitter (ENT) and receiver (ENR) have not been enabled. If inverted HDLC operation is preferred, set the RINV and TINV bits.
25. Set the PSMR2 register to $0000 to configure one opening and one closing flag, 16-bit CCITT-CRC, and prevention of multiple frames in the FIFO.
26. Write $00000030 to the GSMR_L2 register to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

**NOTE**

After 5 bytes and CRC have been transmitted, the Tx BD is closed and the receive buffer is closed after a frame is received. Additionally received data beyond 256 bytes or a single frame causes a busy (out-of-buffers) condition since only one Rx BD is prepared.

**16.14.17.15 SCC HDLC EXAMPLE #2.** The following is an initialization sequence for an SCC HDLC channel that uses the DPLL in a Manchester encoding. The user provides a clock that is 16x the preferred bit rate on the CLK3 pin. CLK3 is then connected to the HDLC transmitter and receiver. A baud rate generator could have been used instead, if preferred. SCC2 is used. The HDLC controller is configured with the RTS2, $\overline{\text{CTS}}$2, and CD2 pins active.

1. Follow all the steps in the HDLC example #1, until the step where the GSMR is initialized.
2. Write $00000000 to the GSMR_H2 register to enable normal behavior of the $\overline{\text{CTS}}$ and CD pins, and idles between frames (as opposed to flags).
3. Write $004AA400 to the GSMR_L2 register to configure carrier sense always active, a 16-bit preamble of “01” pattern, 16x operation of the DPLL for the receiver and transmitter, Manchester encoding for the receiver and transmitter, the $\overline{\text{CTS}}$ and CD pins to automatically control transmission and reception (DIAG bits), and the HDLC mode. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled yet.
4. Set the PSMR4 register to $0000 to configure one opening and one closing flag, 16-bit CCITT-CRC, and to disallow multiple frames in the FIFO.
5. Write $004AA430 to the GSMR_L2 register to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

**NOTE**

After the preamble and 5 bytes have been transmitted, the Tx BD is closed and the receive buffer is closed after 16 bytes are received. Additionally received data beyond 16 bytes causes a busy (out-of-buffers) condition since only one Rx BD is prepared.

### 16.14.18 HDLC Bus Controller

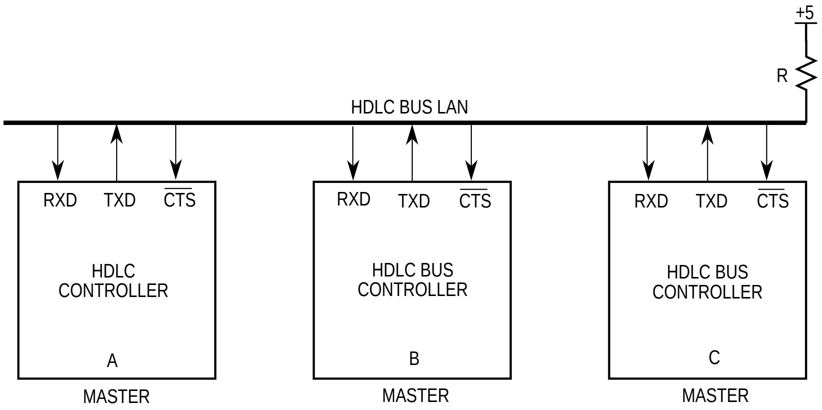
HDLC bus is an enhancement that allows an HDLC-based LAN and other point-to-multipoint configurations to be easily implemented. Most versions of HDLC-based controllers only provide point-to-point communications. HDLC bus is based on the techniques used in the CCITT ISDN I.430 and ANSI T1.605 standards for D-channel point-to-multipoint operation over the S/T interface. However, HDLC bus is not fully compliant with I.430 or T1.605 and cannot be used to replace devices that implement these protocols. Instead, it is more suited to the needs of non-ISDN LAN and point-to-multipoint configurations.

It is recommended that the reader review the basic features of I.430 and T1.605 before learning HDLC bus. I.430 and T1.605 define a method whereby eight terminals can be connected over the D-channel of the S/T bus of ISDN. The protocol used at layer 2 is a variant of HDLC, called LAPD. However, at layer 1, a method is provided to allow any of the eight terminals access to the physical S/T bus to send frames to the switch.

The S/T interface device detects whether the channel is clear by looking at an “echo” bit on the line that is designed to echo whatever bit is most recently transmitted on the D channel. Depending on the “class” of the terminal and the particular situation, the S/T interface device can wait for 7, 8, 9, or 10 ones on the echo bit before allowing the LAPD frame to begin transmission. Once transmission begins, the S/T chip monitors the data that is sent. As long as the echo bit matches the transmit data, the transmission continues. If the echo bit is ever a zero when the transmit bit is a 1, then a collision has occurred between terminals and the station(s) that transmitted a zero immediately stops further transmission. The station that transmitted a one continues normally.

In summary, I.430 and T1.605 provides a physical layer protocol that allows multiple terminals to share the same physical connection. These protocols make very efficient use of the bus by dealing with collisions in such a way that one station is always able to complete it’s transmission. Once a station completes a transmission, it lowers it’s own priority to give other devices fair access to the physical connection.

HDLC bus works much the same way; however, a few differences exist. First, HDLC bus does not use the echo bit, but rather a separate pin ($\overline{\text{CTS}}$) to monitor the data that is transmitted. The transmit data is simply connected to the $\overline{\text{CTS}}$ input. Second, HDLC bus is a synchronous digital open-drain connection for short-distance configurations, rather than the more complex definition of the S/T interface. Third, HDLC bus allows any HDLC-based frame protocol to be implemented at layer 2, not just LAPD. Fourth, HDLC bus devices wait either 8 or 10 bit times before transmitting, rather than 7, 8, 9, or 10 bits (HDLC bus has one “class” rather than two). Figure 16-85 illustrates HDLC bus in it’s most common LAN configuration. All stations can transmit and receive data to/from every other station on the LAN and all transmissions are half-duplex, as is typical in LANs.

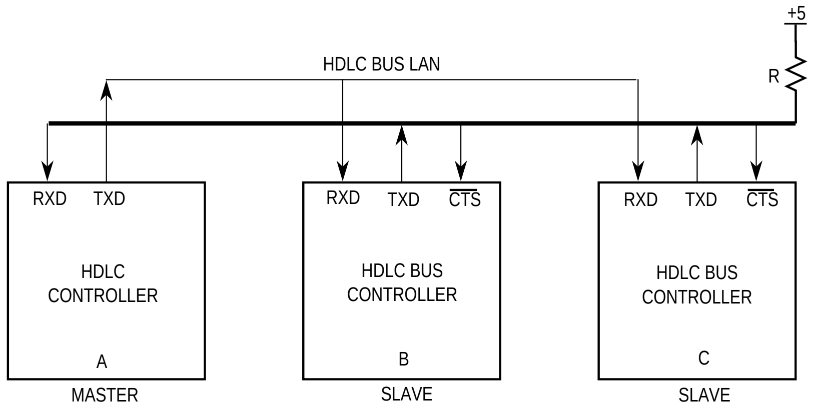


NOTES:

1. Transceivers may be used to extend the LAN size, if necessary.
2. The TXD pins of slave devices should be configured to open-drain in the port C parallel I/O port.

**Figure 16-85. HDLC Bus Multimaster Configuration**

Figure 16-86 illustrates the other LAN-type configuration of HDLC bus. In this configuration, a master station transmits to any slave station, with no collisions. The slaves only communicate with the master, but can experience collisions in their access over the bus. In this configuration, a slave that communicates with another slave must first transmit it's data to the master, where the data is buffered in RAM and then retransmitted to the other slave. The benefit of this configuration, however, is that full-duplex operation can be obtained. In a point-to-multipoint environment, this configuration is preferred.



NOTES:
1. Transceivers may be used to extend the LAN size, if necessary.
2. The TXD pins of slave devices should be configured to open-drain in the port C parallel I/O port.

**Figure 16-86. HDLC Bus Single-Master Configuration**

**16.14.18.1 FEATURES.** The following list contains HDLC bus's important features:

- Superset of the HDLC controller features
- Automatic HDLC bus access
- Automatic retransmission in case of a collision
- May be used with the NMSI mode or a TDM bus
- Delayed RTS mode

### 16.14.18.2 HDLC BUS OPERATION.

**16.14.18.2.1 Accessing the HDLC Bus.** The HDLC bus ensures an orderly access to the bus when two or more transmitters attempt to access it simultaneously. In such a case, one transmitter is always successful in completing it's transmission. This procedure relies on the HDLC flags that consist of the binary pattern 01111110 ($7E) and the zero bit insertion to prevent flag imitation.

While in the active condition (preferring to transmit), the HDLC bus controller monitors the bus through the CTS pin. It counts the number of one bits using the CTS pin and if a zero is detected, the internal counter is cleared. Once eight consecutive ones have been received, the HDLC bus controller begins transmission on the line. While it is transmitting information on the bus, the transmitted data is continuously compared with the CTS pin and is used to sample the external bus. Figure 16-87 illustrates how the CTS pin is used. The CTS sample is taken halfway through the bit time, using the rising edge of the transmit clock. If the transmitted bit is the same as the received CTS sample, the HDLC bus controller continues it's transmission. If, however, the received CTS sample is zero, but the transmitted bit was 1, the HDLC controller ceases transmission following that bit and returns to the active condition. Since the HDLC bus uses a wired-OR scheme, a transmitted zero has priority over a transmitted 1.

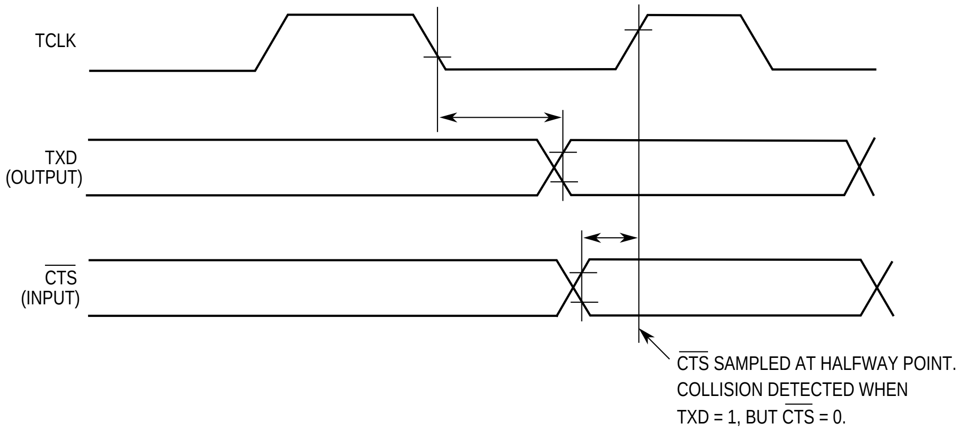


**Figure 16-87. HDLC Bus Collision Detection**

If the source address is included in the HDLC frame and the destination address, a predefined priority of nodes results. In addition, the inclusion of a source address allows collisions to be detected no later than the end of the source address.

**NOTE**

HDLC bus can be used with many different HDLC-based frame formats and does not specify the type of HDLC protocol to use.

To ensure that all stations gain an equal share of the bus, a priority mechanism is implemented in HDLC bus. Once an HDLC bus node has completed the transmission of a frame, it waits for 10 consecutive one bits, rather than just eight, before beginning the next transmission. In this way, all nodes preferring to transmit obtains the bus before a node transmits twice. Once a node detects that 10 consecutive ones have occurred on the bus, it can attempt transmission and reinstate it's original priority of waiting for eight ones.

**16.14.18.2.2 Added Performance.** Since HDLC bus is used in a wired-OR configuration, the limit of HDLC bus operation is determined by the rise time of the one bit. Figure 16-88 illustrates a method to increase performance. The user supplies a clock that is high for a shorter duration than it is low, which allows for more rise time in the case of a one bit.

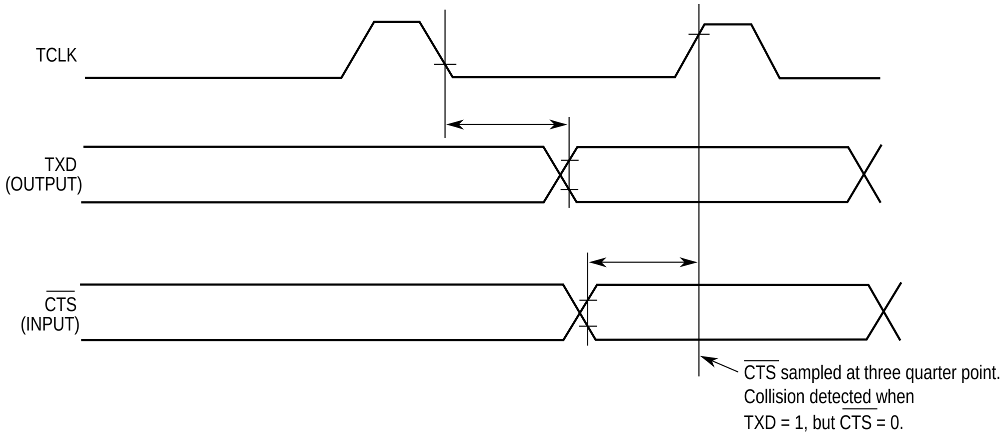
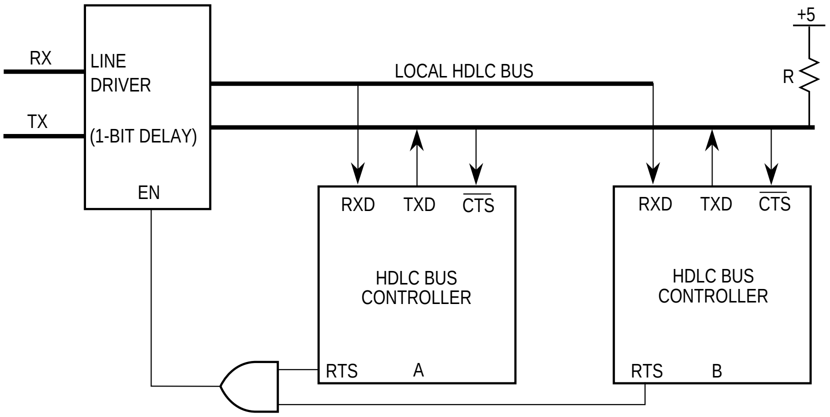


**Figure 16-88. Nonsymmetrical Duty Cycle**

**16.14.18.2.3 Delayed RTS Mode.** Sometimes HDLC bus can be used in a configuration with a local HDLC bus and a standard transmission line that is not an HDLC bus. Figure 16-89 illustrates such a case. The local HDLC bus controllers do not communicate with each other, but with a station on the transmission line; yet the HDLC bus protocol is used to control the access to the transmission line. In such a case, the RTS pin may be used as follows.



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 16-89. HDLC Bus Transmission Line Configuration**

Normally, the RTS pin goes active at the beginning of the opening flag’s first bit. Use of RTS is not required in HDLC bus. However, a mode exists on the MPC821 HDLC bus that delays the RTS signal by one bit with respect to the data. This mode is selected with the BRM bit in the PSMR.

The delayed RTS mode is useful when the HDLC bus connects multiple local nodes to a transmission line. If the transmission line driver has a one-bit delay, then the delayed RTS line can be used to enable the output of the transmission line driver. The result is that the transmission line bits always drive “clean” without any collisions occurring on them. The RTS timing is shown in Figure 16-90.

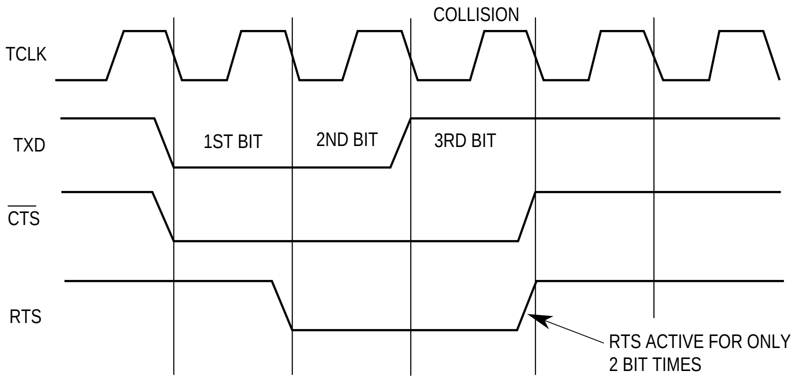
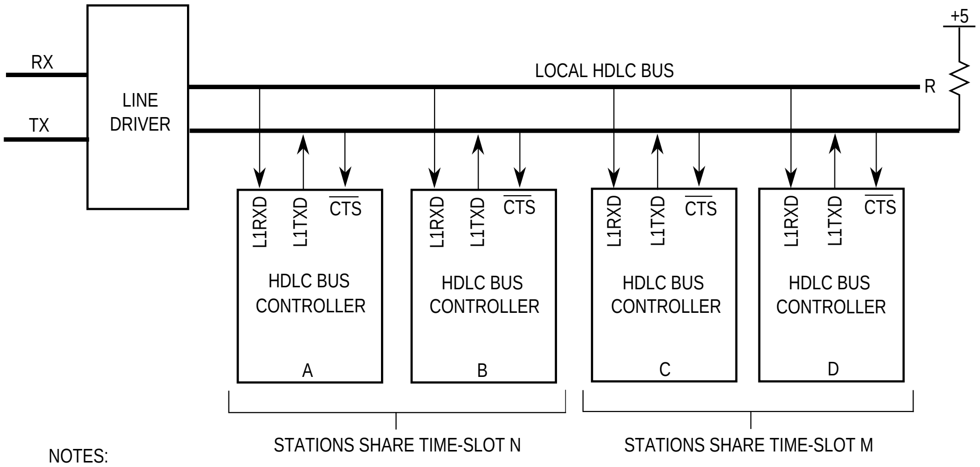


**Figure 16-90. Delayed RTS Mode**

**16.14.18.2.4 Using the TSA.** Sometimes HDLC bus can be used in a configuration that has a local HDLC bus and a TDM transmission line that is not an HDLC bus. Figure 16-91 illustrates such a case. The local HDLC bus controllers all communicate over time-slots. However, more than one HDLC bus controller is assigned to a given time-slot and the HDLC bus protocol controls access during that time-slot.



NOTES:
1. All Tx pins of slave devices should be configured to open-drain in the port C parallel I/O port.
2. The TSA in the 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 16-91. HDLC Bus TSA Transmission Line Configuration**

Once again, the local HDLC controllers do not communicate with each other, only with the transmission line. If the SCC is configured to operate using the TSA of the SI, then the data is received and transmitted using the L1TXDx and L1RXDx pins. The collision sensing is still obtained from the individual SCC $\overline{\text{CTS}}$ pin, thus the $\overline{\text{CTS}}$ pin must be configured in port C to connect to the preferred SCC. Since the SCC only receives clocks during it's time-slot, the $\overline{\text{CTS}}$ pin is only sampled during the transmit clock edges of the particular SCC time-slot.

**16.14.18.3 HDLC BUS MEMORY MAP AND PROGRAMMING.** The HDLC bus on the MPC821 is implemented using the HDLC controller with certain bits set. Otherwise, the user should consult the HDLC controller section for detailed information on the programming of HDLC.

**16.14.18.3.1 GSMR Programming.** The GSMR programming sequence is as follows:

1. Set the MODE bits to HDLC.
2. Set CTSS to 1 and all other bits to zero or to their default condition.
3. Set the DIAG bits for normal operation.
4. Set the RDCR and TDCR bits for 1x clock.
5. Set the TENC and RENC bits for NRZ.
6. Clear RTSM.
7. Set the ENT and ENR bits as preferred.

**16.14.18.3.2 PSMR Programming.** The PSMR programming sequence is as follows:

1. Set the NOF bits as preferred.
2. Set the CRC to 16-bit CRC CCITT.
3. Set the RTE bit.
4. Set the BUS bit.
5. Set the BRM bit to 1 or zero as preferred.
6. Set all other bits to zero or to their default condition.

**16.14.18.3.3 HDLC Bus Controller Example.** Except for the previously discussed register programming, the HDLC example #1 can be followed.

## 16.14.19 ASYNC HDLC Controller

Asynchronous HDLC is a frame-based protocol that uses HDLC framing techniques in conjunction with UART-type characters. This protocol is typically used as the physical layer for the point-to-point protocol (PPP) and the infra-red link access protocol (IRLAP). Even though this protocol can be implemented in conjunction with the CPU, it is more efficient and less computationally intensive for the CPU to allow the CPM of the MPC821 to perform the framing and transparency functions of the protocol.

**13.14.19.1 FEATURES.** The following is a list of the ASYNC HDLC controller’s important features:

- Flexible data buffer structure that allows an entire frame or a section of a frame to be transmitted and received
- Separate interrupts for received frames and transmitted buffers
- Automatic CRC generation and checking (CRC-CCITT)
- Support for NMSI control signals (carrier detect, clear to send, request to send)
- Automatic generation of opening and closing flags
- Reception of frames with only one “shared” flag
- Automatic generation and stripping of transparency characters according to RFC 1549 utilizing transmit and receive control character maps
- Programmable opening flag, closing flag, and control escape characters
- Automatic transmission of the ABORT sequence (control escape, closing flag) after the STOP TRANSMIT command is issued
- Automatic transmission of IDLE characters between frames and characters

**16.14.19.2 ASYNC HDLC CHANNEL FRAME TRANSMISSION PROCESSING.** The ASYNC HDLC controller is designed to work with a minimum amount of intervention from the CPU core. It operates in a similar fashion to the HDLC controller on the MPC821.

When the core enables the transmitter and sets the ready (R) bit in the first buffer descriptor, the ASYNC HDLC controller fetches the data from memory and starts transmitting the frame (after transmitting the opening flag). When the controller reaches the end of the current BD, the CRC and the closing flag are appended if the last (L) bit in the Tx BD is set. If the continuous mode (CM) bit is clear, the ASYNC HDLC transmitter writes the frame status bits into the BD and clears the ready bit. If the interrupt (I) bit is set, the controller sets the TXB event bit in the event register. Thus, the I bit can be used to generate an interrupt after each buffer, after a group of buffers, or after each complete frame has been transmitted.

If the CM bit in the Tx BD is set, the ASYNC HDLC controller writes the signal unit status bits into the BD after transmission but it does not clear the ready bit. The ASYNC HDLC controller then proceeds to the next Tx BD in the table. If it is not ready, the ASYNC HDLC controller waits until it is ready. While the ASYNC HDLC controller is transmitting data from the buffers, it automatically performs the transparency encoding specified by the protocol. This encoding is described in detail in **Section 16.14.19.4 Transmitter Transparency Encoding**.

| BOF | A | C | I | FCS | EOF |
|---|---|---|---|---|---|
| 8 BITS | 8 BITS | 8 BITS | M * 8 BITS | 2 * 8 BITS | 8 BITS |

**Figure 16-89. ASYNC HDLC Frame Structure**

If the user prefers to rearrange the transmit queue before the CP has completed transmission of all the buffers, the user should issue the STOP TRANSMIT command. This can be useful for transmitting expedited data prior to previously linked buffers or for error situations. The ASYNC HDLC controller, when receiving the STOP TRANSMIT command, stops transmitting and sends the ASYNC HDLC ABORT sequence (control escape, closing flag). It then transmits IDLE characters until the RESTART TRANSMIT command is given, at which point it resumes transmission with the next Tx BD.

**16.14.19.3 ASYNC HDLC CHANNEL FRAME RECEPTION PROCESSING.** The ASYNC HDLC receiver is designed to work with a minimum amount of intervention from the CPU core and can perform the decoding of the transparency characters, check the CRC of the frame, and detect errors on the line and in the controller.

When the core enables the receiver, the receiver waits for data to be present on the line. When the receiver detects a data byte of the incoming frame that was preceded by one or more opening flags, the ASYNC HDLC controller fetches the next BD and if the empty (E) bit is set it starts transferring the incoming frame into the BD associated data buffer. When the data buffer is full, the ASYNC HDLC controller clears the empty bit in the BD. If the incoming frame exceeds the length of the data buffer (as defined in the MRBLR parameter), the ASYNC HDLC controller fetches the next BD in the table and, if empty, continues to transfer the rest of the frame into this BD associated data buffer.

During this process, the receiver automatically performs the transparency character decoding required of the ASYNC HDLC protocol. This procedure is described in detail in **Section 16.14.19.5 Receiver Transparency Decoding**. When the frame ends, the controller checks the incoming CRC field and writes it to the data buffer. It then writes the length of the entire frame to the data length field of the last BD. The ASYNC HDLC controller sets the last (L) bit, writes the frame status bits into the BD, and clears the empty bit if the continuous mode (CM) bit is clear. It then sets the RXF bit in the event register indicating that a frame has been received and is in memory. The ASYNC HDLC controller then waits for the start of the next frame which may, or may not, have an opening FLAG.

**16.14.19.4 TRANSMITTER TRANSPARENCY ENCODING.** The ASYNC HDLC controller maps characters according to RFC 1549. It examines the outgoing data bytes and performs the transparency algorithm on a given byte if it matches one of the following criteria:

- The byte is a FLAG (0x7E-PPP, 0xC0/0xC1-IrLAP)
- The byte is a control-escape character (0x7D)
- The byte has a value between 0x00 and 0x1F and the corresponding bit in the TX control character table is set (PPP)

Once the criteria are met, a two-byte sequence is transmitted in place of the byte that consists of the control-escape character (0x7D) followed by the original byte exclusive-OR’ed with 0x20.

**16.14.19.5 RECEIVER TRANSPARENCY DECODING.** The ASYNC HDLC controller maps characters according to RFC 1549. It examines the incoming data bytes and performs the transparency algorithm to recover the original data. The algorithm accomplishes the following tasks:

- Discards any character that has it’s corresponding bit set in the RX control character map. This character is assumed to have been inserted in the character stream by an intermediate device and thus, is not part of the originally transmitted frame.
- Reverses the transmission transparency sequence by discarding a received control-escape character (0x7D) and exclusive-OR’ing the following byte with 0x20 before performing the CRC calculation and writing the byte into memory.

The receive flowchart below illustrates the algorithm because there are some cases that are not addressed in the RFC.

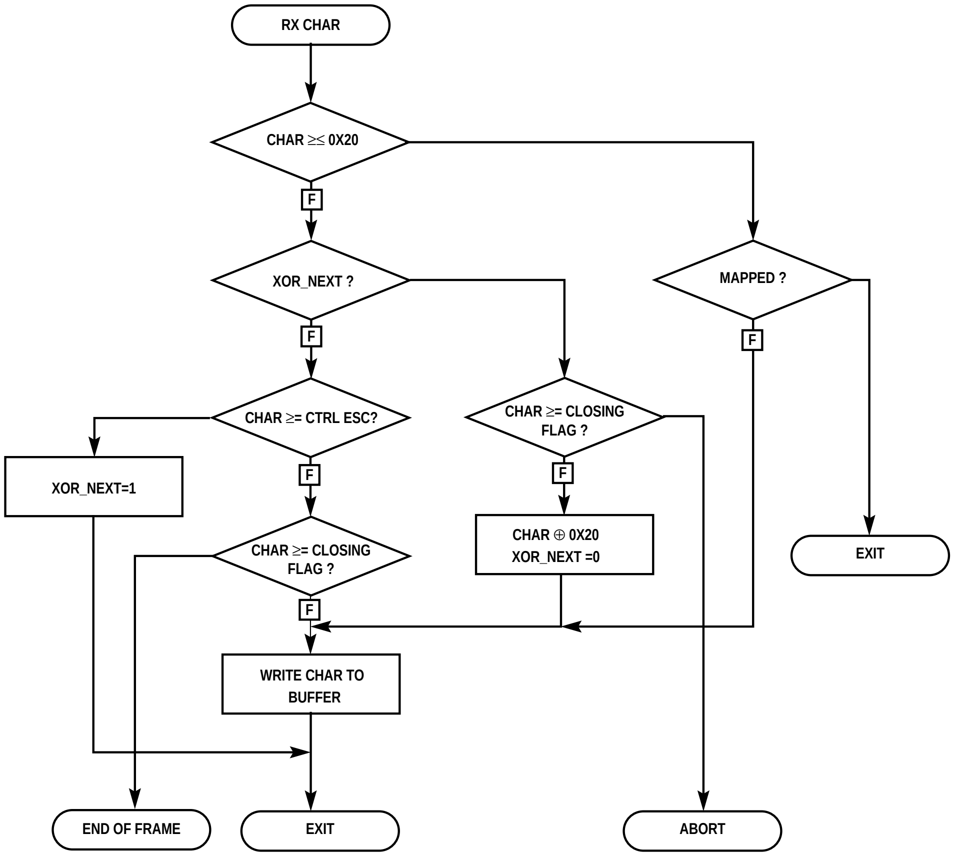


Figure 16-90. Receive Flowchart

### 16.14.19.6 CASES NOT COVERED BY RFC 1549.

- If an 0x7D is followed by a control character and the control character is not mapped, the control character itself is “modified” by the xor process. It is assumed that this will be caught by the CRC check.
- In addition to the abort sequence, frames are terminated by the following errors:
  - Carrier detect lost
  - Receiver overrun
  - Framing error
  - Break sequence
- If the invalid sequence (0x7D, 0x7D) is received, the first control escape character is discarded and the second is unconditionally exclusive-OR’ed with 0x20. This sequence is thus stored in the buffer descriptor as 0x5D.

### 16.14.19.7 IMPLEMENTATION SPECIFICS RELATED TO ASYNCHRONOUS HDLC.

**16.14.19.7.1 FLAG Sequence.** When transmitting, the controller automatically generates the opening and closing flag for the frame. When receiving, the controller strips off the opening and closing FLAG before writing the frame to memory. It receives frames with only one “shared” flag between them and ignores multiple flags between frames.

**16.14.19.7.2 Address Field.** The address field is neither generated nor examined by the microcode while transmitting or receiving. The address field of the frame must be included in the data buffer that the transmit buffer descriptor points to. Any address field compression, expansion, or checking must be performed by the core.

**16.14.19.7.3 Control Field.** The control field is neither generated nor examined by the microcode while transmitting or receiving. The control field of the frame must be included in the data buffer that the transmit buffer descriptor points to. Any control field compression, expansion, or checking must be performed by the core.

**16.14.19.7.4 Frame Check Sequence.** When transmitting, the frame check sequence (CRC) is automatically appended to the end of the frame prior to transmitting the closing flag. The frame check sequence is generated on the original frame before addition of transparency characters, start/stop bits, or flags. The controller only uses a 16-bit CRC-CCITT polynomial.

When receiving, the frame check sequence is automatically checked. The frame check sequence is calculated after removal of any transparency characters, start/stop bits, and flags. The controller only uses a 16-bit CRC-CCITT polynomial.

**16.14.19.7.5 Encoding.** The ASYNC HDLC controller only supports 8 data bits, one start bit, one stop bit, and no parity. This must be programmed in the PSMR such that bits 2 and 3 are set to 1 for proper operation.

**16.14.19.7.6 Time-Fill.** When transmitting, the ASYNC HDLC controller transmits IDLE characters (characters consisting of only “1”s) when no data is available for transmission. When receiving, the ASYNC HDLC controller ignores any IDLE characters.

**16.14.19.8 ASYNC HDLC MEMORY MAP.** When configured to operate in ASYNC HDLC mode, the MPC821 overlays the structure shown listed in Table 16-23 with the ASYNC HDLC-specific parameters described in Table 16-27.

### Table 16-27. ASYNC HDLC-Specific Parameters

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SCC Base+34 | **C_MASK** | Word | CRC Constant |
| SCC Base+38 | **C_PRES** | Word | CRC Preset |
| SCC Base+3C | **BOF** | Half-word | Beginning Of Flag Character |
| SCC Base+3E | **EOF** | Half-word | End Of Flag Character |
| SCC Base+40 | **ESC** | Half-word | Control Escape Character |
| SCC Base+42 | Reserved | Half-word | |
| SCC Base+44 | Reserved | Half-word | |
| SCC Base+46 | **Zero** | Half-word | Must be initialized to zero by user |
| SCC Base+48 | Reserved | Half-word | |
| SCC Base+4A | **RFTHR** | Half-word | Received Frames Threshold |
| SCC Base+4C | Reserved | Half-word | |
| SCC Base+4E | Reserved | Half-word | |
| SCC Base+50 | **TXCTL_TBL** | Word | TX Control Character Mapping Table |
| SCC Base+54 | **RXCTL_TBL** | Word | RX Control Character Mapping Table |
| SCC Base+58 | **NOF** | Half-word | Number of Opening Flags |
| SCC Base+5A | | Half-word | |

NOTE: Items in bold must be initialized by the user.

- C_MASK—This value should be initialized with $0000F0B8.
- C_PRES—This value should be initialized with $0000FFFF.
- BOF—This value should be initialized to the beginning of flag character value (PPP-$7E, IRLAP - $C0).
- EOF—This value should be initialized to the end of flag character (PPP-$7E, IrLAP-$C1).
- ESC—This value should be initialized to the control escape character (PPP-$7D, IrLAP - $7D).
- Reserved—These areas are temporary storage locations for the microcode. They should not be initialized or modified.
- Zero—This field must be set to zero by the user.
- Reserved—These areas are temporary storage locations for the microcode. They should not be initialized or modified.
- RFTHR—Received frames threshold indicates how many frames are received before the RXF bit is set in the event register.

- TXCTL_TBL—Transmit control character table stores the bit array used for the TX control character table. Each bit corresponds to a character that should be mapped according to RFC 1549. If the bit is set, the character corresponding to that bit is mapped and if the bit is not set, the corresponding character is not mapped. The transmit control character table should be initialized to zero for IrLAP.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 0X1F | 0X1E | 0X1D | 0X1C | 0X1B | 0X1A | 0X19 |

••••••

| 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|
| 0X05 | 0X04 | 0X03 | 0X02 | 0X01 | 0X00 |

- RXCTL_TBL—Receive control character table stores the bit array used for the RX control character table. Each bit corresponds to a character that should be mapped according to RFC 1549. If the bit is set, the character corresponding to that bit is discarded if received and if the bit is not set, the corresponding character is received normally. The receive control character table should be initialized to zero for IrLAP.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 0X1F | 0X1E | 0X1D | 0X1C | 0X1B | 0X1A | 0X19 |

••••••

| 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|
| 0X05 | 0X04 | 0X03 | 0X02 | 0X01 | 0X00 |

- NOF—This entry should be initialized to the number of opening flags to be transmitted at the beginning of a frame. A value of *n* corresponds to *n*+1 flags.

**16.14.19.9 CONFIGURING THE GENERAL SCC PARAMETERS.** The general SCC parameters can be configured as described in **Section 16.14.1 Overview** through **Section 16.14.8 Interrupts from the SCCs**, except for the following changes:

**General SCC Mode Registers.** The GSMR bits are the same except for:

RFW—Rx FIFO Width

0– Should not be used.

1– Low-latency operation. The Rx FIFO is 8-bits wide and the receive FIFO is one quarter it's normal size (8 bytes for SCC1 and 4 bytes for the other SCCs). This allows data to be written to the data buffer each time a character is received, without waiting for 32 bits to be received. This configuration must be chosen for character-oriented protocols such as UART, BISYNC, and ASYNC HDLC.

TDCR—Transmit Divide Clock Rate

The TDCR bits determine the divider rate of the transmitter. If the DPLL is not used, the 1x value should be chosen, except in asynchronous UART or HDLC mode where 8x, 16x, or 32x must be chosen. The user should program TDCR to equal RDCR in most applications.

00— Do not use.
01— 8x clock mode (do not use for IrLAP).
10— 16x clock mode.
11— 32x clock mode (do not use for IrLAP).

RDCR—Receive DPLL Clock Rate

The RDCR bits determine the divider rate of the receive DPLL. If the DPLL is not used, the 1x value should be chosen, except in asynchronous UART or HDLC mode where 8x, 16x, or 32x must be chosen. The user should program RDCR to equal TDCR in most applications.

00— Do not use.
01— 8x clock mode (do not use for IrLAP).
10— 16x clock mode.
11— 32x clock mode (do not use for IrLAP).

**Data Synchronization Register.** The DSR is reserved in asynchronous HDLC mode. It should be set to $7E7E.

**16.14.19.10 PROGRAMMING MODEL.** The core configures each SCC to operate in one of the protocols by setting the MODE bits in the GSMR. The ASYNC HDLC controller uses the same data structure as in the other protocols. This data structure supports multibuffer operation.

The receive errors are reported through the Rx BD and the transmit errors are reported through the Tx BD. An indication about the status lines (CD and $\overline{\text{CTS}}$) is reported through the port C pins. A maskable interrupt can be generated upon a status change in either one of those lines.

**16.14.19.11 ASYNC HDLC COMMAND SET.** The following commands are issued to the CPCR.

**16.14.19.11.1 Transmit Commands.** After the hardware or software is reset and the channel is enabled by the SCCM register, the channel is in the transmit enable mode and starts polling the first BD in the table every 8 transmit bit-times or immediately if the TOD bit of the TODR is set.

**STOP TRANSMIT**

This command transmits the asynchronous HDLC ABORT sequence (PPP- $7D;$7E, IrLAP - $7D;$C1) and then disables the transmission of data. If this command is received by the ASYNC HDLC controller during frame transmission, the ABORT sequence is put into the FIFO and the transmitter does not attempt to send any more data from the current Tx BD. The controller does not advance to the next Tx BD. The user determines which BD is terminated by examining the TBPTR entry in the SCC parameter RAM table. However, no new BD is accessed for this channel.

**NOTE**

Unlike the other MPC821 protocols, the asynchronous HDLC controller does not flush the FIFO because of the STOP TRANSMIT command. Thus, up to 16 characters (32 on SCC1) can be transmitted ahead of the ABORT sequence. This can be avoided by programming TFL to 1 in the GSMR.

### GRACEFUL STOP TRANSMIT

This command is not supported by the asynchronous HDLC controller.

### RESTART TRANSMIT

This command reenables the transmission of characters on the transmit channel and is expected by the ASYNC HDLC controller after a STOP TRANSMIT command or a transmitter error. The ASYNC HDLC controller resumes transmission from the first character in the current transmitter BD (TBPTR) in the channel's transmit BD table.

### INIT TX PARAMETERS

This command initializes all the transmit parameters in this serial channel's parameter RAM to their reset state. It must be issued before the transmitter is first enabled and should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the transmit and receive parameters.

**16.14.19.11.2 Receive Commands .** After the hardware or software is reset and the channel is enabled by it's SCCM register, the channel is in the receive enable mode and uses the first BD in the table.

### ENTER HUNT MODE

This command is used to force the ASYNC HDLC controller to close the current Rx BD if it is being used and enter the hunt mode. The ASYNC HDLC resumes reception after detecting a frame that was preceded by one or more opening flags.

### CLOSE RX BD

This command is not supported by the ASYNC HDLC controller.

### INIT RX PARAMETERS

This command initializes all the receive parameters in this serial channel's parameter RAM to their reset state and should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the receive and transmit parameters.

**16.14.19.12 ASYNC HDLC ERROR HANDLING PROCEDURE.** The ASYNC HDLC controller reports frame reception and transmission error conditions using the channel BDs and the ASYNC HDLC event register.

### 16.14.19.12.1 Transmission Errors.

### CTS Lost During Frame Transmission

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the clear to send lost (CT) bit in the Tx BD, and sets the TXE bit in the SCCE register. The channel resumes transmission from the next Tx BD after the RESTART TRANSMIT command is issued.

#### 16.14.19.12.2 Reception Errors.

**Overrun Error**

The ASYNC HDLC controller maintains an internal 8 byte FIFO in SCC1 and 4 byte FIFO in the other SCCs receiving data. A receive overrun occurs when the CP is unable to keep up with the data rate or the SDMA channel is unable to write the received data to memory. The previous data byte and the frame status are lost. The controller closes the buffer with the overrun (OV) bit in the BD set and sets the RXF bit in the SCCE register. The receiver then searches for the next frame.

**CD Lost During Frame Reception**

When this error occurs, the channel terminates frame reception, closes the buffer, and sets the carrier detect lost (CD) bit in the BD and the RXF bit in the event register. This error has the highest priority. The rest of the frame is lost and other errors are not checked in that frame. The receiver then searches for the next frame once CD is reasserted.

**Abort Sequence**

An abort sequence is detected by the ASYNC HDLC controller when the ABORT sequence is received (0x7d followed by 0x7e). When this error occurs, the channel closes the buffer (if already open) by setting the Rx abort sequence (AB) bit in the BD and sets the RXF bit in the SCCE register. The CRC error status condition is not checked on aborted frames. If the ABORT sequence is received and no frame is currently being received, the next BD is opened and then closed with the AB bit set.

**CRC Error**

When this error occurs, the channel writes the received CRC (cyclic redundancy check) to the data buffer, closes the buffer, and sets the CRC error (CR) bit in the BD and the RXF bit in the SCCE register. After receiving a signal unit with a CRC error, the receiver prepares to receive the next frame.

**Break Sequence Received**

This error occurs when the UART receiver detects the first character of a break sequence (one or more all-zero characters). The channel closes the buffer (if already open) by setting the Rx break sequence (BRK) bit in the BD and sets the RXF bit in the SCC event register. The CRC error status condition is not checked. The break start (BRKs) event is set in the SCCE register when the first break of a break sequence is detected and the break end (BRKe) event is set when an idle bit is received after a break sequence.

### 16.14.19.13 ASYNC HDLC REGISTERS.

**16.14.19.13.1 Event Register.** The SCCE register is called the ASYNC HDLC event register when the SCC is operating in asynchronous HDLC mode. The ASYNC HDLC event register is a 16-bit register that is used to generate interrupts and report events recognized by the ASYNC HDLC channel. Upon recognition of an event, the ASYNC HDLC controller sets the corresponding bit in the ASYNC HDLC event register. Interrupts generated by this register can be masked by the ASYNC HDLC mask register.

The ASYNC HDLC event register is a memory-mapped register that can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. However, all unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset.

SCCE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | GLr | GLt | RES | | IDL | RES | BRKe | BRKs | TXE | RXF | BSY | TXB | RXB |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ADDR | A10 (SCCE1), A30 (SCCE2) | | | | | | | | | | | | | | | |

Bits 0–2—Reserved

These bits should be written with zeros.

GLr—Glitch on Rx

A clock glitch is detected by the SCC on the receive clock

GLt—Glitch on Tx

A clock glitch is detected by the SCC on the transmit clock

Bits 5–6—Reserved

These bits should be written with zeros.

IDL—Idle Sequence Status Changed

A change in the status of the serial line is detected and the real-time status of the line can be read in SCCS.

Bit 8—Reserved

This bit should be written with a zero.

TXE—Tx Error

An error (CTS lost) occurs on the transmitter channel.

BRKe—Break End

The end of a break sequence is detected and this indication is set no sooner than after one idle bit is received after a break sequence.

BRKs—Break Start

A break character is received. This is the first break of a break sequence. The user does not receive multiple BRKs events if a long break sequence is detected.

RXF—Rx Frame

A complete frame is received on the ASYNC HDLC channel and this bit is set no sooner than two bit times after the receipt of the last bit of the closing flag.

BSY—Busy Condition

A frame is received and discarded due to a lack of buffers.

TXB—Transmit Buffer

A buffer that had it’s I bit set is transmitted on the ASYNC HDLC channel. This bit is set no sooner than when the last bit of the closing flag begins it’s transmission if the buffer is the last one in the frame. Otherwise, this bit is set after the last byte of the buffer is written to the transmit FIFO.

RXB—Rx Buffer

A buffer is received over the ASYNC HDLC channel that had it’s I bit set but not the L bit.

**16.14.19.13.2 ASYNC HDLC Mode Register.** Each ASYNC HDLC mode register is a 16-bit, memory-mapped, read/write register that controls SCC operation. The term ASYNC HDLC mode register refers to the PSMR when that SCC is configured in ASYNC HDLC mode.

PSMR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | FLC | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | | | | | | | | | | | | | | | | |

FLC—Flow Control

0– Normal operation.

1– Asynchronous flow control. When the $\overline{\text{CTS}}$ pin is negated, the transmitter stops transmitting at the end of the current character. If $\overline{\text{CTS}}$ is negated past the middle of the current character, the next full character can be sent and transmission is stopped. When $\overline{\text{CTS}}$ is asserted once more, transmission continues where it left off and no CTS lost error is reported. No characters except idles are transmitted while $\overline{\text{CTS}}$ is negated.

Bit 1—Reserved

This bit must be set to zero.

Bits 2 and 3—CHLN (1:0)

On other protocols, these bits determine the number of data bits in a character. For ASYNC HDLC and IrLap these bits must be set to 1.

Bits 4–15—Reserved

These bits must be set to zero.

**16.14.19.14 ASYNC HDLC RX BUFFER DESCRIPTOR.** The ASYNC HDLC controller uses the Rx BD to report information about the received data for each buffer. An example of the Rx BD process is illustrated in Figure 16-83.

The first word of the Rx BD contains control and status bits. Bits 0 through 3 and bit 6 are written by the user and bits 8–15 and 4–5 are set by the CP following frame reception. Bit 0 is set by the core when the buffer is available to the ASYNC HDLC controller and it is cleared by the ASYNC HDLC controller when the buffer is full. The format of the control and status word is detailed below.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **E** | RES | **W** | **I** | L | F | **CM** | RES | BRK | BOF | RES | | **AB** | CR | OV | CD |
| **OFFSET + 2** | DATA LENGTH | | | | | | | | | | | | | | | |
| **OFFSET + 4** | RX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

E—Empty

0 = The data buffer associated with this BD is filled with received data or data reception is aborted because of an error condition. The CPU core is free to examine or write to any fields of this Rx BD. The CP does not use this BD again as long as the E-bit is zero.

1 = The data buffer associated with this BD is empty or reception is currently in progress. This Rx BD and it's associated receive buffer are owned by the CP. Once the E-bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last buffer descriptor in the Rx BD table.

1 = This is the last buffer descriptor in the Rx BD 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 Rx BDs in this table are programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = The RXB bit in the ASYNC HDLC event register is not set after this buffer is used, but RXF operation remains unaffected.
1 = The RXB or RXF bit in the ASYNC HDLC event register is set when this buffer is used by the ASYNC HDLC controller.

L—Last in Frame

This bit is set by the ASYNC HDLC controller when this buffer is the last in a frame. This implies the reception of a closing flag or reception of an error, in which case one or more of the BRK, CD, OV, and AB bits are set. The ASYNC HDLC controller writes the number of frame octets to the data length field.

0 = This buffer is not the last one in a frame.
1 = This buffer is the last one in a frame.

F—First in Frame

This bit is set by the ASYNC HDLC controller when this buffer is the first in a frame.

0 = The buffer is not the first one in a frame.
1 = The buffer is the first one in a frame.

CM—Continuous Mode

0 = Normal operation.
1 = The E-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this BD. However, the E-bit is cleared if an error (other than CRC) occurs during reception, regardless of the CM bit.

Bit 7—Reserved

BRK—Break Character Received

The current frame is closed because a break character is received.

BOF—Beginning of Frame Encountered

The current frame is closed because a BOF character is received instead of the expected EOF.

Bits 10–11—Reserved

AB—Rx Abort Sequence

The ASYNC HDLC abort sequence or a framing error is received to terminate this frame.

CR—Rx CRC Error

This frame contains a CRC error. The received CRC bytes are always written to the receive buffer.

OV—Overrun

A receiver overrun occurs during frame reception.

CD—Carrier Detect Lost

The carrier detect signal is negated during frame reception.

Data Length

Data length is the number of octets the CP writes into this BD data buffer once the BD is closed. When this BD is the last one in the frame (L = 1), the data length contains the total number of frame octets (including 2 for CRC).

**NOTE**

If the received frame has a length (plus CRC) that is an exact multiple of MRBLR, the BD with the “L” bit set does not actually have any characters in it and the “data length” field contains a value equal to the sum of the “data length” fields of the other buffer descriptors in the frame.

The actual amount of memory allocated for this buffer should be greater than or equal to the contents of the MRBLR.

Rx Data Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, can reside in the internal or external memory.

**16.14.19.15 ASYNC HDLC TX BUFFER DESCRIPTOR.** Data is presented to the ASYNC HDLC controller for transmission on an SCC channel by arranging it in buffers referenced by the channel TX BD table. The HDLC controller confirms transmission (or indicates error conditions) using the BDs to inform the CPU core that the buffers have been serviced.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **R** | RES | **W** | **I** | L | RES | **CM** | RES | RES | RES | RES | RES | RES | RES | RES | CT |
| OFFSET + 2 | **DATA LENGTH** | | | | | | | | | | | | | | | |
| OFFSET + 4 | TX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or it's associated data buffer. The CP clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which has been prepared for transmission by the user, is not transmitted yet or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Tx BD table.

1 = This is the last BD in the Tx BD table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to in the table. The number of Tx BDs in this table are programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0= The TXB bit in the ASYNC HDLC event register is not set after this buffer is used.

1= The TXB bit in the ASYNC HDLC event register is set when this buffer is transmitted by the ASYNC HDLC controller.

L—Last

0= This is not the last buffer in the current frame.

1= This is the last buffer in the current frame. The proper CRC and closing FLAG are transmitted after the last byte is transmitted.

Bit 5—Reserved

CM—Continuous Mode

0 = Normal operation.

1 = The R-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically retransmitted when the CP next accesses this BD. However, the R-bit is cleared if an error occurs during transmission, regardless of the CM bit.

Bits 7–14—Reserved

The following status bits are written by the ASYNC HDLC controller after it finishes transmitting the associated data buffer.

$\overline{\text{CTS}}$—CTS Lost

$\overline{\text{CTS}}$ in NMSI mode is lost during frame transmission. If data from more than one buffer is currently in the FIFO when this error occurs, this bit is set in the currently open TX BD.

Data Length

Data length is the number of bytes the ASYNC HDLC controller should transmit from this BD data buffer. It is never modified by the CP. The value of this field must be greater than zero.

Tx Data Buffer Pointer

The transmit buffer pointer, which contains the address of the associated data buffer, may be even or odd and can reside in the internal or external memory. This value is never modified by the CP.

**16.14.19.16 DIFFERENCES BETWEEN HDLC AND ASYNC HDLC.** The ASYNC HDLC controller does not work exactly like the standard HDLC controller. This section details some of the differences.

**16.14.19.16.1 Maximum Received Frame Length Counter.** There is no maximum received frame length counter in the ASYNC HDLC controller. Therefore, the controller receives all characters between opening and closing flags. There is no way to have the controller stop writing to memory. This in no way affects the number of bytes received into a specific buffer descriptor. It just means that a frame over the maximum length is received into memory in its entirety.

**16.14.19.16.2 Frame Termination Due To Error.** If frame reception terminates due to an error condition (CD lost, overrun, break character received), the character being received at the time that the error occurred is not written into memory. For example, if a CD lost error occurred, the frame is closed and the partial character is not written to memory. Thus, the octet count only reflects the number of bytes written to memory.

**NOTE**

The GRACEFUL STOP TRANSMIT command is not supported.

**16.14.19.16.3 Automatic Error Counters.** The automatic error counters in HDLC mode (CRCEC and ABTEC) have not been implemented in ASYNC HDLC.

**16.14.19.16.4 Noisy Characters.** Noisy characters (those whose three samples are not the same) are not accounted for in this controller. It is assumed that the CRC catches any data integrity problems.

**16.14.19.17 PROGRAMMING EXAMPLE.** The following list is a suggested initialization sequence for ASYNC HDLC.

1. Initialize the SDCR.
2. Configure ports A and C pints to enable RXD, TXD, $\overline{\text{CTS}}$, CD, and RTS. This assumes the user is using NMSI mode. If not, the time-slot assigner and TSA pins should be appropriately configured.
3. Configure a BRG to generate appropriate channel clocking frequency.
4. Program the SICR to route the BRG clocking to the SCC running ASYNC HDLC.
5. Select whether the channel is using the time-slot assigner or the NMSI pins in the SICR.
6. Write RBASE and TBASE in the SCC parameter RAM to point to the first RxBD and the first TxBD.
7. Issue the INIT RX and TX PARAMETERS commands for that SCC.
8. Program RFCR and TFCR.
9. Write MRBLR with the maximum receive buffer size.
10. Write C_MASK and C_PRES with the standard values.
11. Write the Zero register to 0x0000.
12. Program the RFTHR to the number of frames that should be received before an interrupt is generated.
13. Program the TX and RX Control Character Tables.
14. Initialize all RxBDs.
15. Initialize all TxBDs.
16. Clear the SCCE register by writing $FFFF to it.
17. Program the SCCM register with the proper mask to allow all preferred interrupts.
18. Program the GSMR_H register.
19. Program the GSMR_L register to ASYNC HDLC mode, but do not turn on the transmitter or receiver.
20. Set the PSMR appropriately. Refer to **Section 16.14.19.13.2 ASYNC HDLC Mode Register** for more information.
21. Turn on the transmitter and receiver in the GSMR_L register.

## 16.14.20 AppleTalk Controller

AppleTalk is a set of protocols developed by Apple Computer Inc. to provide a LAN service between Macintosh computers and printers. Although AppleTalk can be implemented over a variety of physical and link layers, including Ethernet, the AppleTalk protocols have traditionally been more closely associated with one particular physical and link layer protocol called LocalTalk.

The term LocalTalk refers to an HDLC-based link and physical layer protocol that runs at the rate of 230.4 kbps. In this document, the term AppleTalk controller refers to a support that the MPC821 provides for the LocalTalk protocol. The AppleTalk controller provides the required frame synchronization, bit sequence, preamble, and postamble onto standard HDLC frames. These capabilities, as well as the use of the HDLC controller in conjunction with the DPLL operating in FM0 mode, provide the proper connection formats to the LocalTalk bus.

**16.14.20.1 LOCALTALK BUS OPERATION.** A LocalTalk frame is a modified HDLC frame as illustrated in Figure 16-91.

| SYNC SEQ | HDLC FLAGS | DEST. ADDR. | SOURCE ADDR. | CONTROL BYTE | DATA (OPTIONAL) | CRC-16 | CLOSING FLAG | ABORT SEQUENCE |
|---|---|---|---|---|---|---|---|---|
| > 3 BITS | 2 OR MORE BYTES | 1 BYTE | 1 BYTE | 1 BYTE | 0–600 BYTES | 2 BYTES | 1 BYTE | 12–18 ONES |

**Figure 16-91. LocalTalk Frame Format**

First, a synchronization sequence of greater than three bits is sent. This sequence consists of at least one logical one bit (FM0 encoded) followed by greater than two bit times of line idle. No particular maximum time is specified for this line idle time. The idle time allows some LocalTalk equipment to sense 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/detection. Two bytes of address, destination and source, are transmitted next. This is followed by a byte of control and 0 to 600 data bytes. Next, two bytes of CRC are sent. The CRC is the common 16-bit CRC-CCITT polynomial referenced in the HDLC standard protocol. The LocalTalk frame is then terminated by a flag and a restricted HDLC abort sequence (a sequence of 12 to 18 logical ones). The transmitter’s driver is then disabled.

The control byte within the LocalTalk frame indicates the type of frame. Control byte values from 0x01 to 0x7f are data frames and control byte values from 0x80 to 0xff are control frames. Four different control frames are currently defined—$\overline{\text{ENQ}}$ (Enquiry), ACK (ENQ acknowledgment), RTS (request to send a data frame), and $\overline{\text{CTS}}$ (clear to send a data frame).

Frames are sent in groups known as dialogs. For instance, to transfer a data frame, three frames are actually sent over the network. An RTS frame (not to be confused with the RS-232 pin RTS) is sent requesting the network, a $\overline{\text{CTS}}$ frame is sent by the destination node, and the data frame is sent by the requesting node. These three frames comprise one possible type of dialog. Once a dialog has begun, other nodes cannot begin transmission until the dialog is complete. Dialogs are typically handled in the software.

Frames within a dialog are transmitted with a maximum interframe gap (IFG) of 200 microseconds. Although the LocalTalk specification does not state it, there is also a minimum recommended IFG of 50 microseconds. Dialogs must be separated by a minimum interdialog gap (IDG) of 400 microseconds. In general, these gaps are implemented via the software.

Due to the protocol definition, collisions should only be encountered during RTS and ENQ frames. Once a frame's transmission is started, it is fully transmitted, regardless of whether it collides with another frame. ENQ frames are infrequent, being sent only when a node is powered 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. FM0 requires one level transition on every bit boundary. If the value to be encoded is a logic zero, FM0 also requires a second transition in the middle of the bit time. The purpose of the FM0 encoding is to eliminate the need to transmit clocking information on a separate wire. With FM0, the clocking information is present whenever valid data is present.

**16.14.20.2 FEATURES.** The following is a list of the AppleTalk controller's important features:

- Superset of the HDLC controller features
- Provides FM0 encoding/decoding
- Programmable transmission of sync sequence
- Automatic postamble transmission
- Reception of sync sequence does not cause extra CD interrupts
- Reception is automatically disabled while transmitting a frame
- Transmit-on-demand feature expedites frames
- Connects directly to an RS-422 transceiver

**16.14.20.3 APPLETALK HARDWARE CONNECTION.** The MPC821 connects to LocalTalk and interfaces to the RS-422 transceiver through the TXD, RTS, and RXD pins. The RS-422, in turn, interfaces to the LocalTalk connector. Although it is not shown, a passive RC circuit is recommended between the transceiver and connector. Refer to Figure 16-92 for details.

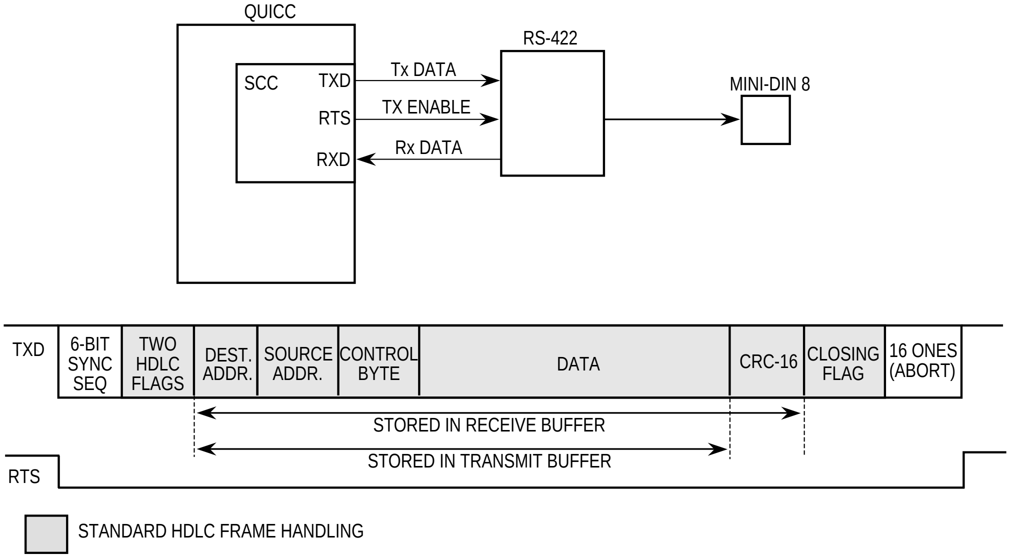


**Figure 16-92. Connecting the MPC821 to LocalTalk**

The 16x overspeed clock of 3.686 MHz can be generated from an external frequency source or from one of the baud rate generators if the resulting BRG output frequency is close to a multiple of the 3.686-MHz frequency (within the tolerance specified by LocalTalk). The MPC821 asserts the RTS signal for the complete duration of the frame thus, RTS can be used to enable the RS-422 transmit driver.

**16.14.20.4 APPLETALK MEMORY MAP AND PROGRAMMING MODEL.** The AppleTalk controller on the MPC821 is implemented using the HDLC controller with certain bits set. Otherwise, the user should consult **Section 16.14.18 HDLC Bus Controller** for detailed information on the programming of HDLC.

**16.14.20.4.1 GSMR Programming.** The GSMR programming sequence is as follows:

1. The MODE bits should be set to AppleTalk.
2. The ENT and ENR bits should be set.
3. The DIG bits should be set for normal operation, with the CD and $\overline{\text{CTS}}$ pins grounded or configured for parallel I/O, which causes CD and $\overline{\text{CTS}}$ to be internally asserted to the SCC.
4. The RDCR and TDCR bits should usually be set to 16x clock.
5. The TENC and RENC bits should be set for FM0.
6. The Tend bit should be zero.
7. The TPP bits should be 11.

8. The TPL bits should be set to 000 to transmit the next frame with no synchronization sequence and to 001 to transmit the next frame with the LocalTalk synchronization sequence. For example, data frames do not require a preceding synchronization sequence. These bits may be modified on-the-fly if the AppleTalk protocol is selected.
9. The TINV and RINV bits should be zero.
10. The TSNC bits should be set to 1.5 bit times 10.
11. The EDGE bits should be zero.
12. RTSM should be zero.
13. All other bits should be set to zero or to their default condition.

**16.14.20.4.2 PSMR Programming.** The PSMR programming sequence is as follows:

1. The NOF bits should be set to 0001 (binary) giving two flags before frames (one opening flag, plus one additional flag).
2. The CRC should be set to 16-bit CRC-CCITT.
3. The DRT bit should be set.
4. All other bits should be set to zero or to their default condition.

**16.14.20.4.3 TODR Programming.** To expedite a transmit frame, the TODR can be used.

**16.14.20.4.4 AppleTalk Controller Example.** Except for the previously discussed register programming, the HDLC example #1 can be followed.

## 16.14.21 BISYNC Controller

The byte-oriented BISYNC protocol was originated by IBM for use in networking products. The three classes of BISYNC frames are transparent, nontransparent with header, and nontransparent without header. Refer to Figure 16-93 for details. The transparent mode in BISYNC allows full binary data to be transmitted 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.

**NOTE**

The transparent frame type in BISYNC is not related to the totally transparent protocol supported by the MPC821. Refer to **Section 16.14.22 Transparent Controller** for details.

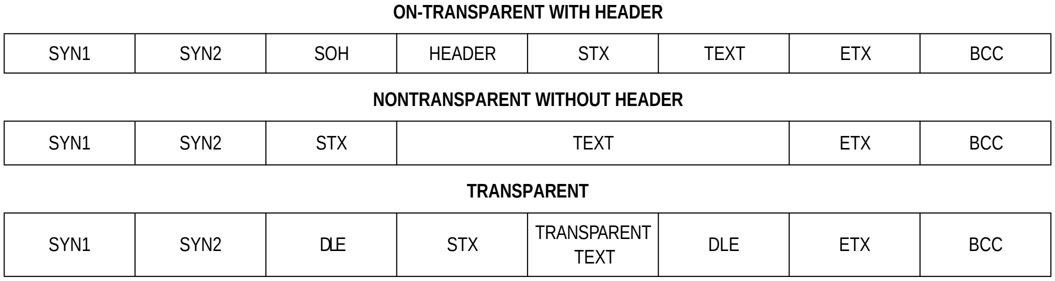


Figure 16-93. Typical BISYNC Frames

The bulk of the frame is divided into fields whose meaning depends on the frame type. The BCC is a 16-bit CRC (CRC16) format if 8-bit characters are used. It is a longitudinal check (a sum check) in combination with vertical redundancy check (parity) if 7-bit characters are used. In transparent operation, to allow the BISYNC control characters to be present in the frame as valid text data, a special character (DLE) is defined that informs the receiver that the character following the DLE is a text character and not a control character. If a DLE is transmitted as valid data, it must be preceded by a DLE character. This technique is sometimes called byte-stuffing. The physical layer of the BISYNC communications link must provide a means of synchronizing the receiver and transmitter. This is usually accomplished by sending at least one pair of synchronization characters prior to every frame.

BISYNC is unusual in that a transmit underrun need not be an error. If an underrun occurs, the synchronization pattern is transmitted until data is once again ready to transmit. The receiver discards the additional synchronization characters as they are received. In nontransparent operation, all synchronization characters (SYNCs) are discarded. In transparent operation, all DLE-SYNC pairs are discarded. Correct operation in this case assumes that, on the transmit side, the underrun does not occur between the DLE and it’s following character, a failure mode that is prevented in the MPC821.

By appropriately setting the SCC mode register, any of the SCC channels can be configured to function as a BISYNC controller that handles the basic functions of the BISYNC protocol in normal and transparent modes. The SCC in BISYNC mode can work with the TSA or NMSI. The SCC supports modem lines by connecting to the port C pins or using the general-purpose I/O pins. The BISYNC controller consists of separate transmit and receive sections whose operations are asynchronous with the CPU core and can either be synchronous or asynchronous with respect to the other SCCs.

**16.14.21.1 FEATURES.** The following is a list of the BISYNC controller's important features:

- 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 generation/checking
- Parity (VRC) generation/checking
- Supports BISYNC transparent operation (use of DLE characters)
- Maintains parity error counter
- Reverse data mode

**16.14.21.2 BISYNC CHANNEL FRAME TRANSMISSION.** The BISYNC transmitter is designed to work with almost no intervention from the CPU core. When this CPU core enables the BISYNC transmitter, it starts transmitting SYN1–SYN2 pairs (located in the data synchronization register) or idles as programmed in the BISYNC mode register. The BISYNC controller polls the first BD in the transmit channel's BD table. If there is a message to transmit, the BISYNC controller fetches the data from memory and starts transmitting the message (after first transmitting the SYN1–SYN2 pair). The entire SYN1–SYN2 pair is always transmitted, regardless of the programming of the SYNL bits in the GSMR.

When a BD data has been completely transmitted, L-bit is checked. If both the L-bit and transmit BCS bit are set in that BD, the BISYNC controller appends the CRC16/LRC. Subsequently, the BISYNC controller writes the message status bits into the BD and clears the R-bit. It then starts transmitting SYN1–SYN2 pairs or idles as programmed in the RTSM bit in the GSMR. When the end of the current BD has been reached and the last bit is not set (working in multibuffer mode), only the R-bit 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 the transmission of each buffer, a specific buffer, or each block. The BISYNC controller then proceeds to the next BD in the table.

If no additional buffers have been presented to the BISYNC controller for transmission, an in-frame underrun is detected and the BISYNC controller begins transmitting SYNCs or idles. If the BISYNC controller was in transparent mode, the BISYNC controller transmits DLE-SYNC pairs. Characters are included in the block check sequence (BCS) calculation on a per-buffer basis. Each buffer can be independently programmed to be included or excluded from the BCS calculation and any characters to be excluded from the BCS calculation must reside in a separate buffer. The BISYNC controller can reset the BCS generator before transmitting a specific buffer. When functioning in transparent mode, the BISYNC controller automatically inserts a DLE before transmitting a DLE character. In this case, only one DLE is used in the calculation of the BCS.

**16.14.21.3 BISYNC CHANNEL FRAME RECEPTION.** Although the BISYNC receiver is designed to work with almost no intervention from the CPU core, it allows user intervention on a per-byte basis if necessary. The BISYNC receiver performs CRC16, longitudinal redundancy check (LRC), or vertical redundancy check (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. A control character is discussed in more detail in **Section 16.14.21.6 BISYNC Control Character Recognition**.

When the CPU core enables the BISYNC receiver, it enters hunt mode. In this mode, as data is shifted into the receiver shift register one bit at a time, the contents of the register are compared to the contents of the SYN1–SYN2 fields in the data synchronization register. If the two are not equal, the next bit is shifted in and the comparison is repeated. When the registers match, the hunt mode is terminated and character assembly begins. The BISYNC controller is now character synchronized and performs SYNC stripping and message reception. The BISYNC controller reverts to the hunt mode when it is issued the ENTER HUNT MODE command, on recognition of some error condition, or on reception of an appropriately defined control character.

When receiving data, the BISYNC controller updates the BCS bit (CR) in the BD for every byte transferred. When the data buffer has been filled, the BISYNC controller clears the E-bit in the BD and generates an interrupt if the I-bit in the BD is set. If the incoming data exceeds the length of the data buffer, the BISYNC controller fetches the next BD in the table and, if it is empty, continue transferring data to this BD associated data buffer.

When a BCS is received, it is checked and written to the data buffer. The BISYNC controller sets the last bit, writes the message status bits into the BD, and clears the E-bit. Then it generates a maskable interrupt, indicating that a block of data has been received and is in memory. Notice that the SYNCs in the nontransparent mode or DLE-SYNC pairs in the transparent mode (an underrun condition) are not included in the BCS calculations.

**NOTE**

The receive FIFO width (RFW) bit in the GSMR must be set for an 8-bit receive FIFO for the BISYNC receiver.

**16.14.21.4 BISYNC MEMORY MAP.** When configured to operate in BISYNC mode, the MPC821 overlays the structure listed in Table 16-23 with the BISYNC-specific parameters described in Table 16-28.

**Table 16-28. BISYNC-Specific Parameters**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SCC Base + 30 | RES | Word | Reserved |
| SCC Base + 34 | **CRCC** | Word | CRC Constant Temp Value |
| SCC Base + 38 | **PRCRC** | Half-word | Preset Receiver CRC16/LRC |
| SCC Base + 3A | **PTCRC** | Half-word | Preset Transmitter CRC16/LRC |
| SCC Base + 3C | **PAREC** | Half-word | Receive Parity Error Counter |
| SCC Base + 3E | **BSYNC** | Half-word | BISYNC SYNC Character |
| SCC Base + 40 | **BDLE** | Half-word | BISYNC DLE Character |
| SCC Base + 42 | **CHARACTER1** | Half-word | CONTROL Character 1 |
| SCC Base + 44 | **CHARACTER2** | Half-word | CONTROL Character 2 |
| SCC Base + 46 | **CHARACTER3** | Half-word | CONTROL Character 3 |
| SCC Base + 48 | **CHARACTER4** | Half-word | CONTROL Character 4 |
| SCC Base + 4A | **CHARACTER5** | Half-word | CONTROL Character 5 |
| SCC Base + 4C | **CHARACTER6** | Half-word | CONTROL Character 6 |
| SCC Base + 4E | **CHARACTER7** | Half-word | CONTROL Character 7 |
| SCC Base + 50 | **CHARACTER8** | Half-word | CONTROL Character 8 |
| SCC Base + 52 | **RCCM** | Half-word | Receive Control Character Mask |

NOTE: Items in bold must be initialized by the user.

- PRCRC and PTCRC—These values should be preset to all ones or zeros, depending on the BCS used.
- PAREC—This 16-bit (modulo $2^{16}$) counter is maintained by the CP. It can be initialized by the user while the channel is disabled. The counter counts parity errors on receive if the parity feature of BISYNC is enabled.
- BSYNC—This register contains the value of the SYNC to be transmitted in an underrun condition, transmitted as the second byte of a DLE–SYNC pair and stripped from incoming data on receive once the receiver has synchronized to the data using the DSR and SYN1–SYN2 pair.
- BDLE—This register contains the value to be transmitted as the first byte of a DLE–SYNC pair and stripped on receive.
- CHARACTER1–8—These values represent control characters that can be recognized by the BISYNC controller.

- RCCM—This value is used to mask the comparison of CHARACTER1–8 so that classes of control characters can be defined. A 1enables the bit comparison and a zero masks it.

The CPU core configures each SCC to operate in one of the protocols by the MODE bits in the GSMR. The SYN1–SYN2 synchronization characters are programmed in the data synchronization register.

The BISYNC controller uses the same basic data structure as the other modes. Receive and transmit errors are reported through their respective BDs. The status of the line is reflected via the port C pins and a maskable interrupt can be generated at each status change.

There are two basic ways of handling the BISYNC channels. First, data can be inspected on a per-byte basis, with the BISYNC controller interrupting the CPU core on receipt of every byte of data. Second, the BISYNC controller can be operated so that the software is only necessary for handling the first two to three bytes of data. Subsequent data (until the end of the block) can be handled by the BISYNC controller without interrupting the CPU core.

**16.14.21.5 COMMAND SET.** The following transmit and receive commands are issued to the CPCR.

#### 16.14.21.5.1 Transmit Commands.

**STOP TRANSMIT**

After the hardware or software is reset and the channel is enabled in the SCC mode register, the channel is in the transmit enable mode and starts polling the first BD in the table every 64 transmit clocks (immediately if the TOD bit in the TODR is set).

This command aborts transmission after a maximum of 64 additional bits are transmitted, without waiting until the end of the buffer is reached and the transmit FIFO is flushed. The TBPTR is not advanced. No new BD is accessed, and no new buffers are transmitted for this channel. SYNC characters consisting of SYNC–SYNC or DLE–SYNC pairs (according to the transmitter mode) is continually transmitted until transmission is reenabled by issuing the RESTART TRANSMIT command. The STOP TRANSMIT command can be used when it is necessary to abort transmission and transmit an EOT control sequence. The EOT sequence should be the first buffer presented to the BISYNC controller for transmission after reenabling transmission.

**NOTE**

The BISYNC controller remains in the transparent or normal mode after receiving the STOP TRANSMIT or RESTART TRANSMIT commands.

### GRACEFUL STOP TRANSMIT

This command is used to stop transmission smoothly rather than abruptly, as performed by the regular STOP TRANSMIT command. It stops transmission after the current frame has completed transmission or immediately if there is no frame being transmitted. The GRA bit in the SCCE register is set once transmission stops. Then the BISYNC transmit parameters (including BDs) can be modified. The TBPTR points to the next Tx BD in the table. Transmission begins once the R-bit of the next BD is set and the RESTART TRANSMIT command is issued.

### RESTART TRANSMIT

This command enables the transmission of characters on the transmit channel. It is expected by the BISYNC controller after a STOP TRANSMIT command, after a STOP TRANSMIT command and disabling the channel in its SCC mode register, after a GRACEFUL STOP TRANSMIT command, or after a transmitter error (underrun or CTS lost). The BISYNC controller resumes transmission from the current TBPTR in the channel's Tx BD table.

### INIT TX PARAMETERS

This command initializes all the transmit parameters in the serial channel's parameter RAM to their reset state and should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS command can also be used to reset the transmit and receive parameters.

### 16.14.21.5.2 Receive Commands.

### RESET BCS CALCULATION

This command immediately resets the receive BCS accumulator. For example, it can be used to reset the BCS after recognizing a control character (such as SOH), signifying that a new block is commencing.

### ENTER HUNT MODE

After the hardware or software is reset and the channel is enabled in the SCC mode register, the channel is in the receive enable mode and uses the first BD in the table.The ENTER HUNT MODE command is used to force the BISYNC controller to abort reception of the current block and enter the hunt mode. Once in the hunt mode, the BISYNC controller continually scans the input datastream for the SYN1–SYN2 sequence as programmed in the data synchronization register. After receiving the command, the current receive buffer is closed and the BCS is reset. Message reception continues using the next BD.

### CLOSE Rx BD

This command is used to force the SCC to close the current Rx BD if it is currently being used and to use the next BD for any subsequent data that is received. If the SCC is not in the process of receiving data, no action is taken by this command.

### INIT RX PARAMETERS

This command initializes all the receive parameters in this serial channel's parameter RAM to their reset state. This command should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS command can also be used to reset the receive and transmit parameters.

**16.14.21.6 BISYNC CONTROL CHARACTER RECOGNITION.** The BISYNC controller recognizes special control characters that are used to customize the BISYNC protocol implemented by the BISYNC controller and to aid it's operation in a DMA-oriented environment. Mainly, they are used for receive buffers longer than one byte. In single-byte buffers, each byte can easily be inspected and control character recognition should be disabled.

The purpose of the control characters table is to enable automatic recognition by the BISYNC controller of the end of the current block. Since the BISYNC controller imposes no restrictions on the format of the BISYNC blocks, user software must respond to the received characters and inform the BISYNC controller of mode changes and certain protocol events like resetting the BCS. However, correct use of the control characters table allows the remainder of the block to be received without interrupting the user software.

Up to eight control characters can be defined that inform the BISYNC controller that the end of the current block has been reached and whether a BCS is expected following the character. For example, the end of text (ETX) character 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 and each entry consists of the control character, an end-of-table bit, a BCS expected bit, and a hunt mode bit. The RCCM entry is used to define classes of control characters with a masking option.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SCC BASE + 42 | E | B | H | | | | | | CHARACTER1 | | | | | | | |
| SCC BASE + 44 | E | B | H | | | | | | CHARACTER2 | | | | | | | |
| SCC BASE + 46 | E | B | H | | | | | | CHARACTER3 | | | | | | | |
| SCC BASE +48 | E | B | H | | | | | | CHARACTER4 | | | | | | | |
| | | | | | | | | • | | | | | | | | |
| | | | | | | | | • | | | | | | | | |
| SCC BASE + 50 | E | B | H | | | | | | CHARACTER8 | | | | | | | |
| SCC BASE + 52 | 1 | 1 | 1 | | | | | | MASK VALUE(RCCM) | | | | | | | |

E—End of Table

0 = This entry is valid. The lower eight bits are checked against the incoming character.
1 = The entry is not valid. No valid entries exist beyond this entry.

**NOTE**

In tables with eight control characters, the E-bit should be zero in all eight positions.

B—BCS Expected

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.

**NOTE**

A maskable interrupt is generated after the buffer is closed.

H—HUNT MODE

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.

CHARACTER1–8—Control Character Value

These fields define control characters.

**NOTE**

When using 7-bit characters with parity, the parity bit should be included in the control character value.

RCCM—Received Control Character Mask

The value in this register is used to mask the comparison of CHARACTER1–8. The lower eight bits of RCCM correspond to the lower eight bits of CHARACTER1–8 and are decoded as follows.

0 = Mask this bit in the comparison of the incoming character and CHARACTER1–8.
1 = The address comparison on this bit proceeds normally and no masking occurs.

**NOTE**

Bits 0 through 2 of RCCM must be set, or erratic operation can occur during the control character recognition process.

**16.14.21.7 BSYNC-BISYNC SYNC REGISTER.** The 16-bit, memory-mapped, read/write BSYNC register is used to define the BISYNC stripping and insertion of the SYNC character. When an underrun occurs during message transmission, the BISYNC controller inserts SYNC characters until the next data buffer is available for transmission. When the BISYNC receiver is not in hunt mode and a SYNC character has been received, the receiver discards this character if the valid bit is set.

BSYNC-BISYNC SYNC REGISTER

| BITS | 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 | | | | | | | | | | | | | | | | |
| 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 | SCC BASE + 3E | | | | | | | | | | | | | | | |

**NOTE**

When using 7-bit characters with parity, the parity bit should be included in the SYNC register value.

**16.14.21.8 BDLE-BISYNC DLE REGISTER.** The 16-bit, memory-mapped, read/write BDLE register is used to define the BISYNC stripping and insertion of the DLE character. When the BISYNC controller is in transparent mode and an underrun occurs during message transmission, the BISYNC controller inserts DLE-SYNC pairs until the next data buffer is available for transmission.

BDLE-BISYNC DLE REGISTER

| BITS | 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 | | | | | | | | | | | | | | | | |
| 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 | SCC BASE = 40 | | | | | | | | | | | | | | | |

When the BISYNC receiver is in transparent mode and a DLE character is received, the receiver discards this character and excludes it from the BCS if the valid bit is set. If the second (next) character is a SYNC character, the BISYNC controller discards it and excludes it from the BCS. If the second character is a DLE, the BISYNC controller writes it to the buffer and includes it in the BCS. If the character is not a DLE or SYNC, the BISYNC controller examines the control characters table and acts accordingly. If the character is not in the table, the buffer is closed with the DLE follow character error (DLE) bit set. If the valid bit is not set, the receiver treats the character as a normal character.

## NOTE

When using 7-bit characters with parity, the parity bit should be included in the DLE register value.

**16.14.21.9 TRANSMITTING AND RECEIVING THE SYNCHRONIZATION SEQUENCE.** The BISYNC channel can be programmed to transmit and receive a synchronization pattern that is defined in the DSR. The length of the SYNC pattern is defined in the SYNL bits in the GSMR. The receiver synchronizes on the synchronization pattern that is located in the DSR. If the SYNL bits specify a nonzero synchronization pattern, then the transmitter sends the entire contents of the DSR prior to each frame, starting with the LSB first. Thus, the user can repeat the preferred SYNC pattern in the other DSR bits as well.

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | 4-BIT SYNC | | | | | | | | | | | | | | | |
| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| FIELD | 8-BIT SYNC | | | | | | | | | | | | | | | |
| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| FIELD | 16-BIT SYNC | | | | | | | | | | | | | | | |

**16.14.21.10 BISYNC ERROR-HANDLING PROCEDURE.** The BISYNC controller reports message reception and transmission error conditions using the channel BDs, the error counters, and the BISYNC event register. The modem interface lines can also be directly monitored via the port C pins.

### 16.14.21.10.1 Transmission Errors.

**Transmitter Underrun**

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the UN bit in the BD, and generates the TXE interrupt if it is enabled. The channel resumes transmission after the RESTART TRANSMIT command is received. Underrun cannot occur between frames or during a DLE–*XXX* pair in transparent mode.

**CTS Lost During Message Transmission**

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the CTS lost bit in the BD, and generates the TXE interrupt if it is enabled. The channel resumes transmission after the RESTART TRANSMIT command is received.

#### 16.14.21.10.2 Reception Errors.

**Overrun Error**

The BISYNC controller maintains an internal FIFO for receiving data. The CP begins programming the SDMA channel (if the data buffer is in external memory) and updating the CRC when the first byte is received into the FIFO. If a FIFO overrun occurs, the BISYNC controller writes the received data byte to the internal FIFO over the previously received byte. The previous character and it's status bits are lost. Then, the channel closes the buffer, sets the OV bit in the BD, and generates the RX interrupt if it is enabled. Finally, the receiver enters hunt mode.

**CD Lost During Message Reception**

When this error occurs, the channel terminates message reception, closes the buffer, sets the carrier detect lost bit in the BD, and generates the RX interrupt if it is enabled. This error has the highest priority and the rest of the message is lost, no other errors are checked in the message. Then the receiver immediately enters hunt mode.

**Parity Error**

When this error occurs, the channel writes the received character to the buffer and sets the PR bit in the BD. The channel terminates message reception, closes the buffer, sets the PR bit in the BD, and generates the RX interrupt if it is enabled. The channel also increments the PAREC and the receiver immediately enters hunt mode.

**CRC Error**

The channel updates the CR bit in the BD every time a character is received with a byte delay (eight serial clocks) between the status update and the CRC calculation. When using control character recognition to detect the end of the block and cause the checking of the CRC that follows, the channel closes the buffer, sets the CR bit in the BD, and generates the RX interrupt if it is enabled.

**16.14.21.11 BISYNC MODE REGISTER.** Each BISYNC mode register is a 16-bit, memory-mapped, read/write register that controls SCC operation. The term BISYNC mode register refers to the PSMR when that SCC is configured for BISYNC mode. This register is cleared at reset. Some of the PSMR bits can be modified on-the-fly (while the receiver and transmitter are enabled).

PSMR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | NOS | | | | CRC | | RBCS | RTR | RVD | DRT | RES | | RPM | | TPM | |
| 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 | A08 (PSMR1), A28 (PSMR2) | | | | | | | | | | | | | | | |

NOS—Minimum Number of SYNCs Between or Before Messages

If NOS0–NOS3 = 0000, then one SYN1–SYN2 pair is transmitted; if NOS0–NOS3 = 1111, then 16 SYN1–SYN2 pairs are transmitted. The SYN1–SYN2 pair is defined in the DSR. The entire SYN1–SYN2 pair is always transmitted regardless of the setting of the SYNL bits in the GSMR. The NOS bits can be modified on-the-fly.

CRC—CRC Selection

00 = Reserved.
01 = CRC16 (BISYNC). X16 + X15 + X2 + 1. The PRCRC and PTCRC registers should be initialized to a preset value of all zeros or all ones before the channel is enabled. In both cases, 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.
10 = Reserved.
11 = LRC (sum check). (BISYNC). For even LRC, the PRCRC and PTCRC registers should be initialized to zero before the channel is enabled, but for odd LRC, they should be initialized to ones.

The receiver checks character parity when BCS is programmed to LRC and the receiver is not in transparent mode. The transmitter transmits character parity when BCS is programmed to LRC and the transmitter is not in transparent mode. Use of parity in BISYNC assumes the use of 7-bit data characters.

RBCS—Receive Block Check Sequence

The BISYNC receiver internally stores two BCS calculations with a byte delay (eight serial clocks) between them. This enables the user to examine a received data byte and then decide whether or not it should be part of the BCS calculation. This is useful when control character recognition and stripping are to be performed in the software. The bit should be set (or reset) within the time taken to receive the following data byte. When this bit is reset, the BCS calculations exclude the latest fully received data byte. When RBCS is set, the BCS calculations continue normally.

0 = Disable receive BCS.
1 = Enable receive BCS.

RTR—Receiver Transparent Mode

0 = The receiver is placed in normal mode with SYNC stripping and control character recognition operative.
1 = The receiver is placed in transparent mode. SYNCs, DLEs, and control characters are only recognized after a leading DLE character. The receiver calculates the CRC16 sequence, even if it is programmed to LRC while in transparent mode. PRCRC should be initialized to the CRC16 preset value before setting this bit.

RVD—Reverse Data

0 = Normal operation.
1 = Any portion of this SCC that is defined to operate in BISYNC mode (either the receiver, transmitters or both) operates by reversing the character bit order and transmitting the MSB first.

DRT—Disable Receiver While Transmitting

0 = Normal operation.
1 = While data is being transmitted by the SCC, the receiver is disabled and then gated by the internal $\overline{\text{RTS}}$ signal. This is useful if the BISYNC channel is being configured onto a multidrop line and the user does not want to receive their own transmission. Notice that although BISYNC is usually implemented as a half-duplex protocol, the receiver is not actually disabled during transmission. Thus, for typical BISYNC operation, DRT should not be set.

Bits 10–11—Reserved

RPM—Receiver Parity Mode

The RPM bits select the type of parity check to be performed by the receiver. The RPM bits can be modified on-the-fly and are ignored unless the CRC bits are selected to be LRC.

00 = Odd Parity.
01 = Low Parity (always check for a zero in the parity bit position).
10 = Even Parity.
11 = High Parity (always check for a one in the parity bit position).

When odd parity is selected, the transmitter counts the number of ones in the data word. If the total number of ones is not an odd number, the parity bit is set to 1 and thus produces an odd number. If the receiver counts an even number of ones, an error in transmission has occurred. In the same manner, for even parity, an even number must result from the calculation performed at both ends of the line. In high/low parity, if the parity bit is not high/low, a parity error is reported. The receive parity errors cannot be disabled, but can be ignored.

TPM—Transmitter Parity Mode

The TPM bits select the type of parity to be performed by the transmitter and can be modified on-the-fly. These bits are ignored unless the CRC bits are selected to be 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).

**16.14.21.12 BISYNC RECEIVE BUFFER DESCRIPTOR.** The CP reports information about the received data for each buffer using BDs. The CP closes the current buffer, generates a maskable interrupt, and starts receiving data into the next buffer after one of the following events occurs:

1. A user-defined control character is received.
2. An error is detected.
3. A full receive buffer is detected.
4. The ENTER HUNT MODE command is issued.
5. The CLOSE Rx BD command is issued.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **E** | RES | **W** | **I** | L | F | **CM** | RES | DE | RES | | NO | RES | CR | OV | CD |
| **OFFSET + 2** | DATA LENGTH | | | | | | | | | | | | | | | |
| **OFFSET + 4** | RX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

E—Empty

0 = The data buffer associated with this Rx BD has been filled with received data, or data reception has been aborted because of an error condition. The CPU core is free to examine or write to any fields of this Rx BD. The CP does not use this BD as long as the E-bit is zero.

1 = The data buffer associated with this Rx BD is empty or reception is currently in progress. This Rx BD and it's associated receive buffer are owned by the CP. Once the E-bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last buffer descriptor in the Rx BD table.

1 = This is the last buffer descriptor in the Rx BD 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 Rx BDs in this table are programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is used.

1 = The RX bit in the BISYNC event register is set when the BISYNC controller closes this buffer. The RX bit can cause an interrupt if it is enabled.

L—Last in Frame

This bit is set by the transparent controller when this buffer is the last in a frame. This implies that $\overline{\text{CD}}$ is negated in envelope mode or an error is received, in which case one or more of the OV, CD, and DE bits are set. The transparent controller writes the number of frame octets to the data length field.

0 = The buffer is not the first one in a frame.
1 = The buffer is the first one in a frame.

F—First in Frame

This bit is set by the transparent controller 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.

CM—Continuous Mode

0 = Normal operation.
1 = The E-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this BD. However, the E-bit is cleared if an error occurs during reception, regardless of the CM bit.

Bit 7—Reserved

DE—DPLL Error

This bit is set by the BISYNC controller when a DPLL error occurs while this buffer is received. In decoding modes where a transition is promised every bit, the DPLL error is set when a missing transition occurs.

Bits 9–10—Reserved

NO—Rx NonOctet Aligned Frame

A frame that contains a number of bits approximately divisible by eight is received.

Bit 12—Reserved

CR—Rx CRC Error

This frame contains a CRC error. The received CRC bytes are always written to the receive buffer.

OV—Overrun

A receiver overrun occurs during frame reception.

CD—Carrier Detect Lost

The carrier detect signal is negated during frame reception.

Data Length

The data length is the number of octets that the CP writes into this BD data buffer, including the BCS (if selected). In BISYNC mode, the data length should initially be set to zero by the user. It is incremented each time a received character is written to the data buffer.

**NOTE**

> The actual amount of memory allocated for this buffer should be greater than or equal to the contents of the MRBLR.

Rx Data Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, can be even or odd. The buffer can reside in internal or external memory.

**16.14.21.13 BISYNC TRANSMIT BUFFER DESCRIPTOR.** Data is presented to the CP for transmission on an SCC channel by arranging it in buffers referenced by the channel Tx BD table. The CP confirms transmission or indicates error conditions using the BDs to inform the processor that the buffers have been serviced.

The status and control bits are prepared by the user before transmission and are set by the CP after the buffer is transmitted.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **R** | RES | **W** | **I** | **L** | **TB** | **CM** | **BR** | **TD** | **TR** | **B** | RESERVED | | | UN | CT |
| OFFSET + 2 | **DATA LENGTH** | | | | | | | | | | | | | | | |
| OFFSET + 4 | **TX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or it's associated data buffer. The CP clears this bit after the buffer has been transmitted or after an error condition is encountered.

1 = The data buffer, which has been prepared for transmission by the user, has not been transmitted or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Tx BD table.
1 = This is the last BD in the Tx BD table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to in the table. The number of Tx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.
1 = Either TX or TXE in the BISYNC event register is set after this buffer is serviced by the CP, which can cause an interrupt.

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 (remains in) normal mode after sending the last character in the buffer and the BCS if it is enabled.

TB—Transmit BCS

This bit is only valid when the L-bit is set.

0 = Transmit the SYN1–SYN2 sequence or idle (according to the RTSM bit in the GSMR) after the last character in the buffer.
1 = Transmit the BCS sequence after the last character. The BISYNC controller also resets the BCS generator after transmitting the BCS.

CM—Continuous Mode

0 = Normal operation.
1 = The R-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this BD. However, the R-bit is cleared if an error occurs during transmission, regardless of the CM bit.

BR—BCS Reset

0 = The transmitter BCS accumulation is not reset.
1 = The transmitter BCS accumulation is reset (used for STX or SOH) before sending the data buffer.

TD—Transmit DLE

0 = No automatic DLE transmission is to occur before the data buffer.
1 = The transmitter transmits a DLE character before sending the data buffer, which saves writing the first DLE to a separate data buffer when working in transparent mode. Refer to the TR bit for information on control characters.

TR—Transparent Mode

0 = The transmitter enters (remains in) the normal mode after sending the data buffer. In this mode, the transmitter automatically inserts SYNCs in an underrun condition.

1 = The transmitter enters or remains in transparent mode after sending the data buffer. In this mode, the transmitter automatically inserts DLE–SYNC pairs in the underrun condition. Underrun occurs when the BISYNC controller finishes a buffer with the L-bit set to zero and the next BD is not available. The transmitter also checks all characters before sending them. If a DLE is detected, another DLE is automatically sent. The user must insert a DLE or program the BISYNC controller to insert it (using TD) before each control character required. The transmitter calculates the CRC16 BCS even if the BCS bit in the BISYNC mode register is programmed to LRC. The PTCRC should be initialized to CRC16 before setting this bit.

B—BCS Enable

0 = Buffer consists of characters to be excluded from the BCS accumulation.

1 = Buffer consists of characters to be included in the BCS accumulation.

Bits 9–10—Reserved

The following status bits are written by the CP after it has finished transmitting the associated data buffer.

UN—Underrun

The BISYNC controller encounters a transmitter underrun condition while transmitting the associated data buffer.

CT—CTS Lost

CTS is lost during message transmission.

Data Length

The data length is the number of octets that the CP should transmit from this BD data buffer. It is never modified by the CP. The data length should be greater than zero.

Tx Data Buffer Pointer

The transmit buffer pointer, which always points to the first byte of the associated data buffer, can be even or odd. The buffer can reside in the internal or external memory.

**16.14.21.14 BISYNC EVENT REGISTER.** The SCCE is called the BISYNC event register when the SCC is operating as a BISYNC controller. It is a 16-bit register used to report events recognized by the BISYNC channel and to generate interrupts. On recognition of an event, the BISYNC controller sets the corresponding bit in the BISYNC event register. Interrupts generated by this register can be masked in the BISYNC mask register.

The BISYNC event register is a memory-mapped register that can be read at any time. A bit is reset by writing a 1(writing a zero does not affect a bit value) and more than one bit can may be reset at a time. All unmasked bits must be reset before the CP negates the internal interrupt request signal. This register is cleared at reset.

SCCE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | GLr | GLt | DCC | RES | | GRA | RES | | TXE | RCH | BSY | TX | RX |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | A10 (SCCE1), A30 (SCC2) | | | | | | | | | | | | | | | |

Bits 0–2—Reserved

These bits must be set to 0.

GLr—Glitch on Rx

A clock glitch is detected by the SCC on the receive clock.

GLt—Glitch on Tx

A clock glitch is detected by the SCC on the transmit clock.

DCC—DPLL CS Changed

The carrier sense status as generated by the DPLL changes state. The real-time status can be found in SCCS. This is not the CD pin status that is discussed elsewhere and it is only valid when the DPLL is used.

Bits 6–7—Reserved

GRA—Graceful Stop Complete

A graceful stop, initiated by the GRACEFUL STOP TRANSMIT command, is now complete. This bit is set as soon the transmitter finishes transmitting any message that is in progress when the command was issued. It is set immediately if no message is in progress when the command was issued.

Bits 9–10—Reserved

TXE—Tx Error

An error (CTS lost or underrun) occurs on the transmitter channel.

RCH—Receive Character

A character is received and written to the buffer.

BSY—Busy Condition

A character is received and discarded due to a lack of buffers. The receiver resumes reception after an ENTER HUNT MODE command.

TX—Tx Buffer

A buffer has been transmitted. This bit is set as the last bit of data or the BCS (if sent) begins transmission.

RX—Rx Buffer

A receive buffer has been closed by the CP on the BISYNC channel.

**16.14.21.15 BISYNC MASK REGISTER.** The SCCM is referred to as the BISYNC mask register when the SCC is operating as a BISYNC controller. It is a 16-bit read/write register that has the same bit format as the BISYNC event register. If a bit in the BISYNC mask register is a 1, the corresponding interrupt in the event register is enabled. If the bit is zero, the corresponding interrupt in the event register is masked. This register is cleared at reset.

**16.14.21.16 SCC STATUS REGISTER.** The SCCS is an 8-bit read-only register that allows the user to monitor real-time status conditions on the RXD line. The real-time status of the $\overline{\text{CTS}}$ and CD pins are part of the port C parallel I/O.

SCCS REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | CS | RES |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R | R |
| ADDR | A17 (SCCS1), A37 (SCCS2) | | | | | | | |

CS—Carrier Sense (DPLL)

This bit shows the real-time carrier sense of the line as determined by the DPLL.

0 = The DPLL does not sense a carrier.
1 = The DPLL senses a carrier.

**16.14.21.17 PROGRAMMING THE BISYNC CONTROLLER.** There are two general techniques that the software employs to handle data received by the BISYNC controllers. The simplest way is to allocate single-byte receive buffers, request (in the status word in each BD) an interrupt on reception of each buffer (byte), and implement the BISYNC protocol entirely in the software on a byte-by-byte basis. This simple approach is flexible and 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 when multibyte buffers are prepared and linked to the receive buffer table and the software is used to analyze the first two to three bytes of the buffer to determine what type of block is being received. Once this is determined, reception continues without further intervention from the user's software until a control character is encountered. The control character signifies the end of the block, causing the software to revert back to a byte-by-byte reception mode.

To accomplish this, the RCH bit in the BISYNC mask register should be set initially, enabling an interrupt on every byte of data received to allow the software to analyze the type of block being received on a byte-by-byte basis. After analyzing the initial characters of a block, the user should either set the RTR bit in the BISYNC mode register or issue the RESET BCS CALCULATION command. For example, if DLE-STX is received, transparent mode should be entered. By setting the appropriate bit in the BISYNC mode register, the BISYNC controller automatically strips the leading DLE from <DLE-character> sequences. Thus, control characters are only recognized when they follow a DLE character. The RTR bit should be cleared after a DLE-ETX is received.

Alternatively, after receiving an SOH, the RESET BCS CALCULATION command should be issued. This command causes the SOH to be excluded from BCS accumulation and the BCS to be reset. Notice that the RBCS bit in the BISYNC mode register (used to exclude a character from the BCS calculation) is not needed here since SYNCs and leading DLEs (in transparent mode) are automatically excluded by the BISYNC controller.

After recognizing the type of block above, the RCH interrupt should be masked. Data reception then continues without further interruption of the CPU core until the end of the current block is reached. This is defined by the reception of a control character matching the one programmed in the receive control characters table. The control characters table should be set to recognize the end of the block as shown in the following table.

| 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 the end of text (ETX), a BCS is expected, then the buffer should be closed. Hunt mode should be entered when the line turnaround occurs (BISYNC is normally half-duplex). ENQ characters are used to abort transmission of a block and for the receiver, this character designates the end of the block, but no CRC is expected. Following control character reception (end of the block), the RCH bit in the BISYNC mask register should be set, reenabling interrupts for each byte of received data.

**16.14.21.18 SCC BISYNC EXAMPLE.** The following list is an initialization sequence for an SCC BISYNC channel assuming an external clock is provided. SCC2 is used. The BISYNC controller is configured with the RTS2, $\overline{\text{CTS2}}$, and CD2 pins active. The CLK3 pin is used for both the BISYNC receiver and transmitter.

1. Configure the port A pins to enable the TXD2 and RXD2 pins. Write PAPAR and PAODR bits 13 and 12 with ones and PADIR bits 13 and 12 with zeros.
2. Configure the port C pins to enable RTS2, $\overline{\text{CTS2}}$, and CD2. Write PCPAR bit 14 with one and bits 9 and 8 with zeros, PCDIR bits 14, 9, and 8 with zeros, and PCSO bits 9 and 8 with ones.
3. Configure port A to enable the CLK3 pin. Write PAPAR bit 5 with a one and PADIR bit 5 with a zero.
4. Connect the CLK3 pin to SCC2 using the SI. Write the R2CS and T2CSbits in SICR to 110.
5. Connect the SCC2 to the NMSI (its own set of pins) and clear the SC2 bit in the SICR.
6. Write to the SDCR to initialize the SDMA Configuration Register.
7. Write RBASE and TBASE in the SCC parameter RAM to point to the Rx BD and Tx BD in the dual-port RAM. Assuming one Rx BD at the beginning of dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.
8. Program the CPCR to execute the INIT RX and TX PARAMS command for the SCC. For instance, to execute this command for SCC1 write $0041 to the CR. This command causes the RBPTR and TBPTR parameters of the serial channel to be updated with the new values just programmed into RBASE and TBASE.
9. Write RFCR and TFCR with $18 for normal operation.
10. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = $0010.
11. Write PRCRC with $0000 to comply with CRC16.
12. Write PTCRC with $0000 to comply with CRC16.
13. Clear PAREC for clarity.
14. Write BSYNC with $8033, assuming a SYNC value of $33.
15. Write BDLE with $8055, assuming a DLE value of $55.
16. Write CHARACTER1–8 with $8000. They are not used.
17. Write RCCM with $E0FF. It is not used.

18. Initialize the Rx BD and assume the Rx data buffer is at $00001000 in main memory. Then write $B000 to Rx_BD_Status, $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
19. Initialize the Tx BD and assume the Tx data buffer is at $00002000 in main memory and contains five 8-bit characters. Then write $BD20 to Tx_BD_Status, $0005 to Tx_BD_Length, and $00002000 to Tx_BD_Pointer.
20. Write $FFFF to the SCCE register to clear any previous events.
21. Write $0013 to the SCCM register to enable the TXE, TX, and RX interrupts.
22. Write $20000000 to the CIMR so that SCC2 can generate a system interrupt. The CICR should also be initialized.
23. Write $00000020 to the GSMR_H2 register to configure a small receive FIFO width.
24. Write $00000008 to the GSMR_L2 register to configure the $\overline{\text{CTS}}$ and CD pins to automatically control transmission and reception (DIAG bits) and the BISYNC mode. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled yet.
25. Set the PSMR to $0600 to configure CRC16, CRC checking on receive, and normal operation (not transparent).
26. Write $00000038 to the GSMR_L2 register to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

**NOTE**

After 5 bytes are transmitted, the Tx BD is closed. The receive buffer is closed after 16 bytes are received. Any other received data beyond 16 bytes causes a busy (out-of-buffers) condition since only one Rx BD is prepared.

### 16.14.22 Transparent Controller

The transparent controller allows serial data to be transmitted and received over an SCC without any modification to that datastream. Transparent mode provides a clear channel on which the SCC performs no bit-level manipulation. Any protocol ran over transparent mode is performed in the software. The job of an SCC in transparent mode is to function as a high-speed serial-to-parallel and parallel-to-serial converter. This mode is also referred to as a totally transparent or promiscuous operation.

There are several basic applications for transparent mode. First, some data needs to be moved serially, but requires no protocol superimposed—for example, voice data. Second, some board-level applications require a serial-to-parallel and parallel-to-serial conversion. This is accomplished to allow communication between chips on the same board. Third, some applications require the switching of data without interfering with the protocol encoding itself. For instance, in a multiplexer, data from a high-speed time-multiplexed serial stream is multiplexed into multiple low-speed datastreams. The concept is to switch the data path without altering the protocol encoded on that data path.

By appropriately setting the GSMR, any of the SCC channels can be configured to function in transparent mode. The MPC821 receives and transmits the entire serial bitstream transparently. This mode is configured by selecting the TTx and TRx bits in the GSMR for the transmitter and receiver, respectively. However, both bits must be set for full-duplex transparent operation.

If just one of the TTx or TRx bits is set, the other half of the SCC operates with another protocol as programmed in the MODE bits of the GSMR. This allows loopback modes to DMA data from one memory location to another while converting the data to a specific serial format. The SCC in transparent mode can work with the TSA or NMSI and support modem lines using the general-purpose I/O pins. The data can be transmitted and received with MSB or LSB first in each octet.

The SCC in transparent mode consists of separate transmit and receive sections whose operations are asynchronous with the CPU core and can either be synchronous or asynchronous with respect to the other SCCs. Each clock can be supplied from the internal baud rate generator bank, DPLL output, or external pins.

**16.14.22.1 FEATURES.** The following is a list of the transparent controller's important features:

- Flexible data buffers
- Automatic SYNC detection on receive
  - 16-bit pattern
  - 8-bit pattern
  - 4-bit pattern
  - External sync pin support
- CRCs can optionally be transmitted and received
- Reverse data mode
- Another protocol can be performed on the SCC other half (transmitter or receiver) during transparent mode
- MC68360-compatible sync options

**16.14.22.2 TRANSPARENT CHANNEL FRAME TRANSMISSION PROCESSING.** The transparent transmitter is designed to work with almost no intervention from the CPU core. When this CPU core enables the SCC transmitter in transparent mode, it starts transmitting idles. The SCC polls the first BD in the transmit channel BD table. When there is a message to transmit, the SCC fetches the data from memory, loads the transmit FIFO, and waits for transmitter synchronization before transmitting the message.

Transmitter synchronization can be achieved using the $\overline{\text{CTS}}$ pin or waiting for the receiver to achieve synchronization, depending on the TXSY bit in the GSMR. Refer to **Section 16.14.22.4 Achieving Synchronization in Transparent Mode** for more details. Once transmitter synchronization is achieved, transmission begins.

When a BD data has been completely transmitted, the last in message (L) bit is checked. If the L-bit is set, the SCC writes the message status bits into the BD and clears the R-bit. It then starts transmitting idles until the next BD is ready. Even if the next BD is already ready, some idles are still transmitted. The transmitter only begins transmission again after it achieves synchronization.

When the end of the current BD has been reached and the L-bit is cleared (working in multibuffer mode), only the R-bit is cleared, and the transmitter moves immediately to the next buffer to begin transmission with no gap on the serial line between buffers. Failure to provide the next buffer in time results in a transmit underrun, causing the TXE bit in the transparent event register to be set.

In both cases, an interrupt is issued according to the interrupt (I) bit in the BD. By appropriately setting the I-bit in each BD, interrupts are generated after the transmission of each buffer or a group of buffers. The SCC then proceeds to the next BD in the table. Any whole number of bytes can be transmitted. If the REVD bit in the GSMR is set, each data byte is reversed in it's bit order before transmission and the MSB of each octet is transmitted first.

The user can decrease the latency of the transmitter by decreasing the transmit FIFO size. This option is enabled by the TFL bit in the GSMR. The user, however, should note that this option causes transmitter underruns at higher transmission speeds. An optional CRC can ay be appended to each transparent frame if it is enabled in the Tx BD. The CRC pattern is chosen in the TCRC bits of the GSMR.

**16.14.22.3 TRANSPARENT CHANNEL FRAME RECEPTION PROCESSING.** When the CPU core enables the SCC receiver in transparent mode, it waits to achieve synchronization before receiving data. The receiver can be synchronized to the data by a synchronization pulse or SYNC pattern.

After a buffer is filled, the SCC clears the empty (E) bit in the BD and generates an interrupt if the interrupt (I) bit in the BD is set. It then moves to the next Rx BD in the table and begins moving data to it's associated buffer. If the next buffer is not available as needed, the BSY bit in the transparent event register signifies a busy signal that can generate a maskable interrupt. The receiver reverts to the hunt mode when the ENTER HUNT MODE command is received or an error condition (lack of receive buffers, CD lost, receiver overrun) is recognized. If the REVD bit in the GSMR is set, each data byte is reversed in it's bit order before it is written to memory.

The user can decrease the latency of the receiver by decreasing the receive FIFO width. This option is enabled by the RFW bit in the GSMR. The user, however, should note that this option causes receiver overruns at higher transmission speeds. The receiver always checks the CRC of the received frame, according to the TCRC bits in the GSMR. If a CRC is not required, the resulting errors can be ignored.

**16.14.22.4 ACHIEVING SYNCHRONIZATION IN TRANSPARENT MODE.** Once the SCC transmitter is enabled for transparent operation in the GSMR, the Tx BD is prepared for the SCC and the transmit FIFO has been preloaded by the SDMA channel, another process must occur before data can be transmitted. It is called transmit synchronization.

Similarly, once the SCC receiver is enabled for transparent operation in the GSMR and the Rx BD is made empty for the SCC another process must occur before data can be received and it is called receive synchronization. The synchronization process gives the user bit-level control of when the transmission and reception begins. There are two basic methods for this:

- An inline synchronization pattern
- External synchronization signals

**16.14.22.4.1 Inline Synchronization Pattern.** The transparent channel can be programmed to transmit and receive a synchronization pattern if the SYNL bits in the GSMR are not zero. The pattern is defined in the DSR. The length of the SYNC pattern is defined in the SYNL bits of the GSMR.

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | 4-BIT SYNC | | | | | | | | | | | | | | | |
| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| FIELD | 8-BIT SYNC | | | | | | | | | | | | | | | |
| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| FIELD | 16-BIT SYNC | | | | | | | | | | | | | | | |

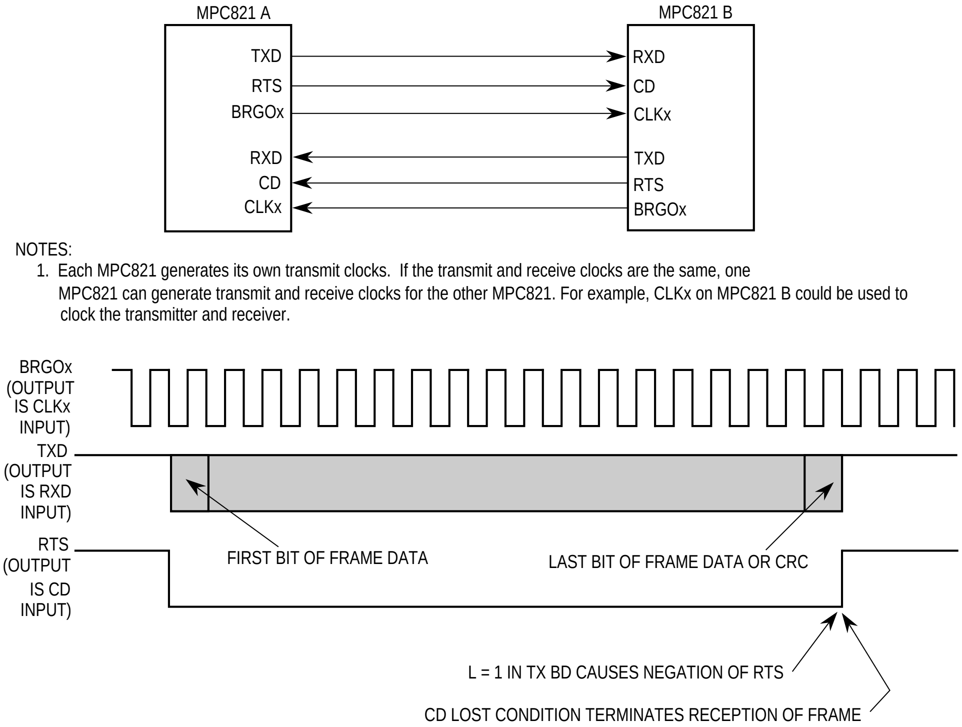
The receiver synchronizes on the synchronization pattern that is located in the DSR. For instance, if a 4-bit SYNC is selected, then reception begins as soon as these four bits are received, beginning with the first bit following the 4-bit SYNC. The transmitter synchronizes on the receiver pattern if the RSYN bit of the GSMR is set. This effectively links the transmitter synchronization to the receiver synchronization.

**16.14.22.4.2 External Synchronization Signals.** If the SYNL bits of the GSMR are programmed to 00, an external signal is used to begin the sequence. The $\overline{\text{CTS}}$ pin is used for the transmitter and the CD pin is used for the receiver and these pins share two options—the pulse and sampling options.

The pulse option determines whether the CD pin or $\overline{\text{CTS}}$ pins need to only be asserted once to begin reception/transmission or whether they must be asserted and stay that way for the duration of the transparent frame. This is controlled by the CDP and CTSP bits of the GSMR. If the user expects a continuous stream of data without interruption, then the pulse operation should be used. However, if the user is trying to identify frames of transparent data, then the envelope mode of the these pins should be used.

The sampling option determines the delay between CD and $\overline{\text{CTS}}$ being asserted and the resulting action by the SCC. These pins can be assumed to be asynchronous to the data and then internally synchronized by the SCC or they can be assumed to be synchronous to the data giving faster operation. This option allows the RTS pin of one SCC to be connected to the CD pin of another SCC (on another MPC821) 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 in **Section 16.14.11 SCC Timing Control**.

**16.14.22.4.3 Transparent Synchronization Example.** Figure 16-94 illustrates an example of synchronization using external signals.


MPC821 A
MPC821 B
TXD
RTS
BRGOx
RXD
CD
CLKx
RXD
CD
CLKx
TXD
RTS
BRGOx
BRGOx (OUTPUT IS CLKx INPUT)
TXD (OUTPUT IS RXD INPUT)
RTS (OUTPUT IS CD INPUT)
FIRST BIT OF FRAME DATA
LAST BIT OF FRAME DATA OR CRC
L = 1 IN TX BD CAUSES NEGATION OF RTS
CD LOST CONDITION TERMINATES RECEPTION OF FRAME

NOTES:

1. Each MPC821 generates its own transmit clocks. If the transmit and receive clocks are the same, one MPC821 can generate transmit and receive clocks for the other MPC821. For example, CLKx on MPC821 B could be used to clock the transmitter and receiver.

NOTES:

1. $\overline{\text{CTS}}$ should be configured as always asserted in the port C parallel I/O or connected to ground externally.
2. 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.
3. The transparent frame contains a CRC if the TC bit is set in the Tx BD.

**Figure 16-94. Sending Transparent Frames Between MPC821s**

MPC821(A) and MPC821(B) exchange transparent frames and synchronize each other using the RTS and CD pins. However, the $\overline{\text{CTS}}$ pin is not required since transmission begins at any time. Thus, the RTS signal is directly connected to the other MPC821 CD pin. The RSYN option in GSMR is not used and transmission and reception from each MPC821 are independent.

**16.14.22.5 TRANSPARENT MEMORY MAP.** When configured to operate in transparent mode, the MPC821 overlays the structure listed in Table 16-23 onto the protocol-specific area of the SCC parameter RAM listed in Table 16-29.

**Table 16-29. Transparent-Specific Parameters**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SCC Base + 30 | CRC_P | Long | CRC Preset for Totally Transparent |
| SCC Base + 34 | CRC_C | Long | CRC Constant for Totally Transparent Receiver |

NOTE: SCC base = IMMR + 1C00 (SCC1) or 1D00 (SCC2).

- CRC_P—For the 16-bit CRC-CCITT, CRC_P should be initialized with $0000FFFF. For the 32-bit CRC-CCITT, CRC_P should be initialized with $FFFFFFFF and for the CRC-16, CRC_P should be initialized with ones ($0000FFFF) or zeros ($00000000).
- CRC_C—For the 16-bit CRC-CCITT, CRC_C should be initialized with $0000F0B8. For the 32-bit CRC-CCITT, CRC_C should be initialized with $DEBB20E3 and for the CRC-16 which is normally used with BISYNC, CRC_C should be initialized with $00000000.

**NOTE**

This value overlaps with the CRC constant (mask) for the HDLC-based protocols. This overlap is not detrimental since the CRC constant (mask) is only used for the receiver; thus, only one entry is required. Therefore, the user can choose HDLC transmitter with a transparent receiver or a transparent transmitter with an HDLC receiver.

**16.14.22.6 COMMAND SET.** The following transmit and receive commands are issued to the CPCR.

**16.14.22.6.1 Transmit Commands.**

**STOP TRANSMIT**

After the hardware or software is reset and the channel is enabled in the SCC mode register, the channel is in the transmit enable mode and starts polling the first BD in the table every 64 clocks (immediately if the TOD bit of the TODR is set).

This command disables the transmission of frames on the transmit channel. If this command is received by the transparent controller during frame transmission, buffer transmission is aborted after a maximum of 64 additional bits and the transmit FIFO is flushed. The TBPTR is not advanced, no new BD is accessed and no new buffers are transmitted for this channel. The transmitter will send idles.

### GRACEFUL STOP TRANSMIT

This command is used to stop transmission smoothly rather than abruptly, as performed by the regular STOP TRANSMIT command. It stops transmission after the current frame finishes transmitting or immediately if there is no frame being transmitted. A transparent frame is not complete until a BD with the L-bit set has it's associated buffer completely transmitted. The GRA bit in the SCCE register is set once transmission stops. After transmission ceases, the transmit parameters, including BDs, can be modified. The TBPTR points to the next Tx BD in the table. Transmission begins once the R-bit of the next BD is set and the RESTART TRANSMIT command is issued.

### RESTART TRANSMIT

This command reenables the transmission of characters on the transmit channel. This command is expected by the transparent controller after a STOP TRANSMIT command, after a STOP TRANSMIT command and disabling the channel in its SCC mode register, after a GRACEFUL STOP TRANSMIT command, or after a transmitter error (underrun or CTS lost). The transparent controller resumes transmission from the current TBPTR in the channel Tx BD table.

### INIT TX PARAMETERS

This command initializes all transmit parameters in this serial channel parameter RAM to their reset state. This command should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the transmit and receive parameters.

### 16.14.22.6.2 Receive Commands.

### ENTER HUNT MODE

After the hardware or software is reset and the channel is enabled in the SCC mode register, the channel is in the receive enable mode and uses the first BD in the table.

This command is used to force the transparent receiver to abort reception of the current frame and enter the hunt mode. In the hunt mode, the transparent controller waits for the synchronization sequence. After receiving the command, the current receive buffer is closed. Further data reception uses the next BD.

### CLOSE Rx BD

This command is used to force the SCC to close the Rx BD if it is currently being used and to use the next BD for any subsequently received data. If the SCC is not in the process of receiving data, no action is taken by this command.

### INIT RX PARAMETERS

This command initializes all the receive parameters in this serial channel parameter RAM to their reset state and should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the receive and transmit parameters.

**16.14.22.7 TRANSPARENT ERROR-HANDLING PROCEDURE.** The SCC reports message reception and transmission error conditions using the channel BDs, the error counters, and the SCC event register.

#### 16.14.22.7.1 Transmission Errors.

**Transmitter Underrun**

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the UN bit of the BD, and generates the TXE interrupt if it is enabled. The channel resumes transmission after the RESTART TRANSMIT command is received. Underrun occurs after a transmit frame for which the L-bit of the Tx BD was not set. In this case, only the TXE bit is set. Underrun cannot occur between transparent frames.

**CTS Lost During Message Transmission**

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the CT bit of the BD, and generates the TXE interrupt if it is enabled. The channel resumes transmission after the RESTART TRANSMIT command is received.

#### 16.14.22.7.2 Reception Errors.

**Overrun Error**

The SCC maintains an internal FIFO for receiving data. The CP begins programming the SDMA channel (if the data buffer is in external memory) and updating the CRC when 8 or 32 bits (according to the RFW bit of the GSMR) are received in the FIFO. If a FIFO overrun occurs, the SCC writes the received data byte to the internal FIFO over the previously received byte. The previous character and it's status bits are lost. Following this, the channel closes the buffer, sets the OV bit of the BD, and generates the RX interrupt if it is enabled. The receiver then enters hunt mode immediately.

**CD Lost During Message Reception**

When this error occurs, the channel terminates message reception, closes the buffer, sets the CD bit of the BD, and generates the RX interrupt if it is enabled. This error has the highest priority, the rest of the message is lost, and no other errors are checked in the message. The receiver then enters hunt mode immediately.

**16.14.22.8 TRANSPARENT MODE REGISTER.** The PSMR is called the transparent mode register when an SCC is programmed for transparent mode. However, since all transparent mode selections are in the GSMR, this register is not used by the transparent controller. If transparent mode is only selected for the transmitter/receiver, then the transmitter/receiver can be programmed to support another protocol. In such a case, the PSMR can be used for the other protocol.

**16.14.22.9 TRANSPARENT RECEIVE BUFFER DESCRIPTOR.** The CP reports information about the received data for each buffer using an Rx BD and closes the current buffer, generates a maskable interrupt, and starts receiving data into the next buffer after one of the following events occurs:

1. An error is detected.
2. A full receive buffer id detected.
3. The ENTER HUNT MODE command is issued.
4. The CLOSE Rx BD command is issued.

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **E** | RES | **W** | **I** | **L** | F | **CM** | RES | DE | RES | | NO | RES | CR | OV | CD |
| **OFFSET + 2** | DATA LENGTH | | | | | | | | | | | | | | | |
| **OFFSET + 4** | RX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

E—Empty

0 = The data buffer associated with this Rx BD has been filled with received data or data reception has been aborted due to an error condition. The CPU core is free to examine or write to any fields of this Rx BD. The CP does not use this BD as long as the E-bit is zero.

1 = The data buffer associated with this BD is empty or reception is currently in progress. This Rx BD and it's associated receive buffer are owned by the CP. Once the E-bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Rx BD table.

1 = This is the last BD in the Rx BD 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 Rx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is used.

1 = When this buffer is closed by the transparent controller, the RX bit in the transparent event register is set. The RX bit can cause an interrupt if it is enabled.

L—Last in Frame

This bit is set by the transparent controller when this buffer is the last in a frame. This implies the negation of CD in envelope mode or the reception of an error, in which case one or more of the OV, CD, and DE bits are set. The transparent controller writes the number of frame octets to the data length field.

0 = This buffer is not the last one in a frame.
1 = This buffer is the last one in a frame.

F—First in Frame

This bit is set be the transparent controller when this buffer is the first in the frame:

0 = The buffer is not the first in a frame.
1 = The buffer is the first in a frame.

CM—Continuous Mode

0 = Normal operation.
1 = The E-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this BD. However, the E-bit is cleared if an error occurs during reception, regardless of the CM bit.

Bit 7—Reserved

DE—DPLL Error

This bit is set by the transparent controller when a DPLL error occurs while this buffer is received. In decoding modes, where a transition is promised every bit, the DPLL error is set when a missing transition occurs.

Bits 9–10—Reserved

NO—Rx Non-Octet

A frame that contained a number of bits not exactly divisible by eight was received.

Bit 12—Reserved

CR—CRC Error indication bits

This frame contains a CRC error. The received CRC bytes are always written to the receive buffer.

OV—Overrun

A receiver overrun occurs during buffer reception.

CD—Carrier Detect Lost

The carrier detect signal is negated during buffer reception.

Data Length

The data length is the number of octets that the CP writes into this BD data buffer. It is only written once by the CP as the buffer is closed.

**NOTE**

> The actual amount of memory allocated for this buffer should be greater than or equal to MRBLR.

Rx Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, must be divisible by four (unless the RFW bit in the GSMR is set to 8-bits wide, even or odd). The buffer can reside in internal or external memory.

**16.14.22.10 TRANSPARENT TRANSMIT BUFFER DESCRIPTOR.** Data is presented to the CP for transmission on an SCC channel by arranging it in buffers referenced by the channel Tx BD table. The CP confirms transmission or indicates error conditions using the BDs to inform the processor that the buffers have been serviced. The status and control bits are prepared by the user before transmission and are set by the CP after the buffer has been transmitted.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **R** | RES | **W** | **I** | **L** | **TC** | **CM** | RESERVED | | | | | | | UN | CT |
| OFFSET + 2 | **DATA LENGTH** | | | | | | | | | | | | | | | |
| OFFSET + 4 | **TX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or it's associated data buffer. The CP clears this bit after the buffer is transmitted or an error condition is encountered.

1 = The data buffer, which has been prepared for transmission by the user, is not transmitted yet or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Tx BD table.

1 = This is the last BD in the Tx BD table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to in the table. The number of Tx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.
1 = When this buffer is serviced by the CP, TX, or TXE bit in the transparent event register is set. The TX and TXE bits can cause interrupts if they are enabled.

L—Last in Message

0 = The last byte in the buffer is not the last byte in the transmitted transparent frame. Data from the next transmit buffer (if ready) is transmitted immediately following the last byte of this buffer.
1 = The last byte in the buffer is the last byte in the transmitted transparent frame. After this buffer is transmitted, the transmitter requires synchronization before the next buffer is transmitted.

TC—Transmit CRC

0 = No CRC sequence is transmitted after this buffer.
1 = A frame check sequence as defined by the TCRC bits in the GSMR is transmitted after the last byte of this buffer.

CM—Continuous Mode

0 = Normal operation.
1 = The R-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this BD. However, the R-bit is cleared if an error occurs during transmission, regardless of the CM bit.

Bits 7–13—Reserved

UN—Underrun

The SCC encountered a transmitter underrun condition while transmitting the associated data buffer.

CT—CTS Lost

CTS is lost during frame transmission.

Data Length

The data length is the number of bytes that the CP should transmit from this BD data buffer. It should be greater than zero and can be even or odd. This value is never modified by the CP.

Tx Data Buffer Pointer

The transmit buffer pointer, which always points to the first byte of the associated data buffer, can be even or odd. The buffer can reside in internal or external memory.

**16.14.22.11 TRANSPARENT EVENT REGISTER.** The SCCE is called the transparent event register when the SCC is operating as a transparent controller. It is a 16-bit register used to report events recognized by the transparent channel and to generate interrupts. On recognition of an event, the transparent controller sets the corresponding bit in the transparent event register. Interrupts generated by this register can be masked in the transparent mask register.

The transparent event register is a memory-mapped register that can be read at any time. A bit is reset by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be reset at a time. All unmasked bits must be reset before the CP negates the internal interrupt request signal. This register is cleared at reset.

SCCE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | GLr | GLt | DCC | RES | | GRA | RES | | TXE | RCH | BSY | TX | RX |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ADDR | A10 (SCCE1), A30 (SCCE2) | | | | | | | | | | | | | | | |

Bits 0–2—Reserved

GLr—Glitch on Rx
A clock glitch is detected by the SCC on the receive clock.

GLt—Glitch on Tx
A clock glitch is detected by the SCC on the transmit clock.

DCC—DPLL CS Changed
The carrier sense status as generated by the DPLL changes state. The real-time status can be found in SCCS. This is not the CD pin status that is reported elsewhere and it is only valid when the DPLL is used.

Bits 6–7—Reserved

GRA—Graceful Stop Complete
A graceful stop, initiated by the GRACEFUL STOP TRANSMIT command, is now complete. This bit is set as soon as the transmitter finishes transmitting any frame that was in progress when the command was issued. It is set immediately if no frame was in progress when the command was issued.

Bits 9–10—Reserved

TXE—Tx Error
An error (CTS lost or underrun) occurs on the transmitter channel.

RCH—Receive Character

A byte or word is received and written to the buffer. This depends on the setting of the RFW bit of the GSMR.

BSY—Busy Condition

A byte/word is received and discarded due to a lack of buffers. The receiver resumes reception after an ENTER HUNT MODE command.

TX—Tx Buffer

A buffer is transmitted. This bit is set no sooner than when the last bit of the last byte of the buffer begins it's transmission, assuming the L-bit of the Tx BD is set. If the L-bit is not set, TX is set when the last byte of data is written to the transmit FIFO.

RX—Rx Buffer

A complete buffer is received on the SCC channel. This bit is set no sooner than two serial clocks after the last bit of the last byte in which the buffer is received on the RXD pin.

**16.14.22.12 TRANSPARENT MASK REGISTER.** The SCCM is referred to as the transparent mask register when the SCC is operating in transparent mode. It is a 16-bit read/write register that has the same bit format as the transparent event register. If a bit in the transparent mask register is a 1, the corresponding interrupt in the event register is enabled. If the bit is zero, the corresponding interrupt in the event register is masked. This register is cleared at reset.

**16.14.22.13 SCC STATUS REGISTER.** The SCCS is an 8-bit read-only register that allows the user to monitor real-time status conditions on the RXD line. The real-time status of the $\overline{\text{CTS}}$ and CD pins are part of the port C parallel I/O.

SCCS REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | CS | RES |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R | R |
| ADDR | A17 (SCCS1), A37 (SCCS2) | | | | | | | |

Bits 0–5—Reserved

CS—Carrier Sense (DPLL)

This bit shows the real-time carrier sense of the line as determined by the DPLL (if it is used).

0 = The DPLL does not sense a carrier.
1 = The DPLL senses a carrier.

Bit 7—Reserved

**16.14.22.14 SCC TRANSPARENT EXAMPLE.** The following list is an initialization sequence for an SCC transparent channel. The transmitter and receiver are both enabled, but operate independently of each other. They implement the connection illustrated on MPC821(B) in Figure 16-94.

Both transmit and receive clocks are provided externally to MPC821(B) using the CLK3 pin. SCC2 is used. The transparent controller is configured with the RTS2 and CD2 pins active. CTS2 is grounded internally by the configuration in port C. A 16-bit CRC-CCITT is sent with each transparent frame. The FIFOs are configured for fast operation.

1. Configure the port A pins to enable the TXD2 and RXD2 pins. Write PAPAR bits 13 and 12 with ones and then PADIR and PAODR bits 13 and 12 with zeros.
2. Configure the port C pins to enable RTS2, CTS2, and CD2. Write PCPAR bit 14 with one and bits 8 and 9 with zero, PCDIR bits 14, 9, and 8 with zero, and PCSO bits 8 and 9 with one.
3. Configure port A to enable the CLK3 pin. Write PAPAR bit 5 with a one and PADIR bit 5 with a zero.
4. Connect the CLK3 pin to SCC2 using the SI. Write the R2CS and T2CS bits of the SICR to 110.
5. Connect the SCC2 to the NMSI (it’s own set of pins) and clear the SC2 bit of the SICR.
6. Write $0001 to SDCR to initialize the SDMA Configuration Register.
7. Write RBASE and TBASE in the SCC parameter RAM to point to the Rx BD and Tx BD in the dual-port RAM. Assuming one Rx BD at the beginning of dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.
8. Program the CPCR to execute the INIT RX and TX PARAMS commands for the SCC. To execute this command for SCC2, write $0041 to the CR.
9. Write RFCR and TFCR with $18 for normal operation.
10. Write MRBLR with the maximum number of bytes per receive buffer and assume 16-bytes, so MRBLR = $0010.
11. Write CRC_P with $0000FFFF to comply with the 16-bit CRC-CCITT.
12. Write CRC_C with $0000F0B8 to comply with the 16-bit CRC-CCITT.
13. Initialize the Rx BD. Assume the Rx data buffer is at $00001000 in main memory. Write $B000 to Rx_BD_Status, $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
14. Initialize the Tx BD. Assume the Tx data buffer is at $00002000 in main memory and contains five 8-bit characters. Write $BC00 to Tx_BD_Status, $0005 to Tx_BD_Length, and $00002000 to Tx_BD_Pointer.
15. Write $FFFF to the SCCE register to clear any previous events.
16. Write $0013 to the SCCM register to enable the TXE, TX, and RX interrupts.
17. Write $20000000 to the CIMR so the SCC2 can generate a system interrupt. The CICR should also be initialized.
18. Write $00001980 to the GSMR_H2 register to configure the transparent channel.

19. Write $00000000 to the GSMR_L2 register to configure the $\overline{\text{CTS}}$ and CD pins to automatically control transmission and reception (DIAG bits). Normal operation of the transmit clock is used (TCI is cleared). Notice that the transmitter (ENT) and receiver (ENR) are not enabled yet.
20. Write $00000030 to the GSMR_L2 register to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

**NOTE**

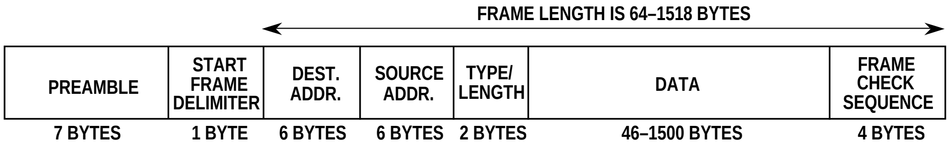
> After 5 bytes are transmitted, the Tx BD is closed. The receive buffer is closed after 16 bytes are received. Any received data beyond 16 bytes caused a busy (out-of-buffers) condition since only one Rx BD id prepared.

### 16.14.23 RAM Microcodes

Additional protocols can be added in the future through the use of RAM microcodes.

### 16.14.24 Ethernet Controller

The Ethernet IEEE 802.3 protocol is a widely used LAN based on the carrier sense multiple access/collision detect (CSMA/CD) approach. Ethernet and IEEE 802.3 frames are very similar and able to coexist on the same LAN. The two protocols are referred to synonymously as Ethernet in this manual, unless specifically noted. Ethernet/IEEE 802.3 frames are based on the frame structure illustrated in Figure 16-95.



NOTE: The LSB of each octet is transmitted first.

**Figure 16-95. Ethernet Frame Structure**

The frame begins with a 7-byte preamble of alternating ones and zeros. Since the frame is Manchester encoded, the preamble gives receiving stations a known pattern on which to lock. The start frame delimiter, which signifies the beginning of the frame, follows the preamble. The 48-bit destination address is next, followed by the 48-bit source address. Original versions of the IEEE 802.3 specification allowed 16-bit addressing; however, this addressing has never been widely used in the industry.

The next field is the type field in Ethernet and the length field in IEEE 802.3. The type field signifies the protocol used in the rest of the frame (TCP/IP) and the length field specifies the length of the data portion of the frame. For Ethernet and IEEE 802.3 frames to coexist on the same LAN, the length field of the frame must always be unique from any type fields used in Ethernet. This has limited the length of the data portion of the frame to 1,500 bytes, and therefore the total frame length to 1,518 bytes. The last 4 bytes of the frame are the FCS, which is the standard 32-bit CCITT-CRC polynomial used in many other protocols.

When a station needs to transmit, it checks for activity on the LAN and when the LAN becomes silent for a specified period, the station starts transmitting. During transmission, the station continually checks for collision on the LAN and if one is detected, the station forces a jam of all ones on it's frame and stops transmitting. Collisions usually occur close to the beginning of a frame. The station then waits a random period of time (backoff) before attempting to transmit again. Once the backoff is complete, the station waits for silence on the LAN and then begins retransmission on the LAN. This is called a retry. If the frame is not successfully transmitted within 15 retries, then an error is indicated. The Ethernet of 10 Mbps gives 0.8 µs per byte. The preamble plus start frame delimiter is transmitted in 6.4 µs. The minimum interframe gap is 9.6 µs and the slot time is 52 µs.

**16.14.24.1 ETHERNET ON THE MPC821.** When the MODE bits of the SCC GSMR select the Ethernet protocol, then that SCC performs the full set of IEEE 802.3/Ethernet CSMA/CD media access control and channel interface functions. Refer to Figure 16-96 for details.

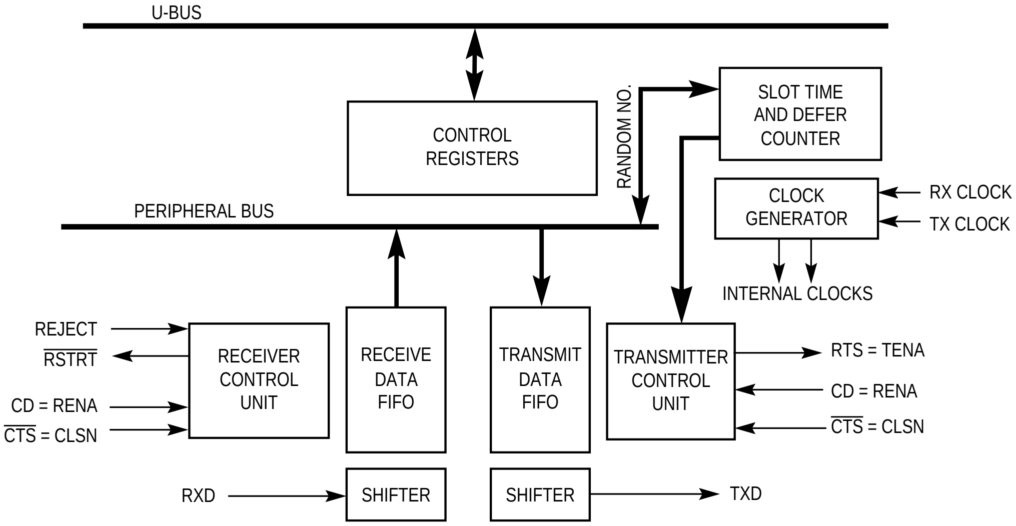


**Figure 16-96. Ethernet Block Diagram**

The MPC821 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 MPC821+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 MPC821, Manchester encoding and decoding, automatic selection of 10BASE-T versus AUI ports, 10BASE-T polarity detection and correction, LED drivers, and a low-power mode. For more information, refer to the MC68160 device description.

The MPC821 Ethernet controller provides a number of features. Although the MPC821 contains DPLLs that allow Manchester encoding and decoding, these DPLLs were not designed for Ethernet rates. Therefore, the Ethernet controller on the MPC821 bypasses the on-chip DPLLs and uses the external SIA on the EEST instead.

**16.14.24.2 FEATURES.** The following is a list of Ethernet's important features:

- Performs MAC layer functions of Ethernet and IEEE 802.3
- Performs framing functions
  - Preamble generation and stripping
  - Destination address checking
  - CRC generation and checking
  - Automatic "short frames" padding 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 "attempt limit" is reached)
  - Automatic discard of incoming collided frames
  - Delay transmission of new frames for specified interframe gap
- Bit rates up to 10 Mbps
- Receives back-to-back frames
- 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 reject pin asserted
- External CAM support on both serial and system bus interfaces
- Up to eight parallel I/O pins can be sampled and appended to any frame
- Heartbeat indication

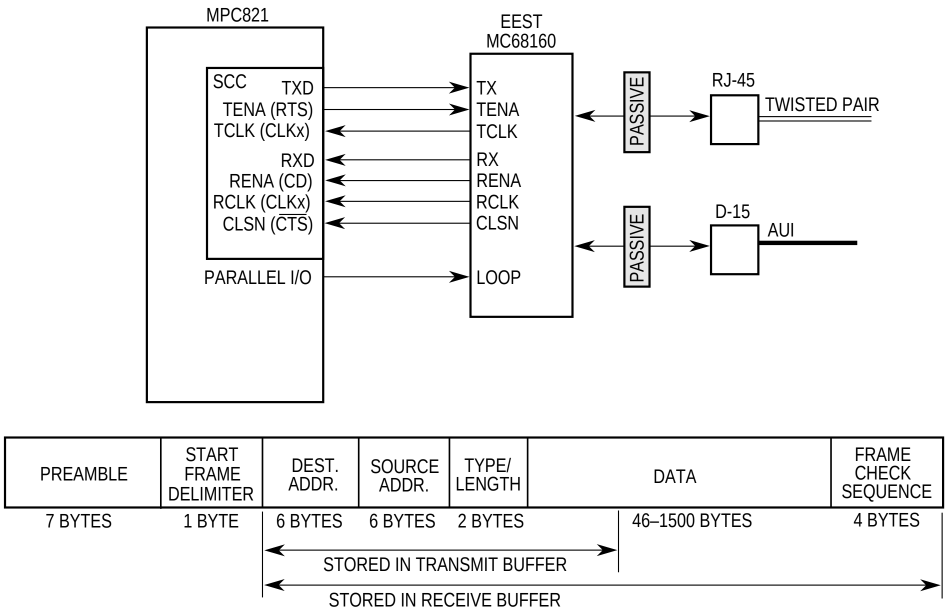
- Transmitter network management and diagnostics
  - Lost carrier sense
  - Underrun
  - Number of collisions exceeded the maximum allowed
  - Number of retries per frame
  - Deferred frame indication
  - Late collision
- Receiver network management and diagnostics
  - CRC error indication
  - Nonoctet 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

**16.14.24.3 LEARNING ETHERNET ON THE MPC821.** The following paragraphs detail the Ethernet functionality on the MPC821. However, they show the additions made to the standard SCC functionality to implement Ethernet. Therefore, the reader is encouraged to learn the basics of the SCCs and the overall architecture of the CPM before attempting to learn this section in great detail. It is important that first-time users of the MPC821 who plan to use Ethernet read the following sections of this manual first.

1. **Section 16.4 RISC Microcontroller**, **Section 16.5 Command Set** and **Section 16.6 Dual-Port RAM**. These sections discuss how the RISC controller is issues special commands to the Ethernet channel. The dual-port RAM loads Ethernet parameters and initialize BDs for the Ethernet channel to use.
2. **Section 16.10 SDMA Channels** discusses how SDMA channels are used to transfer data to/from the Ethernet channel and system memory.
3. **Section 16.12.8 NMSI Configuration** explains how clocks are routed to the SCCs through the bank of clocks.
4. **Section 16.14.1 Overview** contains more detailed information on the SCCs that apply to all protocols. The reader does not need to read the SCC DPLL description since the on-chip DPLLs are not used in Ethernet.
5. **Section 16.14.24 Ethernet Controller** should be read next.
6. **Section 16.19 Parallel I/O Ports** shows how to configure the preferred Ethernet pin functions to be active.
7. **Section 16.20 CPM Interrupt Controller** defines the interrupt priority of this SCC and how interrupts are generated to the CPU core.

**16.14.24.4 CONNECTING THE MPC821 TO ETHERNET.** Figure 16-97 illustrates the basic components and pins required to make the Ethernet connection between the MPC821 and EEST. The MPC821 Ethernet controller has seven basic pins that make up the interface to the external EEST chip:

- Receive clock (RCLK)—The CLK1, CLK2, CLK3, or CLK4 pin that is routed through the bank of clocks on the MPC821.
- Transmit clock (TCLK)—The CLK1, CLK2, CLK3, or CLK4 pin that is routed through the bank of clocks on the MPC821. The SCC RCLK and TCLK should not be connected to the same CLKx pin since the EEST provides a separate receive and transmit clock signal.
- Transmit data (TXD)—The MPC821 TXD pin.
- Receive data (RXD)—The MPC821 RXD pin.



| PREAMBLE | START FRAME DELIMITER | DEST. ADDR. | SOURCE ADDR. | TYPE/ LENGTH | DATA | FRAME CHECK SEQUENCE |
|---|---|---|---|---|---|---|
| 7 BYTES | 1 BYTE | 6 BYTES | 6 BYTES | 2 BYTES | 46–1500 BYTES | 4 BYTES |

NOTE: Short transmit frames are padded automatically by the MPC821.

**Figure 16-97. Connecting the MPC821 to Ethernet**

The following pins take on new meanings when the Ethernet protocol is selected for the SCC:

Transmit Enable (TENA)—The MPC821 RTS pin changes to become TENA when the SCC is configured for Ethernet operation. The polarity of TENA is active high, whereas the polarity of RTS is active low.

Receive Enable (RENA)—The SCC CD pin changes to become RENA when the SCC is configured for Ethernet operation. The polarity of RENA is active high, whereas the polarity of CD is active low.

Collision (CLSN)—The SCC $\overline{\text{CTS}}$ pin changes to become CLSN when the SCC is configured for Ethernet operation. The polarity of CLSN is active high, whereas the polarity of $\overline{\text{CTS}}$ is active low.

**NOTE**

The carrier sense signal is often referred to in Ethernet descriptions because it defines whether the LAN is currently in use. Carrier sense is defined as RENA OR’ed with CLSN.

The EEST has similar names for its connection to the seven basic MPC821 pins and contains a loopback pin so the MPC821 can perform external loopback testing. This can be controlled by any available parallel I/O pin on the MPC821. The MPC821 has additional pins for interfacing with an optional external content-addressable memory (CAM) and they are described in more detail in **Section 16.14.24.7 CAM Interface**. The passive components (principally transformers) required to connect to AUI or twisted-pair media are external to the EEST. For more information on the EEST connection circuits, refer to the MC68160 device description.

The MPC821 stores every byte received after the start frame delimiter into system memory, using the SDMA channels. On transmit, the user provides the destination address, source address, type/length field, and the transmit data. The MPC821 automatically pads frames that have less than 46 bytes in the data field to meet the minimum frame requirements. In addition, the MPC821 appends the FCS to the frame.

**16.14.24.5 ETHERNET CHANNEL FRAME TRANSMISSION.** The Ethernet transmitter is designed to work with almost no intervention from the host. When the host enables the transmitter, the Ethernet controller polls the first Tx BD in the channel Tx BD table and the poll occurs every 128 serial clocks. If the user has a frame ready to transmit, the TOD bit in the TODR can be set to avoid waiting for the next poll to occur.

When there is a frame to transmit, the Ethernet controller begins fetching the data from the data buffer, asserts TENA to the EEST, and starts transmitting the preamble sequence, the start frame delimiter, and then the frame information. However, the controller defers the transmission if the line is busy (carrier sense is active). Before transmitting, the controller waits for carrier sense to become inactive and once carrier sense is inactive, the controller determines if carrier sense stays inactive for 6.0 µs. If so, then the transmission begins after waiting an additional 3.6 µs (9.6 µs after carrier sense originally became inactive).

If a collision occurs during the transmit frame, the Ethernet controller follows the specified backoff procedures and attempts 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 (8 bytes if the transmit frame was word aligned) in internal RAM, so that they do not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency in the case that the backoff timer output results in a need for an immediate retransmission. If a collision occurs during the transmission of the frame, the Ethernet controller returns to the first buffer for a retransmission. The only restriction is that the first buffer must contain at least 9 bytes.

**NOTE**

If an Ethernet frame is made up of multiple buffers, the user should not reuse the first buffer descriptor until the last buffer descriptor of the frame has had it's ready bit cleared by the CPM.

When the end of the current BD is reached and the L-bit in the Tx BD is set, the FCS (32-bit CRC) bytes are appended (if the TC bit is set in the Tx BD), and TENA is negated. This notifies the EEST of the need to generate the illegal Manchester encoding that signifies the end of an Ethernet frame. Following the transmission of the CRC, 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 (a frame is comprised of multiple buffers), only the R-bit is cleared.

In either mode, an interrupt can be issued according to the I-bit in the Tx BD. The Ethernet controller then proceeds to the next Tx BD in the table. In this way, the user can be interrupted after each frame, after each buffer, or after a specific buffer is transmitted. The Ethernet controller has an option to add pad characters to short frames. If the PAD bit is set in the Tx BD, the frame is padded up to the value of the minimum frame length register.

To rearrange the transmit queue before the CP has completed transmission of all frames, issue the GRACEFUL STOP TRANSMIT command. This technique can be useful for transmitting expedited data before previously linked buffers or for error situations. When the GRACEFUL STOP TRANSMIT command is issued, the Ethernet controller stops immediately if no transmission is in progress or continues transmission until the current frame either finishes or terminates with a collision. When the Ethernet controller is given the RESTART TRANSMIT command, it resumes transmission. The Ethernet controller transmits bytes LSB first.

**16.14.24.6 ETHERNET CHANNEL FRAME RECEPTION.** The Ethernet receiver is designed to work with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, maximum DMA transfer checking, and maximum frame length checking.

When the host enables the Ethernet receiver, it enters hunt mode as soon as the RENA signal is asserted if CLSN is negated. In hunt mode, as data is shifted into the receive shift register one bit at a time, the contents of the register are compared to the contents of the SYN1 field in the data synchronization register. This compare function becomes valid a certain number of clocks after the start of the frame (depending on the NIB bits in the PSMR). If the two are not equal, the next bit is shifted in, and the comparison is repeated. If a double zero fault or double one fault is detected between bits 14 to 21 from the start of the frame, the frame is rejected. If a double zero fault is detected after 21 bits from the start of the frame and before detection the start frame delimiter, the frame is also rejected. When the registers match, the hunt mode is terminated and character assembly begins.

When the receiver detects the first bytes of the frame, the Ethernet controller performs address recognition functions on the frame (refer to **Section 16.14.24.11 Ethernet Address Recognition** for more information). The receiver can receive physical (individual), group (multicast), and broadcast addresses. Ethernet receive frame data is not written to memory until the internal address recognition algorithm is complete, which improves bus utilization in the case of frames not addressed to this station. The receiver also operates with an external CAM. In the case of an external CAM, frame reception continues normally, unless the CAM specifically signals the frame to be rejected. Refer to **Section 16.14.24.7 CAM Interface** for more details.

If a match is detected, the Ethernet controller fetches the next Rx BD and, if it is empty, starts transferring the incoming frame to the Rx BD associated data buffer. If a collision is detected during the frame, the Rx BDs associated with this frame are reused. Thus, no collision frames are presented to the user except late collisions, which indicate serious LAN problems. When the data buffer has been filled, the Ethernet controller clears the E-bit in the Rx BD and generates an interrupt if the I-bit is set. If the incoming frame exceeds the length of the data buffer, the Ethernet controller fetches the next Rx BD in the table and, if it is empty, continues transferring the rest of the frame to this BD associated data buffer. The Rx BD length is determined in the MRBLR value in the SCC general-purpose parameter RAM. The user should program MRBLR to be at least 64 bytes.

During reception, the Ethernet controller checks for a frame that is either too short or too long. When the frame ends (carrier sense 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 and it enables the software to correctly recognize the frame-too-long condition.

When the receive frame is complete, the Ethernet controller has the option to sample one byte from the port B parallel I/O (PB23–PB16) and append this byte to the end of the last Rx BD in the frame. For any of the PB23–PB16 pins that are defined as outputs, the contents of the PBDAT latch is read, rather than the pin itself. Although this capability is useful for CAM applications, it can be used whether or not an external CAM is present. The sampling occurs at the end of frame reception.

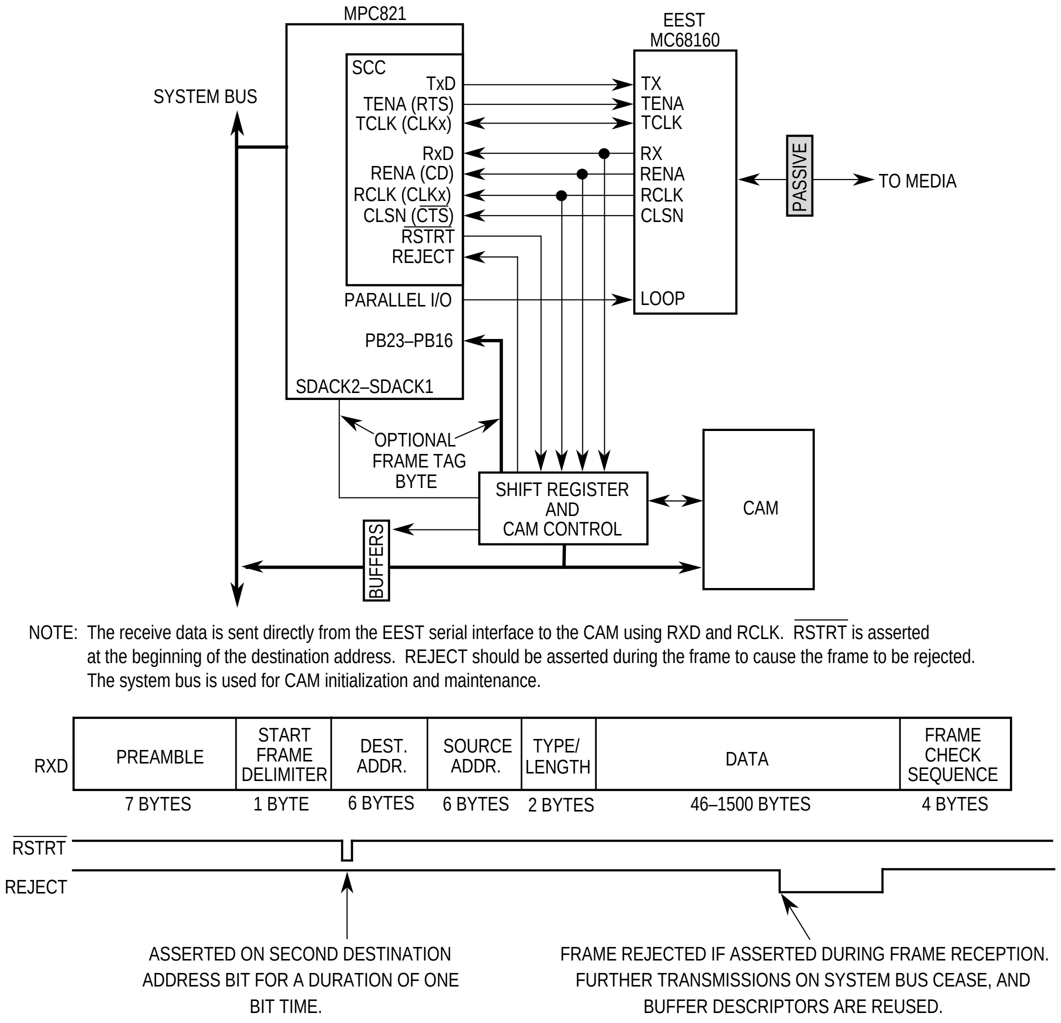
The Ethernet controller then sets the L-bit in the Rx BD, writes the other frame status bits into the Rx BD, and clears the E-bit. The Ethernet controller next generates a maskable interrupt, indicating that a frame has been received and is in memory. The Ethernet controller then waits for a new frame. The Ethernet controller receives serial data LSB first.

**16.14.24.7 CAM INTERFACE.** The Ethernet controller has two options for connecting to an external CAM—a serial interface or a system bus interface. Actually, both options can be used at the same time (there is no mode bit to select them), but they are described independently for clarity. To implement an option, the user only needs to enable the particular pins that are preferable. Both options use a reject pin on the MPC821 to signify that the current frame should be discarded. The MPC821 internal address recognition logic can be used in combination with an external CAM. Refer to **Section 16.14.24.11 Ethernet Address Recognition** for more details.

The serial interface option is illustrated in Figure 16-98. The MPC821 outputs a receive start ($\overline{\text{RSTRT}}$) signal when the start frame delimiter is recognized. This signal is asserted for one bit time on the second destination address bit. The CAM control logic uses $\overline{\text{RSTRT}}$ (in combination with the RXD and RCLK signals) to store the destination or source address and generate writes to the CAM for address recognition. In addition, the RENA signal supplied from the EEST can be used to abort the comparison if a collision occurs on the receive frame.

After the comparison occurs, the CAM control logic asserts the receive reject (REJECT) pin, if the current receive frame is rejected. The MPC821 Ethernet controller then immediately stops writing data to system memory and reuses the buffer(s) for the next frame. If the CAM accepts the frame, the CAM control logic does nothing (REJECT is not asserted). However, if REJECT is asserted, it must be done prior to the end of the receive frame.

Additionally, the CAM control logic might want to provide additional information on the PB23–PB16 pins. The MPC821 Ethernet controller writes this additional byte to memory during the last SDMA write if the SIP bit is set in the PSMR. This information tag is sampled by the MPC821 Ethernet controller as the last FCS byte is read from the receive FIFO. The information TAG should be provided by the CAM control logic no later than when RENA is negated at the end of a noncollision frame, and should be held stable on the PB23–PB16 pins until the SDACK1–SDACK2 pins signal that the tag byte is being written to memory.



NOTE: The receive data is sent directly from the EEST serial interface to the CAM using RXD and RCLK. RSTRT is asserted at the beginning of the destination address. REJECT should be asserted during the frame to cause the frame to be rejected. The system bus is used for CAM initialization and maintenance.

| PREAMBLE | START FRAME DELIMITER | DEST. ADDR. | SOURCE ADDR. | TYPE/ LENGTH | DATA | FRAME CHECK SEQUENCE |
|---|---|---|---|---|---|---|
| 7 BYTES | 1 BYTE | 6 BYTES | 6 BYTES | 2 BYTES | 46–1500 BYTES | 4 BYTES |

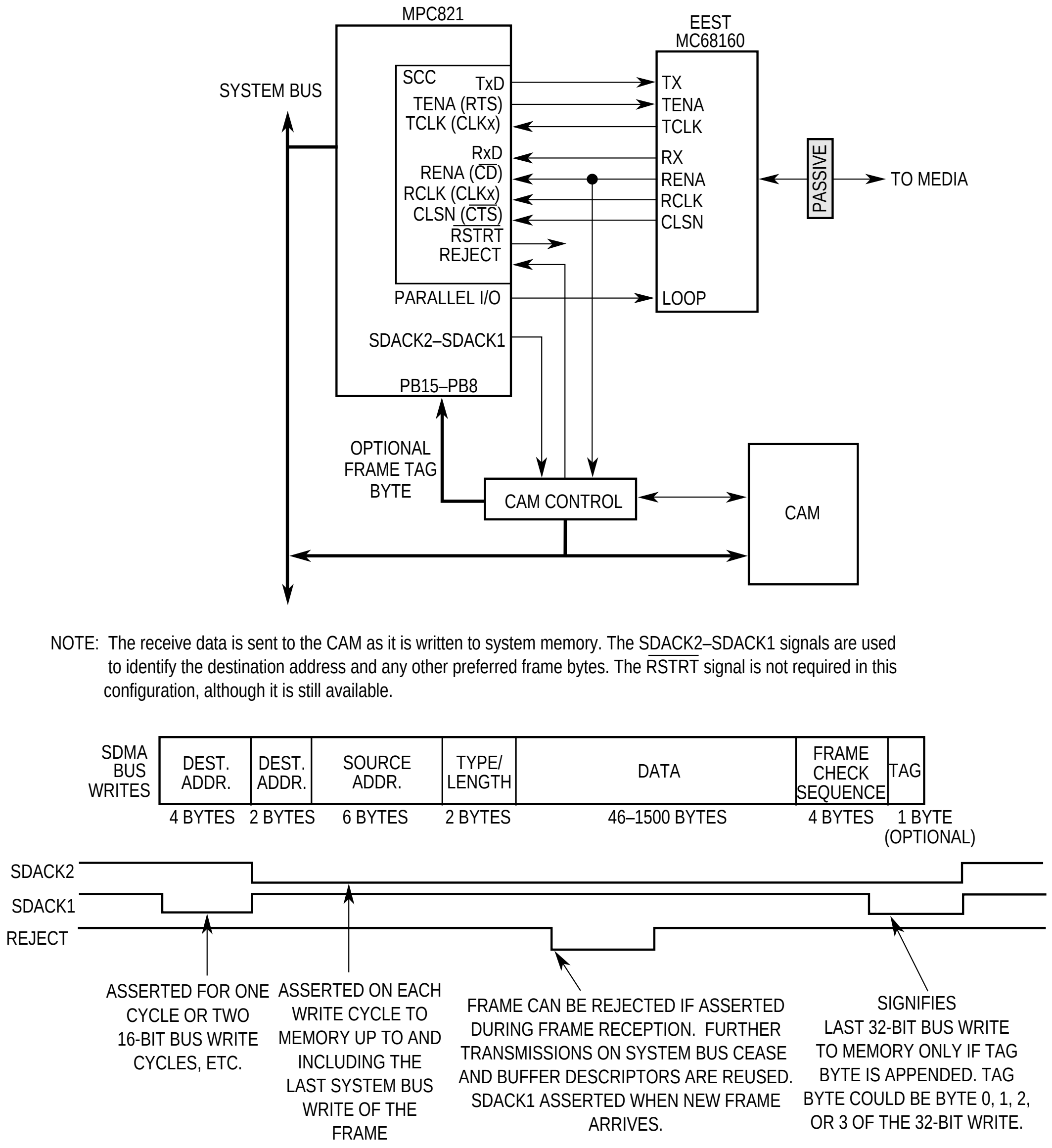
ASSERTED ON SECOND DESTINATION ADDRESS BIT FOR A DURATION OF ONE BIT TIME.

FRAME REJECTED IF ASSERTED DURING FRAME RECEPTION. FURTHER TRANSMISSIONS ON SYSTEM BUS CEASE, AND BUFFER DESCRIPTORS ARE REUSED.

**Figure 16-98. MPC821 Ethernet Serial CAM Interface**

The parallel interface option is illustrated in Figure 16-99. The MPC821 outputs two signals every time it writes Ethernet frame data to system memory. The signals SDMA acknowledge (SDACK1–SDACK2), are asserted during all bus cycles on which Ethernet frame data is written to memory. These signals are not used for other protocols.

The CAM control logic uses these pins to enable the CAM writes simultaneously with system memory writes. In this way, the CAM captures the frame data at the same time that it is being written to system memory. The chief advantage of this approach is that the data is already in parallel form when it leaves the MPC821.


MPC821
EEST
MC68160
SYSTEM BUS
SCC
TxD
TENA (RTS)
TCLK (CLKx)
RxD
RENA (CD)
RCLK (CLKx)
CLSN (CTS)
RSTRT
REJECT
PARALLEL I/O
SDACK2–SDACK1
PB15–PB8
TX
TENA
TCLK
RX
RENA
RCLK
CLSN
LOOP
PASSIVE
TO MEDIA
OPTIONAL
FRAME TAG
BYTE
CAM CONTROL
CAM
SDMA
BUS
WRITES
DEST.
ADDR.
DEST.
ADDR.
SOURCE
ADDR.
TYPE/
LENGTH
DATA
FRAME
CHECK
SEQUENCE
TAG
4 BYTES
2 BYTES
6 BYTES
2 BYTES
46–1500 BYTES
4 BYTES
1 BYTE
(OPTIONAL)
SDACK2
SDACK1
REJECT
ASSERTED FOR ONE CYCLE OR TWO 16-BIT BUS WRITE CYCLES, ETC.
ASSERTED ON EACH WRITE CYCLE TO MEMORY UP TO AND INCLUDING THE LAST SYSTEM BUS WRITE OF THE FRAME
FRAME CAN BE REJECTED IF ASSERTED DURING FRAME RECEPTION. FURTHER TRANSMISSIONS ON SYSTEM BUS CEASE AND BUFFER DESCRIPTORS ARE REUSED. SDACK1 ASSERTED WHEN NEW FRAME ARRIVES.
SIGNIFIES LAST 32-BIT BUS WRITE TO MEMORY ONLY IF TAG BYTE IS APPENDED. TAG BYTE COULD BE BYTE 0, 1, 2, OR 3 OF THE 32-BIT WRITE.

NOTE: The receive data is sent to the CAM as it is written to system memory. The SDACK2–SDACK1 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, the SDACK2–1 pins would not be asserted for other bus activity.

**Figure 16-99. MPC821 Ethernet Parallel CAM Interface**

The SDACK1–SDACK2 signals are asserted during all bus cycle writes of the frame data. A certain SDACK1–SDACK2 combination specifically identifies the first 32 bits of the frame, another identifies all mid-frame data, and a third combination 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 PB23–PB16 if the SIP bit is set in the PSMR. The tag byte is always in byte 3 of the last 32-bit write. The Rx BD data length does not include tag byte in the length calculation.

If the system memory is 32 bits, then the MPC821 32-bit write takes one bus cycle. If the system memory is 16 or 8 bits, then the MPC821 32-bit write takes two or four bus cycles. In any case, the SDACK1–SDACK2 signals are valid on each bus cycle of a 32-bit write cycle and only during bus cycles associated with the Ethernet receiver. Additionally, the user can choose a unique address type (AT0–AT3) associated with the SDMA receive channel associated with the Ethernet SCC to have an alternate method of identifying accesses from this SCC.

**NOTE**

> 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 Rx BD data length does not show the length of the tag byte in the frame. Also, the SDACK2-1 signals will equal 00 whenever the frame is not an even multiple of four (it does not depend on whether the tag byte is appended).

**16.14.24.8 ETHERNET MEMORY MAP.** When configured to operate in Ethernet mode, the MPC821 overlays the structure described in Table 16-23 onto the protocol-specific area of the SCC parameter RAM described in Table 16-30.

Table 16-30. Ether et-Specific Parameter RAM

| ADDRESS | NAME | WIDTH | DESCRIPTION | USER WRITES |
|---|---|---|---|---|
| SCC Base + 30 | **C_PRES** | Word | Preset CRC | $FFFFFFFF |
| SCC Base + 34 | **C_MASK** | Word | Constant MASK for CRC | $DEBB20E3 |
| SCC Base + 38 | **CRCEC** | Word | CRC Error Counter | |
| SCC Base + 3C | **ALEC** | Word | Alignment Error Counter | |
| SCC Base + 40 | **DISFC** | Word | Discard Frame Counter | |
| SCC Base + 44 | **PADS** | Half-word | Short Frame PAD character | |
| SCC Base + 46 | **RET_Lim** | Half-word | Retry Limit Threshold | 15 dec |
| SCC Base + 48 | RET_cnt | Half-word | Retry Limit Counter | |
| SCC Base + 4A | **MFLR** | Half-word | Maximum Frame Length Register | 1518 dec |
| SCC Base + 4C | **MINFLR** | Half-word | Minimum Frame Length Register | 64 dec |
| SCC Base + 4E | **MAXD1** | Half-word | Max DMA1 Length Register | 1520 dec |
| SCC Base + 50 | **MAXD2** | Half-word | Max DMA2 Length Register | 1520 dec |
| SCC Base + 52 | MAXD | Half-word | Rx Max DMA | |
| SCC Base + 54 | DMA_cnt | Half-word | Rx DMA Counter | |
| SCC Base + 56 | MAX_b | Half-word | Max BD Byte Count | |
| SCC Base + 58 | **GADDR1** | Half-word | Group Address Filter 1 | |
| SCC Base + 5A | **GADDR2** | Half-word | Group Address Filter 2 | |
| SCC Base + 5C | **GADDR3** | Half-word | Group Address Filter 3 | |
| SCC Base + 5E | **GADDR4** | Half-word | Group Address Filter 4 | |
| SCC Base + 60 | TBUF0.data0 | Word | Save Area 0 - Current Frame | |
| SCC Base + 64 | TBUF0.data1 | Word | Save Area 1 - Current Frame | |
| SCC Base + 68 | TBUF0.rba0 | Word | | |
| SCC Base + 6C | TBUF0.crc | Word | | |
| SCC Base + 70 | TBUF0.bcnt | Half-word | | |
| SCC Base + 72 | **PADDR1_H** | Half-word | Physical Address 1 (MSB) | |
| SCC Base + 74 | **PADDR1_M** | Half-word | Physical Address 1 | |
| SCC Base + 76 | **PADDR1_L** | Half-word | Physical Address 1 (LSB) | |
| SCC Base + 78 | **P_Per** | Half-word | Persistence | |
| SCC Base + 7A | RFBD_ptr | Half-word | Rx First BD Pointer | |

### Table 16-30. Ether et-Specific Parameter RAM (Continued)

| ADDRESS | NAME | WIDTH | DESCRIPTION | USER WRITES |
|---|---|---|---|---|
| SCC Base + 7C | TFBD_ptr | Half-word | Tx First BD Pointer | |
| SCC Base + 7E | TLBD_ptr | Half-word | Tx Last BD Pointer | |
| SCC Base + 80 | TBUF1.data0 | Word | Save Area 0 - Next Frame | |
| SCC Base + 84 | TBUF1.data1 | Word | Save Area 1 - Next Frame | |
| SCC Base + 88 | TBUF1.rba0 | Word | | |
| SCC Base + 8C | TBUF1.crc | Word | | |
| SCC Base + 90 | TBUF1.bcnt | Half-word | | |
| SCC Base + 92 | TX_len | Half-word | Tx Frame Length Counter | |
| SCC Base + 94 | **IADDR1** | Half-word | Individual Address Filter 1 | |
| SCC Base + 96 | **IADDR2** | Half-word | Individual Address Filter 2 | |
| SCC Base + 98 | **IADDR3** | Half-word | Individual Address Filter 3 | |
| SCC Base + 9A | **IADDR4** | Half-word | Individual Address Filter 4 | |
| SCC Base + 9C | BOFF_CNT | Half-word | Backoff Counter | |
| SCC Base + 9E | **TADDR_L** | Half-word | Temp Address (LSB) | |
| SCC Base + A0 | **TADDR_M** | Half-word | Temp Address | |
| SCC Base + A2 | **TADDR_H** | Half-word | Temp Address (MSB) | |

NOTE: Items in bold must be initialized by the user.
SCC base = IMMR + 1C00 (SCC1) or 1D00 (SCC2).

- C_PRES—For the 32-bit CRC-CCITT, C_PRES should be initialized with $FFFFFFFF.
- C_MASK—For the 32-bit CRC-CCITT, C_MASK should be initialized with $DEBB20E3.
- CRCEC, ALEC, and DISFC—These 32-bit (modulo $2^{32}$) counters are maintained by the CP and can be initialized by the user while the channel is disabled. CRCEC is incremented for each received frame with a CRC error, except it does not include frames not addressed to the user, 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 user, frames received in the out-of-buffers condition, or frames with overrun errors. DISFC is incremented for frames discarded because of the out-of-buffers condition or an overrun error. The CRC does not have to be correct for this counter to be incremented.

- PADS—The user writes the pattern of the pad characters into this 16-bit register that should be sent when short frame padding is implemented. The byte pattern written to the register may be of any value, but both the high and low bytes should be the same.
- RET_Lim—The user writes the number of retries that should be made to transmit a frame into this 16-bit register. This value is typically 15 decimal. If the frame is not transmitted after this limit is reached, an interrupt can be generated. RET_cnt is a temporary down-counter used to count the number of retries made.
- MFLR—The Ethernet controller checks the length of an incoming Ethernet frame against the user-defined value given in this 16-bit register. Typically this register is set to 1,518 decimal. If this limit is exceeded, the remainder of the incoming frame is discarded and the LG (Rx frame too long) bit is set in the last Rx BD belonging to that frame. The Ethernet controller reports the frame status and length in the last Rx BD.

  MFLR is defined as all the in-frame bytes between the start frame delimiter and the end of the frame (destination address, source address, length, LLC data, PAD, and FCS). DMA_cnt is a temporary down-counter used to track the frame length.

- MINFLR—The Ethernet controller checks the length of an incoming Ethernet frame against the user-defined value given in this 16-bit register that is typically set to 64 decimal. If the received frame length is less than the register value, then this frame is discarded unless the RSH (receive short frames) bit in the PSMR is set. If RSH is set, then the SH (Rx frame too short) bit is set in the last Rx BD belonging to that frame. For transmit operation, if the frame is too short, the Ethernet controller adds PADs to the transmitted frame (according to the PAD bit in the Tx BD and the PAD value in the parameter RAM). PADs are added to make the transmit frame MINFLR bytes in length.
- MAXD1—This parameter gives the user the ability to stop system bus writes from occurring after a frame has exceeded a certain size. The value of this register is valid only if an address match is detected. The Ethernet controller checks the length of an incoming Ethernet frame against the user-defined value given in this 16-bit register that is usually set to 1,520 decimal. If this limit is exceeded, the remainder of the incoming frame is discarded. The Ethernet controller waits to the end of the frame (or until MFLR bytes have been received) and reports the frame status and the frame length in the last Rx BD.
- MAXD2—This parameter gives the user the ability to stop system bus writes from occurring after a frame has exceeded a certain size. The value of this register is valid in promiscuous mode when no address match is detected. The Ethernet controller checks the length of an incoming Ethernet frame against the user-defined value given in this 16-bit register that is usually set to 1,520 decimal. If this limit is exceeded, the remainder of the incoming frame is discarded. The Ethernet controller waits until the end of the frame (or until MFLR bytes have been received) and reports the frame status and length in the last Rx BD.

  In a monitor station, MAXD2 can be programmed to a value much less than MAXD1 to receive entire frames addressed to this station, but receive only the headers of all the other frames.

- GADDR1–4—These four registers are used in the hash table function of the group addressing mode. The user may 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.
- PADDR1—The user writes the 48-bit individual address of this station into this location. PADDR1_L is the lowest order half-word, and PADDR1_H is the highest order half-word.
- P_Per—This parameter allows the Ethernet controller to be less aggressive in it's behavior after a collision. Normally, this parameter should be set to $0000. To decrease the aggressiveness of the Ethernet controller, P_Per can be set to a value from 1 to 9, with 9 being the least aggressive. The P_Per value is added to the retry count in the backoff algorithm to reduce the probability of transmission on the next time-slot.

**NOTE**

The use of P_Per is fully allowed within the Ethernet/802.3 specifications. In a heavily congested Ethernet LAN, a less aggressive backoff algorithm used by multiple stations on the LAN increases the overall LAN throughput by reducing the probability of collisions. The SBT bit in the PSMR offers another way to reduce the aggressiveness of the Ethernet controller.

### IADDR1–4

These four registers are used in the hash table function of the individual addressing mode. The user can write zeros to these values after reset and before the Ethernet channel is enabled to disable all individual hash address recognition functions. The SET GROUP ADDRESS command is used to enable the hash table.

### TADDR

This parameter allows the user to add and delete addresses from the individual and group hash tables. After placing an address in TADDR, the user then issues the SET GROUP ADDRESS command. TADDR_L is the lowest order half-word and TADDR_H is the highest order half-word.

**16.14.24.9 PROGRAMMING MODEL.** The host configures SCC to operate as an Ethernet controller by the MODE bits in the GSMR. The receive errors (collision, overrun, nonoctet aligned frame, short frame, frame too long, and CRC error) are reported through the Rx BD. The transmit errors (underrun, heartbeat, late collision, retransmission limit, and carrier sense lost) are reported through the Tx BD.

Several bit fields in the GSMR must be programmed to special values for Ethernet. Refer to the GSMR for more details. The user should program the DSR as shown below. The 6 bytes of preamble programmed in the GSMR, in combination with the programming of the DSR shown in the table below, causes 8 bytes of preamble on transmit (including the 1-byte start delimiter with the value $D5).

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | SYN2 = $D5 | | | | | | | | SYN1 = $55 | | | | | | | |
| RESET | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | A0E (DSR1), A2E (DSR2) | | | | | | | | | | | | | | | |

**16.14.24.10 ETHERNET COMMAND SET.** The following transmit and receive commands are issued to the CPCR.

**NOTE**

Before issuing the CP RESET command, configure the TENA (RTS) pin to be an input. Refer to step 3 of **Section 16.14.24.23 SCC Ethernet Example** for more information.

### 16.14.24.10.1 Transmit Commands.

**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**

This command is used to smoothly stop transmission after the current frame finishes transmitting or undergoes a collision (immediately if there is no frame being transmitted). The GRA bit in the SCCE register is set once transmission stops. Then the Ethernet transmit parameters (including BDs) can be modified by the user. The TBPTR points to the next Tx BD in the table. Transmission begins once the R-bit of the next BD is set and the RESTART TRANSMIT command is issued.

### NOTE

If the GRACEFUL STOP TRANSMIT command is issued and the current transmit frame ends in a collision, the TBPTR points to the beginning of the collided frame, with the R-bit still set in the Tx BD (the frame looks as if it was never transmitted).

#### RESTART TRANSMIT

This command enables the transmission of characters on the transmit channel. It is expected by the Ethernet controller after a GRACEFUL STOP TRANSMIT command or transmitter error (underrun, retransmission limit reached, or late collision). The Ethernet controller will resumes transmission from the current TBPTR in the channel Tx BD table.

#### INIT TX PARAMETERS

This command initializes all the transmit parameters in this serial channel parameter RAM to their reset state. This command should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the transmit and receive parameters.

### 16.14.24.10.2 Receive Commands.

#### ENTER HUNT MODE

After the hardware or software is reset and the channel in the SCC mode register is enabled, the channel is in the receive enable mode and uses the first BD in the table.

This command is generally used to force the Ethernet receiver to abort reception of the current frame and enter the hunt mode. In the hunt mode, the Ethernet controller continually scans the input datastream for a transition of carrier sense from inactive to active and then a preamble sequence followed by the start frame delimiter. After receiving the command, the current receive buffer is closed and the CRC calculation is reset. Further frame reception uses the next Rx BD.

#### CLOSE Rx BD

This command should not be used with the Ethernet controller.

#### INIT RX PARAMETERS

This command initializes all the receive parameters in this serial channel parameter RAM to their reset state and should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the receive and transmit parameters.

**SET GROUP ADDRESS**

> The SET GROUP ADDRESS command is used to set a bit in one of the 64 bits of the four individual/group address hash filter registers (GADDR1–4 or IADDR1–4). The individual or group address (48 bits) to be added to the hash table should be written to TADDR_L, TADDR_M, and TADDR_H in the parameter RAM prior to executing this command. The RISC controller checks the I/G bit in the address stored in TADDR to determine whether to use the individual hash table or the group hash table. A zero in the I/G bit implies an individual address and a 1 in the I/G bit implies a group address. This command can be executed at any time, regardless of whether the Ethernet channel is enabled.

If an address from the hash table must be deleted, the Ethernet channel should be disabled, the hash table registers should be cleared, and the SET GROUP ADDRESS command must be executed for the remaining preferred addresses. This is required because the hash table might have mapped multiple addresses to the same hash table bit.

**16.14.24.11 ETHERNET ADDRESS RECOGNITION.** The Ethernet controller can filter the 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 illustrated in Figure 16-100.

In the physical type of address recognition, the Ethernet controller compares the destination address field of the received frame with the physical address that the user programs in the PADDR1. Alternatively, the user can perform address recognition on multiple individual addresses using the IADDR1–4 hash table.

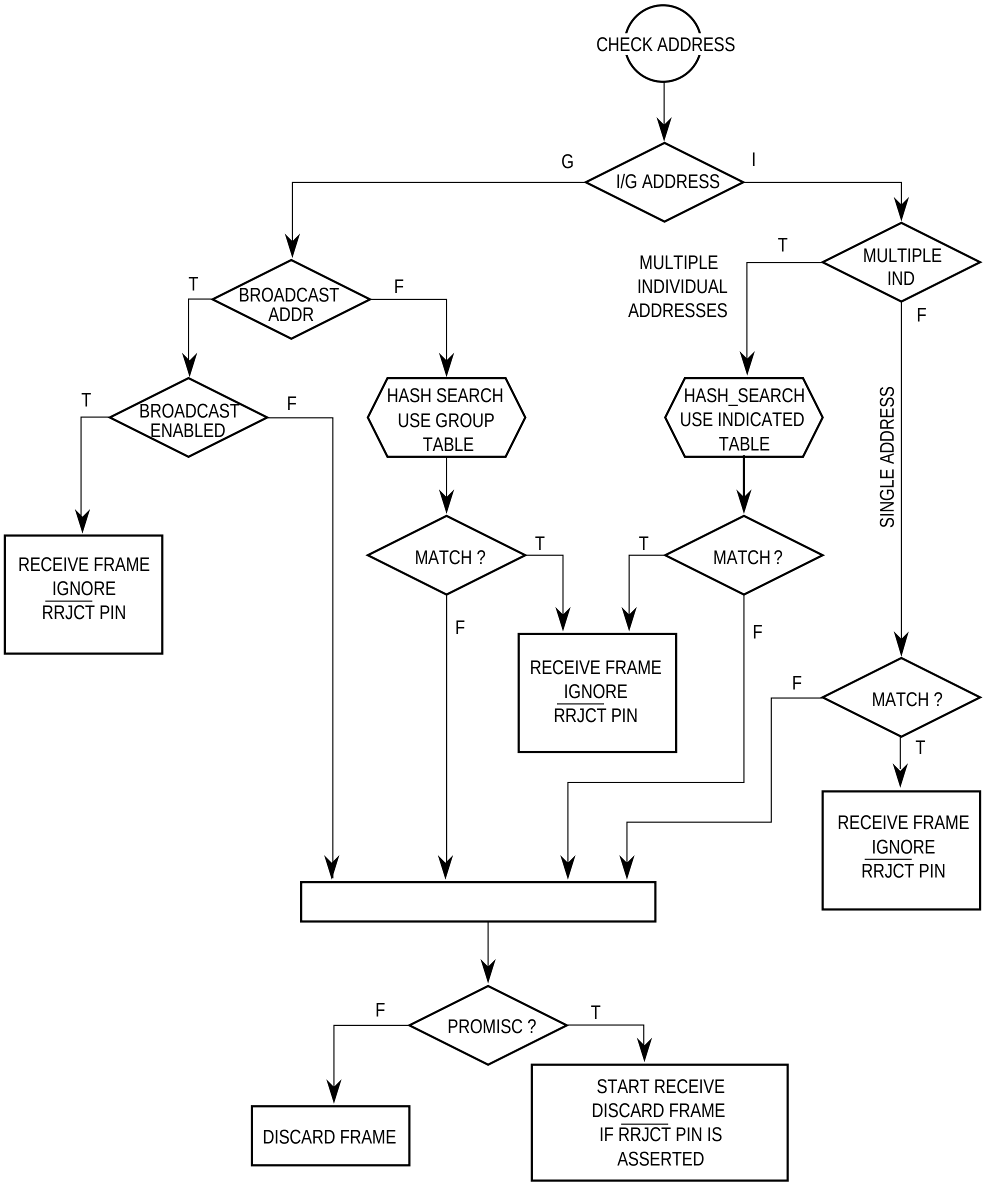

CHECK ADDRESS
I/G ADDRESS
G
I
BROADCAST ADDR
T
F
MULTIPLE IND
T
F
MULTIPLE INDIVIDUAL ADDRESSES
BROADCAST ENABLED
T
F
HASH SEARCH USE GROUP TABLE
HASH_SEARCH USE INDICATED TABLE
SINGLE ADDRESS
RECEIVE FRAME IGNORE RRJCT PIN
MATCH ?
T
F
MATCH ?
T
F
RECEIVE FRAME IGNORE RRJCT PIN
MATCH ?
F
T
RECEIVE FRAME IGNORE RRJCT PIN
PROMISC ?
F
T
DISCARD FRAME
START RECEIVE DISCARD FRAME IF RRJCT PIN IS ASSERTED

**Figure 16-100. Ethernet Address Recognition Flowchart**

In the group type of address recognition, the Ethernet controller determines whether or not the group address is a broadcast address. If broadcast addresses are enabled, then the frame is accepted, but if the group address is not a broadcast address, then the user can perform address recognition on multiple group addresses using the GADDR1–4 hash table. In the promiscuous mode, the Ethernet controller receives all the incoming frames regardless of their address, unless the REJECT pin is asserted.

If an external CAM is used for address recognition, then the user should select the promiscuous mode, and the frame can be rejected by assertion of the REJECT pin while the frame is received. The on-chip address recognition functions can be used in addition to the external CAM address recognition functions.

**NOTE**

> If the external CAM is used to store addresses that should be rejected rather than accepted, then the use of the REJECT pin by the CAM should be logically inverted.

**16.14.24.12 HASH TABLE ALGORITHM.** The hash table process used in the individual and group hash filtering operates as follows. The Ethernet controller maps any 48-bit address into one of 64 bins, which are represented by 64 bits stored in GADDR1–4 or IADDR1–4. When the SET GROUP ADDRESS command is executed, the Ethernet controller maps the selected 48-bit address into one of the 64 bits. 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 1 and 64. Bits 31–30 of the CRC result select one of the four GADDRs or IADDRs and bits 29–26 of the CRC result select the bit within the selected register.

When a frame is received by the Ethernet controller, the same process is used. If the CRC generator selects a bit that is set in the group/individual hash table, the frame is accepted; otherwise, it is rejected. The result is that if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (or 87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight preferred addresses.

Better performance is achieved by using the group and individual hash tables at the same time. For instance, if eight group and eight physical addresses are stored in their respective hash tables, 87.5% of all frames (not just group address 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, preventing only a small fraction of the frames from reaching memory. In such instances, an external CAM is advised if the extra bus utilization cannot be tolerated. Refer to **Section 16.14.24.7 CAM Interface** for more details.

NOTE

> The 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.

**16.14.24.13 INTERPACKET GAP TIME.** The minimum interpacket gap time for back-to-back transmission is 9.6 µs. The receiver receives back-to-back frames with this minimum spacing. In addition, after the backoff algorithm, the transmitter waits for carrier sense to be negated before retransmitting the frame. The retransmission begins 9.6 µs after carrier sense is negated if it stays negated for at least 6.4 µs.

**16.14.24.14 COLLISION HANDLING.** If a collision occurs during frame transmission, the Ethernet controller will continue the transmission for at least 32 bit times, transmitting a JAM pattern consisting of 32 ones. If the collision occurs during the preamble sequence, the JAM pattern will be sent after the end of the preamble sequence.

If a collision occurs within 64 byte times, the retry process is initiated. The transmitter waits a random number of slot times. A slot time is 512 bit times (52 µs). If a collision occurs after 64 byte times, then no retransmission is performed and the buffer is closed with an LC error indication. If a collision occurs during frame reception, the reception is stopped. This error is only reported in the BD if the length of this frame is greater than or equal to the MINFLR or if the RSH mode is enabled in the PSMR.

**16.14.24.15 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 the LPB bit in the PSMR and the DIAG bits in the GSMR. Because of the full-duplex nature of the loopback operation, the performance of the other SCCs are degraded.

Internal loopback disconnects the SCC from the SI. The receive data is connected to the transmit data and the receive clock is connected to the transmit clock. Both FIFOs are used. The transmitted data from the transmit FIFO is received immediately into the receive FIFO. There is no heartbeat check in this mode and TENA should be configured to be a general-purpose output.

> (PCPAR[15]=0, PCDIR[15]=1, PCDAT[15]=0) and the HBC bit in the PSMR should be zero.

In external loopback operation, the Ethernet controller listens for data received from the EEST at the same time that it is transmitting.

**16.14.24.16 ETHERNET ERROR-HANDLING PROCEDURE.** The Ethernet controller reports frame reception and transmission error conditions using the channel BDs, the error counters, and the Ethernet event register.

#### 16.14.24.16.1 Transmission Errors.

**Transmitter Underrun**

If this error occurs, the channel sends 32 bits that ensure a CRC error, terminates buffer transmission, closes the buffer, sets the UN bit in the Tx BD, and sets TXE in the Ethernet event register. The channel resumes transmission after receiving the RESTART TRANSMIT command.

**Carrier Sense Lost During Frame Transmission**

When this error occurs and no collision is detected in the frame, the channel sets the CSL bit in the Tx BD, sets the TXE in the Ethernet event register, and continues the buffer transmission normally. No retries are performed as a result of this error.

**Retransmission Attempts Limit Expired**

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the RL bit in the Tx BD, and sets TXE. The channel resumes transmission after receiving the RESTART TRANSMIT command.

**Late Collision**

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the LC bit in the Tx BD, and sets TXE. The channel resumes transmission after receiving the RESTART TRANSMIT command.

**NOTE**

The definition of what constitutes a late collision is made in the LCW bit of the PSMR.

**Heartbeat**

Some transceivers have a self-test feature called “heartbeat” or “signal quality error.” To signify a good self-test, the transceiver indicates a collision to the MPC821 within 20 clocks after completion of a frame transmitted by the Ethernet controller. This indication of a collision does not imply a real collision error on the network, but is rather an indication that the transceiver still seems to be functioning properly. This is called the heartbeat condition.

If the HBC bit is set in the Ethernet mode register and the heartbeat condition is not detected by the MPC821 after a frame transmission, then a heartbeat error occurs. When this error occurs, the channel closes the buffer, sets the HB bit in the Tx BD, and generates the TXE interrupt if it is enabled.

### 16.14.24.16.2 Reception Errors.

#### Overrun Error

The Ethernet controller maintains an internal FIFO for receiving data. If a receiver FIFO overrun occurs, the channel writes the received data byte to the internal FIFO over the previously received byte. The previous data byte and frame status are lost. The channel closes the buffer, sets the OV bit in the Rx BD, sets RXF in the Ethernet event register, and increments the discarded frame counter (DISFC). The receiver then enters the hunt mode.

#### Busy Error

A frame is received and discarded because of a lack of buffers. The channel sets BSY in the Ethernet event register and increments the discarded frame counter (DISFC).

#### Non-Octet Error (Dribbling Bits)

The Ethernet controller handles up to seven dribbling bits when the receive frame terminates nonoctet aligned and it checks the CRC of the frame on the last octet boundary. If there is a CRC error, then the frame nonoctet aligned (NO) error is reported, the RXF bit is set, and the alignment error counter (ALEC) is incremented. If there is no CRC error, then no error is reported.

#### CRC Error

When a CRC error occurs, the channel closes the buffer, sets the CR bit in the Rx BD, and sets the RXF bit. The channel also increments the CRC error counter (CRCEC). After receiving a frame with a CRC error, the receiver enters hunt mode. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

**16.14.24.17 ETHERNET MODE REGISTER.** The Ethernet mode register is a 16-bit, memory-mapped, read/write register that controls the SCC operation. The term Ethernet mode register refers to the PSMR of the SCC when that SCC is configured for Ethernet. This register is cleared at reset.

PSMR

| BITS | 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 | 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 | A08 (PSMR1), A28 (PSMR2) | | | | | | | | | | | | | | | |

HBC—Heartbeat Checking

0 = No heartbeat checking is performed. Do not wait for a collision after transmission.

1 = Wait 20 transmit clocks (2 µs) for a collision asserted by the transceiver after transmission. The HB bit in the Tx BD is set if the heartbeat is not heard within 20 transmit clocks.

FC—Force Collision

0 = Normal operation.
1 = The channel forces a collision on transmission of every transmit frame. The MPC821 should be configured in loopback operation when using this feature, which allows the MPC821 collision logic to be tested by the user. It results in the retry limit being exceeded for each transmit frame.

RSH—Receive Short Frames

0 = Discard short frames (less than MINFLR in length).
1 = Receive short frames.

IAM—Individual Address Mode

0 = Normal operation. A single 48-bit physical address stored in PADDR1 is checked on receive.
1 = The individual hash table is used to check all individual addresses that are received.

CRC—CRC Selection

00 = Reserved.
01 = Reserved.
10 = 32-bit CCITT-CRC (Ethernet). X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X1 +1. Select this to comply with Ethernet specifications.
11 = Reserved.

PRO—Promiscuous

0 = Check the destination address of the incoming frames.
1 = Receive the frame regardless of it’s address, unless the REJECT pin is asserted during the frame reception.

BRO—Broadcast Address

0 = Receive all frames containing the broadcast address.
1 = Reject all frames containing the broadcast address unless PRO = 1.

SBT—Stop Backoff Timer

0 = The backoff timer functions normally.
1 = The backoff timer (for the random wait after a collision) is stopped whenever carrier sense is active. In this method, the retransmission is less aggressive than the maximum allowed in the IEEE 802.3 standard. The persistence (P_Per) feature in the parameter RAM can be used in combination with the SBT bit (or in place of the SBT bit), if preferred.

LPB—Loopback Operation

0 = Normal operation.
1 = The channel is configured into internal or external loopback operation as determined by the DIAG bits of the GSMR. For external loopback, the DIAG bits should be configured for normal operation and for internal loopback they should be configured for loopback operation.

SIP—Sample Input Pins

0 = Normal operation.
1 = After the frame is received, the value on the PB23–PB16 pins is sampled and written to the end of the last receive buffer of the frame. This value is called a tag byte. If the frame is discarded, the tag byte is also discarded.

LCW—Late Collision Window

0 = A late collision is any collision that occurs at least 64 bytes from the preamble.
1 = A late collision is any collision that occurs at least 56 bytes from the preamble.

NIB—Number of Ignored Bits

This parameter determines how soon after RENA assertion that the Ethernet controller should begin looking for the start frame delimiter. In most situations, the user should select 22 bits.

000 = Begin searching for the SFD 13 bits after the assertion of RENA.
001 = Begin searching for the SFD 14 bits after the assertion of RENA.
010 = Begin searching for the SFD 15 bits after the assertion of RENA.
011 = Begin searching for the SFD 16 bits after the assertion of RENA.
100 = Begin searching for the SFD 21 bits after the assertion of RENA.
101 = Begin searching for the SFD 22 bits after the assertion of RENA.
110 = Begin searching for the SFD 23 bits after the assertion of RENA.
111 = Begin searching for the SFD 24 bits after the assertion of RENA.

FDE—Full Duplex Ethernet

0 = Disable full-duplex ethernet mode.
1 = Enable full-duplex ethernet.

**NOTE**

When this bit is set to 1, the LPB bit is set to 1 as well.

**16.14.24.18 ETHERNET RECEIVE BUFFER DESCRIPTOR.** The Ethernet controller uses the Rx BD to report information about the received data for each buffer. Figure 16-101 illustrates an Ethernet Rx BD example.

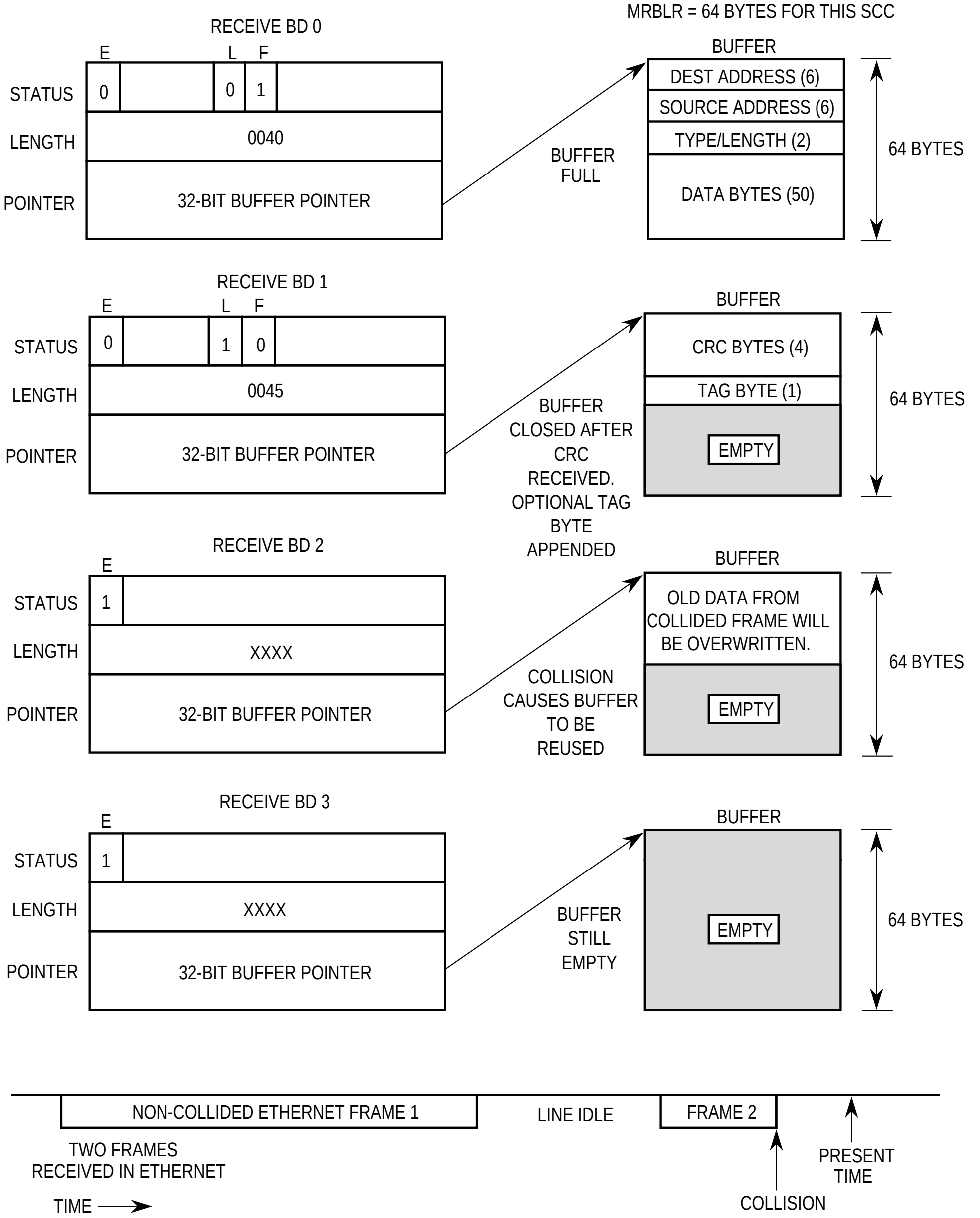


**Figure 16-101. Ethernet Rx BD Example**

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **E** | RES | **W** | **I** | L | F | | M | | | LG | NO | SH | CR | OV | CL |
| **OFFSET + 2** | DATA LENGTH | | | | | | | | | | | | | | | |
| **OFFSET + 4** | RX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

E—Empty

0 = The data buffer associated with this Rx BD has been filled with received data, or data reception has been aborted due to an error condition. The CPU core is free to examine or write to any fields of this Rx BD. The CP does not use this BD as long as the E-bit is zero.

1 = The data buffer associated with this Rx BD is empty or reception is currently in progress. This Rx BD and it's associated receive buffer are owned by the CP. Once the E bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Rx BD table.

1 = This is the last BD in the Rx BD 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 Rx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is used.

1 = The RXB or RXF bit of the Ethernet event register is set when this buffer is used by the Ethernet controller. These two bits can cause interrupts if they are enabled.

L—Last in Frame

This bit is set by the Ethernet controller when this buffer is the last one in a frame. This implies the end of the frame or reception of an error, in which case 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.

F—First in Frame

This bit is set by the Ethernet controller 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.

Bits 6–9—Reserved

M—Miss

This bit is set by the Ethernet controller for frames that are accepted in promiscuous mode, but are flagged as a “miss” by the internal address recognition. Thus, while in promiscuous mode, the user uses the Miss bit to quickly determine whether the frame is destined to this station. This bit is valid only if the L bit is set.

0 = The frame is received because of an address recognition hit.
1 = The frame is received because of promiscuous mode.

LG—Rx Frame Length Violation

A frame length greater than the maximum defined for this channel is recognized (only the maximum-allowed number of bytes is written to the data buffer).

NO—Rx Nonoctet Aligned Frame

A frame that contained a number of bits not divisible by eight is received and the CRC check that occurs at the preceding byte boundary generated an error.

SH—Short Frame

A frame length that is less than the minimum defined for this channel is recognized. This indication is possible only if the RSH bit is set in the PSMR.

CR—Rx CRC Error

This frame contains a CRC error.

OV—Overrun

A receiver overrun occurs during frame reception.

CL—Collision

This frame is closed because a collision occurs during frame reception. This bit is only set if a late collision occurs or if the RSH bit is enabled in the PSMR. The late collision definition is determined by the LCW bit of the PSMR.

Data Length

The data length is the number of octets the CP writes into this BD data buffer. It is written by the CP once as the buffer is closed.

When this BD is the last BD in the frame (L = 1), the data length contains the total number of frame octets (including four bytes for CRC).

**NOTE**

The actual amount of memory allocated for this buffer should be greater than or equal to the contents of the MRBLR.

Rx Data Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, can reside in internal or external memory. This pointer must be divisible by four.

**16.14.24.19 ETHERNET TRANSMIT BUFFER DESCRIPTOR.** Data is presented to the Ethernet controller for transmission on an SCC channel by arranging it in buffers referenced by the channel Tx BD table. The Ethernet controller confirms transmission or indicates error conditions using the BDs to inform the host that the buffers have been serviced.

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **R** | **PAD** | **W** | **I** | **L** | **TC** | DEF | HB | LC | RL | RC | | | | UN | CSL |
| **OFFSET + 2** | **DATA LENGTH** | | | | | | | | | | | | | | | |
| **OFFSET + 4** | **TX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission and the user is free to manipulate this BD or it's associated data buffer. The CP clears this bit after the buffer has been transmitted or after an error condition is encountered.

1 = The data buffer, which has been prepared for transmission by the user, has not been transmitted or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

PAD—Short Frame Padding

This bit is only valid when the L-bit is set; otherwise, it is ignored.

0 = Do not add PADs to short frames.

1 = Add PADs to short frames. Pad bytes are inserted until the length of the transmitted frame equals the MINFLR and they are stored in PADs in the parameter RAM.

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Tx BD table.

1 = This is the last BD in the Tx BD table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to in the table. The number of Tx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

**NOTE**

The Tx BD table must contain more than one BD in Ethernet mode.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.
1 = The TXB or TXE bit is set in the Ethernet event register after this buffer is serviced. These bits can cause interrupts if they are enabled.

L—Last

0 = This is not the last buffer in the transmit frame.
1 = This is the last buffer in the current transmit frame.

TC—Tx CRC

This bit is valid only when the L-bit is set; otherwise, it is ignored.

0 = End transmission immediately after the last data byte.
1 = Transmit the CRC sequence after the last data byte.

The following status bits are written by the Ethernet controller after it finishes transmitting the associated data buffer.

DEF—Defer Indication

This frame had a collision before being successfully sent. Useful for channel statistics.

HB—Heartbeat

The collision input is not asserted within 20 transmit clocks following the completion of transmission. This bit cannot be set unless the HBC bit is set in the PSMR.

LC—Late Collision

A collision occurs after the number of bytes defined with the LCW bit in the PSMR (either 56 or 64) are transmitted. The Ethernet controller terminates the transmission.

RL—Retransmission Limit

The transmitter fails Retry Limit + 1 attempts to successfully transmit a message because of repeated collisions on the medium.

RC—Retry Count

These four bits indicate the number of retries required before this frame is successfully transmitted. If RC = 0, then the frame is transmitted correctly the first time. If RC = 15 and RET_Lim = 15 in the parameter RAM, then 15 retries are required. If RC = 15 and RET_Lim > 15 in the parameter RAM, then 15 or more retries are required.

UN—Underrun

The Ethernet controller encounters a transmitter underrun condition while transmitting the associated data buffer.

CSL—Carrier Sense Lost

Carrier sense is lost during frame transmission.

Data Length

The data length is the number of octets the Ethernet controller should transmit from this BD data buffer. It is never modified by the CP. The value of this field should be greater than zero.

Tx Data Buffer Pointer

The transmit buffer pointer, which contains the address of the associated data buffer, can be even or odd. The buffer can reside in internal or external memory. This value is never modified by the CP.

**16.14.24.20 ETHERNET EVENT REGISTER.** The SCCE is called the Ethernet event register when the SCC is operating as an Ethernet controller. It is a 16-bit register used to generate interrupts and report events recognized by the Ethernet channel. On recognition of an event, the Ethernet controller sets the corresponding bit in the Ethernet event register. Interrupts generated by this register can be masked in the Ethernet mask register. An example of interrupts that can be generated in the HDLC protocol is illustrated in Figure 16-102.

The Ethernet event register is a memory-mapped register that can be read at any time. A bit is cleared by writing a 1(writing a zero does not affect a bit value) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset.

SCCE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | | GRA | RES | | TXE | RXF | BSY | TXB | RXB |
| 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 | A10 (SCC1), A30 (SCC2) | | | | | | | | | | | | | | | |

Bits 0–7—Reserved

GRA—Graceful Stop Complete

A graceful stop, initiated by the GRACEFUL STOP TRANSMIT command, is now complete. This bit is set as soon the transmitter finishes transmitting any frame that was in progress when the command was issued. It is set immediately if no frame was in progress when the command was issued.

Bits 9–10—Reserved

TXE—Tx Error

An error occurs on the transmitter channel.

RXF—Rx Frame

A complete frame is received on the Ethernet channel.

BSY—Busy Condition

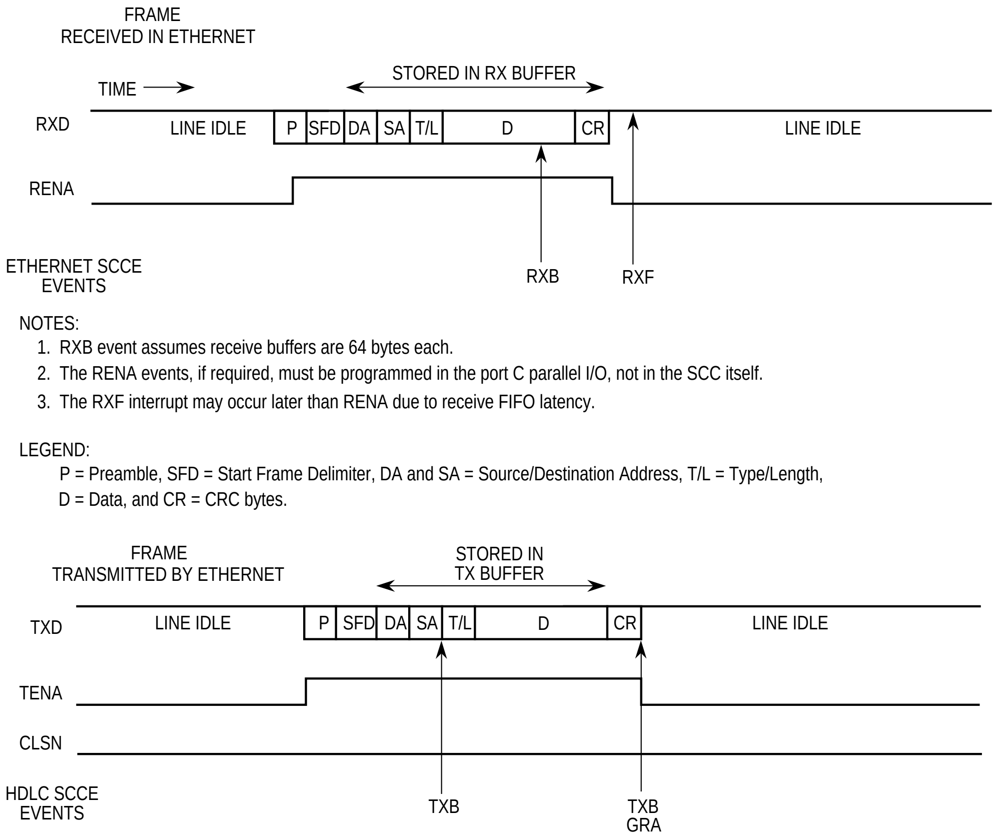
A frame is received and discarded due to a lack of buffers.

TXB—Tx Buffer

A buffer has been transmitted on the Ethernet channel.

RXB—Rx Buffer

A buffer that was not a complete frame is received on the Ethernet channel.



NOTES:

1. RXB event assumes receive buffers are 64 bytes each.
2. The RENA events, if required, must be programmed in the port C parallel I/O, not in the SCC itself.
3. The RXF interrupt may occur later than RENA due to receive FIFO latency.

LEGEND:

P = Preamble, SFD = Start Frame Delimiter, DA and SA = Source/Destination Address, T/L = Type/Length,
D = Data, and CR = CRC bytes.

NOTES:

1. TXB events assume the frame required two transmit buffers.
2. The GRA event assumes a GRACEFUL STOP TRANSMIT command was issued during frame transmission.
3. The TENA or CLSN events, if required, must be programmed in the port C parallel I/O, not in the SCC itself.

**Figure 16-102. Ethernet Interrupt Events Example**

**16.14.24.21 ETHERNET MASK REGISTER.** The SCCM is referred to as the Ethernet mask register when the SCC is operating as an Ethernet controller. It is a 16-bit read/write register that has the same bit formats as the Ethernet event register. If a bit in the Ethernet mask register is a 1, the corresponding interrupt in the event register is enabled. If the bit is zero, the corresponding interrupt in the event register is masked. This register is cleared upon reset.

**16.14.24.22 ETHERNET STATUS REGISTER.** This register is not valid for the Ethernet protocol. The current state of the RENA and CLSN signals can be read in port C.

**16.14.24.23 SCC ETHERNET EXAMPLE.** The following list is an initialization sequence for an Ethernet channel. SCC1 is used. The CLK1 pin is used for the Ethernet receiver, and the CLK2 pin is used for the Ethernet transmitter.

1. Configure the port A pins to enable the TXD1 and RXD1 pins. Write PAPAR bits 15 and 14 with ones, PADIR bits 15 and 14 with zeros, and PAODR bit 14 with zero.
2. Configure the port C pins to enable $\overline{\text{CTS1}}$ (CLSN) and CD1 (RENA). Write PCPAR and PCDIR bits 11 and 10 with zeros and PCSO bits 11 and 10 with ones.
3. Do not enable the RTS1 (TENA) pin yet because the pin is still functioning as RTS (inactive in the high state) and transmission on the LAN could accidentally begin.
4. Configure port A to enable the CLK1 and CLK2 pins. Write PAPAR bits 7 and 6 with ones and PADIR bits 7 and 6 with zeros.
5. Connect the CLK1 and CLK2 pins to SCC1 using the SI. Write the R1CS bits in the SICR register to 101 and the T1CS bits to 100.
6. Connect the SCC1 to the NMSI (its own set of pins) and clear the SC1 bit in the SICR.
7. Write $0001 to the SDDR to initialize the SDMA configuration register.
8. Write RBASE and TBASE in the SCC parameter RAM to point to the Rx BD and Tx BD in the dual-port RAM. Assuming one Rx BD at the beginning of the dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.
9. Program the CPCR to execute the INIT RX BD PARAMS command for this channel.
10. Write RFCR and TFCR with $18 for normal operation.
11. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 1,520 bytes, so MRBLR = $05F0. In this example, the user wants to receive an entire frame into one buffer, so the MRBLR value is chosen to be the first value larger than 1,518 that is evenly divisible by four.
12. Write C_PRES with $FFFFFFFF to comply with 32-bit CCITT-CRC.
13. Write C_MASK with $DEBB20E3 to comply with 32-bit CCITT-CRC.
14. Clear CRCEC, ALEC, and DISFC for clarity.
15. Write PAD with $8888 for the PAD value.
16. Write RET_Lim with $000F.
17. Write MFLR with $05EE to make the maximum frame size 1,518 bytes.
18. Write MINFLR with $0040 to make the minimum frame size 64 bytes.

19. Write MAXD1 and MAXD2 with $05EE to make the maximum DMA count 1,518 bytes.
20. Clear GADDR1–GADDR4. The group hash table is not used.
21. Write PADDR1_H with $0000, PADDR1_M with $0000, and PADDR1_L with $0040 to configure the physical address.
22. Write P_Per with $0000. It is not used.
23. Clear IADDR1–IADDR4. The individual hash table is not used.
24. Clear TADDR_H, TADDR_M, and TADDR_L for clarity.
25. Initialize the Rx BD and assume the Rx data buffer is at $00001000 in main memory. Write $B000 to Rx_BD_Status, $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
26. Initialize the Tx BD and assume the Tx data frame is at $00002000 in main memory and contains fourteen 8-bit characters (destination and source addresses plus the type field). Write $FC00 to Tx_BD_Status, add PAD to the frame and generate a CRC. Then write $000D to Tx_BD_Length and $00002000 to Tx_BD_Pointer.
27. Write $FFFF to the SCCE register to clear any previous events.
28. Write $001A to the SCCM register to enable the TXE, RXF, and TXB interrupts.
29. Write $40000000 to the CIMR so that SCC1 can generate a system interrupt. The CICR register should also be initialized.
30. Write $00000000 to the GSMR_H1 register to enable normal operation of all modes.
31. Write $1088000C to the GSMR_L1 register to configure the $\overline{\text{CTS}}$ (CLSN) and CD (RENA) pins 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 MPC821 transmit data. TPL and TPP are set for Ethernet requirements. The DPLL is not used with Ethernet. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled yet.
32. Write $D555 to the DSR.
33. Set the PSMR1 to $0A0A to configure 32-bit CRC, promiscuous mode (receive all frames), and begin searching for the start frame delimiter 22 bits after RENA.
34. Enable the TENA pin (RTS). Since the MODE bits of the GSMR are written to Ethernet, the TENA signal is low. Write PCPAR bit 15 with a one and PCDIR bit 15 with a zero.
35. Write $1088003C to the GSMR_L1 register to enable the SCC1 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

**NOTE**

After 14 bytes and the 46 bytes of automatic pad (plus the 4 bytes of CRC) are transmitted, the Tx BD is closed. Additionally, the receive buffer is closed after a frame is received. Any additional receive data beyond 1,520 bytes or a single frame causes a busy (out-of-buffers) condition since only one Rx BD is prepared.

## 16.15 SERIAL MANAGEMENT CONTROLLERS

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

- Each SMC can implement the UART protocol on it's own pins
- Each SMC can implement a totally transparent protocol on a multiplexed or nonmultiplexed line. This mode can also be used for a fast connection between MPC821s
- Each SMC channel fully supports the C/I and monitor channels of the GCI (IOM-2) in ISDN applications
- Two SMCs fully 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

### 16.15.1 Overview

The SMCs are two full-duplex ports that can be independently configured to support any one of three protocols—UART, transparent, or GCI. The SMCs can support simple UART operation for such purposes as providing a debug/monitor port in an application, allowing the two SCCs to be free for another purpose. The SMCs' UART functionality is reduced in comparison to the SCC's UART functionality. The SMC clock can be derived from one of the four internal baud rate generators or from an external clock pin. The clock provided to the SMC should be a 16x clock.

Totally transparent operation is also supported. In this mode, the SMC can be connected to a TDM channel (such as a T1 line) or directly to it's own set of pins. The receive and transmit clocks are derived from the TDM channel, the internal baud rate generators, or from an external clock. In either case, the clock provided to the SMCs should be a 1× clock. The transparent protocol also allows the use of an external synchronization pin for the transmitter and receiver. The SMC's transparent functionality is reduced in comparison to the SCC's transparent functionality.

Each SMC supports the C/I and monitor channels of the GCI bus (OM-2). In this case, the SMC is connected to a TDM channel in the SI. Refer to **Section 16.12 Serial Interface with Time-Slot Assigner** for the details of configuring the GCI interfaces. The SMCs also support loopback and echo modes for testing. Refer to Figure 16-103 for the SMC block diagram. The SMC receiver and transmitter are double-buffered, as illustrated in the block diagram. This corresponds to an effective FIFO size (latency) of two characters.

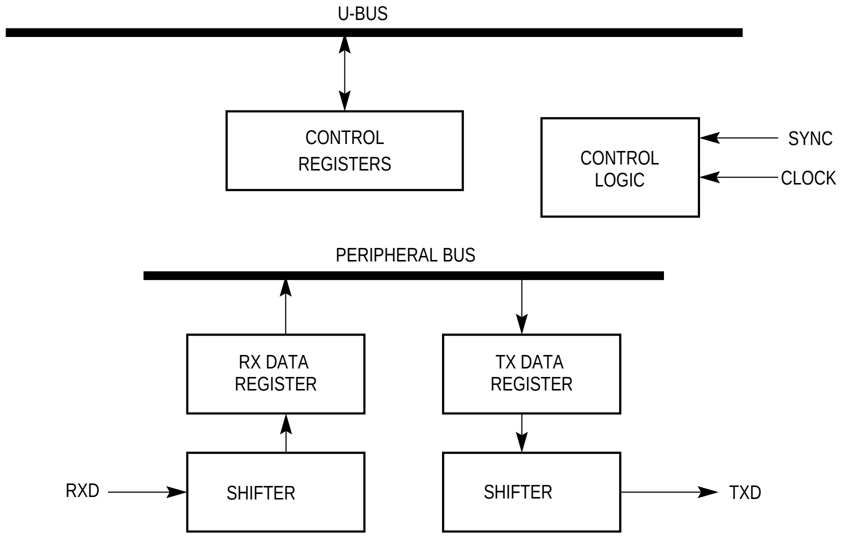


Figure 16-103. SMC Block Diagram

The receive data source for an SMC can be the L1RXD pin, if the SMC is connected to a TDM channel of the SI or the SMRXD pin, if the SMC is connected to the NMSI. Likewise, the transmit data source can be the L1TXD pin, if the SMC is connected to a TDM or the SMTXD pin, if the SMC is connected to the NMSI.

If the SMC is connected to a TDM, the SMC receive and transmit clocks can be independent from each other as defined in the SI description. However, if the SMC is connected to the NMSI, the SMC receive and transmit clocks must be connected to a single clock source called SMCLK, which is an internal signal name for a clock that is generated from the bank of clocks. SMCLK originates from an external pin or one of the four internal baud rate generators. Refer to **Section 16.12.8 NMSI Configuration** for more details.

If the SMC is connected to a TDM, it derives it’s synchronization pulse from the TSA. Otherwise, if the SMC is connected to the NMSI and the totally transparent protocol is selected, the SMC can use the SMSYN pin as a synchronization pin to determine when transmission and reception begins. The SMSYN pin is not used in the SMC UART mode.

## 16.15.2 General SMC Mode Register

The operating mode of each SMC port is defined by the 16-bit, memory-mapped, read/write general SMC mode register (SMCMR). Refer to the specific SMC protocol section for more information on this register.

### 16.15.3 SMC Buffer Descriptors

When the SMCs are configured to operate in GCI mode, the memory structure for the SMCs is predefined to be one half-word long for transmit and one half-word long for receive. For more information on these half-word structures, refer to **Section 16.15.14 SMC as a GCI Controller**.

However, in UART and transparent modes of operation, the SMCs have a memory structure that is like that of the SCCs, except that SMC associated data is stored in buffers. Each buffer is referenced by buffer descriptor and organized in a buffer descriptor ring located in the dual-port RAM. Refer to Figure 16-104 for details.

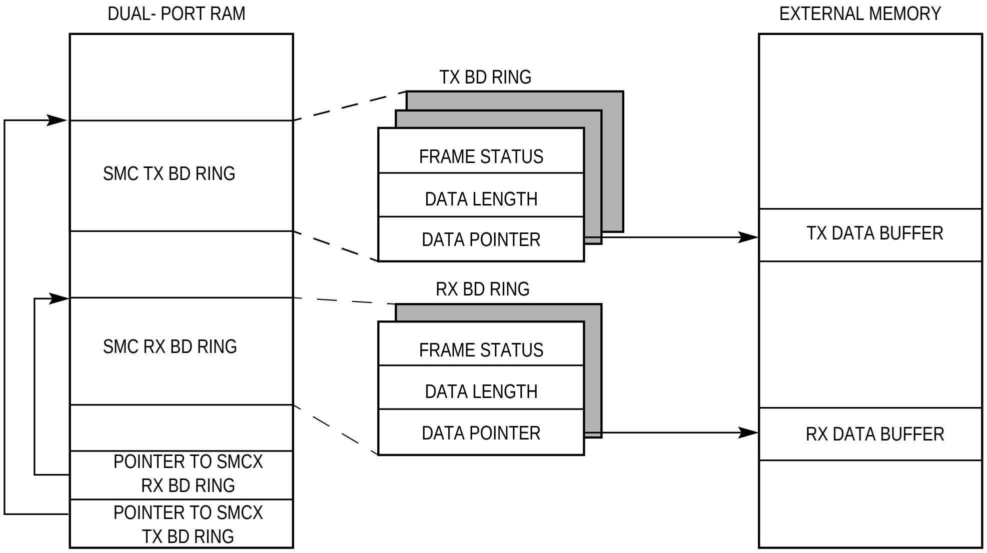


**Figure 16-104. SMC Memory Structure**

The BD ring allows the user to define buffers for transmission and buffers for reception. Each BD ring forms a circular queue. The CP confirms reception and transmission (or indicates error conditions) using the BDs to inform the processor that the buffers have been serviced. The actual buffers reside in external or internal memory and the data buffers reside in the parameter area of an SCC or SMC if that channel is not enabled.

### 16.15.4 SMC Parameter RAM

Each SMC parameter RAM area begins at the same offset from each SMC base area. The protocol-specific portions of the SMC parameter RAM are discussed in the specific protocol description sections. The part of the SMC parameter RAM that is the same for the UART and transparent SMC protocols is shown in Table 16-31. The following discussion does not apply to the GCI SMC protocol, which has it's own parameter RAM that is located in **Section 16.15.14 SMC as a GCI Controller**.

### Table 16-31. SMC UART and Transparent General-Purpose Parameter RAM

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SMC Base + 00 | **RBASE** | Half-word | Rx Buffer Descriptors Base Address |
| SMC Base + 02 | **TBASE** | Half-word | Tx Buffer Descriptors Base Address |
| SMC Base + 04 | **RFCR** | Byte | Rx Function Code |
| SMC Base + 05 | **TFCR** | Byte | Tx Function Code |
| SMC Base + 06 | **MRBLR** | Half-word | Maximum Receive Buffer Length |
| SMC Base + 08 | RSTATE | Word | Rx Internal State |
| SMC Base + 0C |  | Word | Rx Internal Data Pointer |
| SMC Base + 10 | RBPTR | Half-word | Rx Buffer Descriptor Pointer |
| SMC Base + 12 |  | Half-word | Rx Internal Byte Count |
| SMC Base + 14 |  | Word | Rx Temp |
| SMC Base + 18 | TSTATE | Word | Tx Internal State |
| SMC Base + 1C |  | Word | Tx Internal Data Pointer |
| SMC Base + 20 | TBPTR | Half-word | Tx Buffer Descriptor Pointer |
| SMC Base + 22 |  | Half-word | Tx Internal Byte Count |
| SMC Base + 24 |  | Word | Tx Temp |
| SMC Base + 28 |  |  | First Word of Protocol Specific Area |
| SMC Base + 36 |  |  | Last Word of Protocol Specific Area |

NOTE: Items in bold must be initialized by the user.
SMC base = IMMR + 1E80 (SMC1), 1F80 (SMC2).

Certain parameter RAM values need to be initialized by the user before the SMC is enabled and other values are initialized/written by the CP. Once initialized, most parameter RAM values do not need to be accessed in the user software since most of the activity is centered around the transmit and receive BDs and not the parameter RAM. However, if the parameter RAM is accessed by the user, the following restrictions should be noted.

- The parameter RAM can be read at any time.
- The parameter RAM values related to the SMC transmitter can only be written whenever the TEN bit in the SMCMR is zero or after the STOP TRANSMIT and before RESTART TRANSMIT commands.
- The parameter RAM values related to the SMC receiver can only be written whenever the REN bit in the SMCMR is zero or if the receiver is previously enabled after the ENTER HUNT MODE or CLOSE Rx BD commands before the REN bit is set.

**16.15.4.1 BD TABLE POINTER.** The RBASE and TBASE entries define the starting location in the dual-port RAM for the set of buffer descriptors for the SMC's receive and transmit functions. This provides flexibility in how buffer descriptors for an SMC are partitioned. By selecting RBASE and TBASE entries for all SMCs and by setting the W-bit in the last BD in each BD list, the user can select how many BDs to allocate for the transmit and receive side of every SMC. The user must initialize these entries before enabling the corresponding channel. Furthermore, the user should not configure BD tables of two enabled SMCs to overlap or erratic operation occurs.

**NOTE**

RBASE and TBASE should contain a value that is divisible by eight.

**16.15.4.2 SMC FUNCTION CODE REGISTERS.** There are eight separate function code registers for the two SMC channels—four for receive data buffers (RFCRx) and four for transmit data buffers (TFCRx). The FC entry contains the value that the user would like to appear on the function code pins AT0–AT3 when the associated SDMA channel accesses memory. It also controls the byte-ordering convention to be used in the transfers.

RFCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | SMC BASE + 04 | | | | | | | |

Bits 0–2—Reserved

BO—Byte Ordering

This bit field should be set by the user to select the required byte ordering of the data buffer.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

TFCR REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | SMC BASE + 05 | | | | | | | |

Bits 0–2—Reserved

BO—Byte Ordering

This bit field should be set by the user to select the required byte ordering of the data buffer. If this bit field is modified on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next BD.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

**16.15.4.3 MAXIMUM RECEIVE BUFFER LENGTH REGISTER.** Each SMC has one MRBLR to define it’s receive buffer length. MRBLR defines the maximum number of bytes that the MPC821 writes to a receive buffer on that SMC before moving to the next buffer. The MPC821 can write fewer bytes to the buffer than MRBLR if a condition like an error or end-of-frame occurs, but it never writes more bytes than the MRBLR value. It follows then, that buffers supplied by the user for the MPC821 to use should always be the size of MRBLR (or greater) in length.

The transmit buffers for an SMC are not affected in any way by the value programmed into MRBLR. Transmit buffers can be individually chosen to have varying lengths and the number of bytes to be transmitted is chosen by programming the data length field in the Tx BD.

**NOTE**

MRBLR should not be dynamically changed while an SMC is operating. However, if it is modified in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back), then a dynamic change in receive buffer length can be successfully achieved. This occurs when the CP moves control to the next Rx BD in the table. Thus, a change to MRBLR does not have an immediate effect. To guarantee the exact Rx BD on which the change occurs, the user should only change MRBLR while the SMC receiver is disabled. The MRBLR value should be greater than zero and should be even if the character length of the data is greater than 8 bits.

**16.15.4.4  RECEIVER BUFFER DESCRIPTOR POINTER.** The RBPTR for each SMC channel points to the next BD that the receiver will transfer 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 this pointer to the value programmed in the RBASE entry. Although RBPTR need never be written by the user in most applications, it may be modified by the user when the receiver is disabled or when the user is sure that no receive buffer is currently in use.

**16.15.4.5  TRANSMITTER BUFFER DESCRIPTOR POINTER.** The TBPTR for each SMC channel points to the next BD that the transmitter will transfer data from when it is in idle state or to the current BD during frame transmission. After a reset or when the end of the BD table is reached, the CP initializes this pointer to the value programmed in the TBASE entry. Although in most applications the user should never write RBPTR, it can be modified when the transmitter is disabled or the transmit buffer is not currently in use. For instance, after a STOP TRANSMIT command is issued, after a GRACEFUL STOP TRANSMIT command is issued, and the frame completes it's transmission.

**16.15.4.6  OTHER GENERAL PARAMETERS.** Additional parameters are listed in Table 13-30. These parameters do not need to be accessed by the user in normal operation and are listed only because they can provide helpful information for experienced users and for debugging. The Rx and Tx internal data pointers are updated by the SDMA channels to show the next address in the buffer to be accessed.

The Tx internal byte count is a down-count value that is initialized with the Tx BD data length and decremented with every byte read by the SDMA channels. On the other hand, the Rx internal byte count is a down-count value that is initialized with the MRBLR value and decremented with every byte written by the SDMA channels.

**NOTE**

> To extract data from a partially full receive buffer, the CLOSE Rx BD command can be used.

The Rx internal state, Tx internal state, Rx temp, Tx temp, and reserved areas are only for RISC use.

### 16.15.5 Disabling the SMCs On-the-Fly

If an SMC is not needed for a while, it can be disabled and reenabled later. In this case, a sequence of operations is followed that ensures that any buffers in use are properly closed and that new data is transferred to and from a new buffer. Such a sequence is required if the parameters to be changed are not allowed to be changed dynamically. If the register or bit description states that dynamic (on-the-fly) changes are allowed, the following sequences are not required and the register or bit can may be changed immediately. In all other cases, the sequence should be used. For instance, the baud rate generators allow on-the-fly changes.

**NOTE**

> The modification of parameter RAM does not require the SMC to be fully disabled. Refer to the parameter RAM description for details on when parameter RAM values can be modified. If the user wants to disable all SCCs, SMCs, SPI, and the $I^2C$, then the CPCR can be used to reset the CP with a single command.

**16.15.5.1 SMC TRANSMITTER FULL SEQUENCE.** For the SMC transmitter, the full disable and enable sequence is as follows:

1. Issue the STOP TRANSMIT command. This command is recommended when the SMC is currently in the process of transmitting data since it stops transmission smoothly. If the SMC is not transmitting (no Tx BDs are ready), then this command is not required. Furthermore, if the TBPTR is overwritten by the user or the INIT TX PARAMETERS command is executed, this command is not required.
2. Clear the TEN bit in the SMCMR. This disables the SMC transmitter and puts it in a reset state.
3. Make modifications. The user can make modifications to the SMC transmit parameters, including the parameter RAM. If the user prefers to switch protocols or restore the SMC transmit parameters to their initial state, the INIT TX PARAMETERS command can now be issued.
4. Issue the RESTART TRANSMIT command. This command is required if the INIT TX PARAMETERS command was not issued in step 3.
5. Set the TEN bit in the SMCMR. Transmission now begins using the Tx BD that the TBPTR value pointed to as soon as the Tx BD R-bit is set.

**16.15.5.2 SMC TRANSMITTER SHORTCUT SEQUENCE.** A shorter sequence is possible if the user prefers to reinitialize the transmit parameters to the state they had after reset. This sequence is as follows:

1. Clear the TEN bit in the SMCMR.
2. Issue the INIT TX PARAMETERS command. Any additional modifications can now be made.
3. Set the TEN bit in the SMCMR.

**16.15.5.3 SMC RECEIVER FULL SEQUENCE.** For the receiver, the full disable and enable sequence is as follows:

1. Clear the REN bit in the SMCMR. Reception is then aborted immediately. This disables the receiver of the SMC and puts it in a reset state.
2. Make modifications. The user can make modifications to the SMC receive parameters, including the parameter RAM. If the user prefers to switch protocols or restore the SMC receive parameters to their initial state, the INIT RX PARAMETERS command can now be issued.
3. Issue the CLOSE Rx BD command. This command is required if the INIT RX PARAMETERS command was not issued in step 2.
4. Set the REN bit in the SMCMR. Reception begins immediately using the Rx BD that the RBPTR pointed to if the Rx BD E-bit is set.

**16.15.5.4 SMC RECEIVER SHORTCUT SEQUENCE.** A shorter sequence is possible if the user desires to re-initialize the receive parameters to the state they had after reset. This sequence is as follows:

1. Clear the REN bit in the SMCMR.
2. INIT RX PARAMETERS Command. Any additional modifications can now be made.
3. Set the REN bit in the SMCMR.

**16.15.5.5 SWITCHING PROTOCOLS.** Sometimes the user prefers to switch the protocol that the SMC is executing (for instance, UART to transparent) without resetting the board or affecting any other SMC. This can be accomplished with one command and a short number of steps:

1. Clear the TEN and REN bits in the SMCMR.
2. Issue the INIT TX AND RX PARAMETERS command. This command initializes both the transmit and receive parameters. Any additional modifications can now be made in the SMCMR to change the protocol.
3. Set the SMCMR TEN and REN bits. The SMC is enabled with the new protocol.

### 16.15.6 Saving Power

When the TEN and REN bits of an SMC are cleared, that SMC consumes a minimal amount of power.

### 16.15.7 SMC as a UART

**16.15.7.1 FEATURES.** The following is a list of the SMC UART's important features:

- Flexible message-oriented data structure
- Programmable data length (5–14 bits)
- Programmable 1 or 2 stop bits
- Even/odd/no parity generation and checking
- Frame error, break, and IDLE detection
- Transmit preamble and break sequences
- Received break character length indication
- Continuous receive and transmit modes

**16.15.7.2 SMC UART COMPARISON.** As compared to the UART modes supported by the SCCs, the SMCs generally offer less functionality and performance. This fits with their purpose of providing simple debug/monitor ports rather than full-featured UARTs. The following is a list of the features that the SMC UART does not support.

- RTS, $\overline{\text{CTS}}$, and CD pins
- Receive and transmit sections being clocked at different rates
- Fractional stop bits
- Built-in multidrop modes
- Freeze mode for implementing flow control
- Isochronous operation (1x clock)
- Interrupts on receiving special control characters
- Ability to transmit data on demand using the TODR
- SCCS register to determine idle status of the receive pin
- Other features for the SCCs as described in the GSMR

However, the SMCs in UART mode do provide one feature not provided by the regular SCCs. The SMCs allow a data length option of up to 14 bits, whereas the SCCs provide a data length up to 8 bits. Refer to Figure 16-105 for the SMC UART frame format.

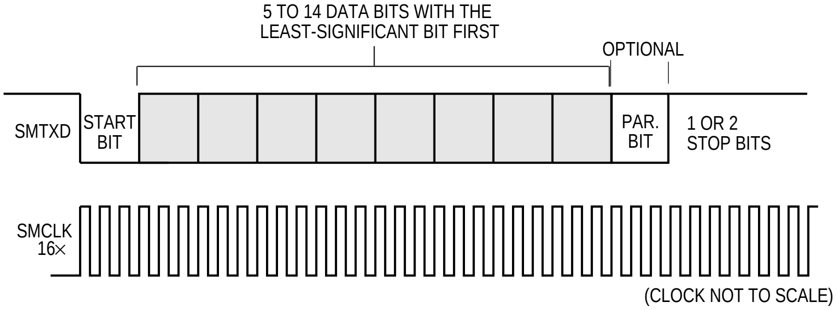


**Figure 16-105. SMC UART Frame Format**

**16.15.7.3 SMC UART MEMORY MAP.** When configured to operate in UART mode, the MPC821 overlays the structure listed in Table 16-23 with the UART-specific parameters described in Table 16-32.

**Table 16-32. SMC UART-Specific Parameter RAM**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SMC Base + 28 | **MAX_IDL** | Half-word | Maximum Idle Characters |
| SMC Base + 2A | IDLC | Half-word | Temporary Idle Counter |
| SMC Base + 2C | BRKLN | Half-word | Last Received Break Length |
| SMC Base + 2E | **BRKEC** | Half-word | Receive Break Condition Counter |
| SMC Base +30 | **BRKCR** | Half-word | Break Count Register (Transmit) |
| SMC Base +32 | R_mask | Half-word | Temporary Bit Mask |

NOTE: Items in bold must be initialized by the user.
SMC base = IMMR + 1E80 (SMC1), 1F80 (SMC2).

- MAX_IDL—Once a character of data is received on the line, the UART controller begins counting any idle characters received. If a MAX_IDL number of idle characters is received before the next data character is received, an idle timeout occurs and the buffer closes. This, in turn, produces an interrupt request to the CPU core to receive the data from the buffer. Thus, MAX_IDL provides a convenient way to demarcate frames in the UART mode. If the MAX_IDL functionality is not preferred, the user should program MAX_IDL to $0000 and the buffer will never be closed, regardless of the number of idle characters received. A character of idle is calculated as the following number of bit times: 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, the character length is 10 bits.
- IDLC—This value is used by the RISC to store the current idle counter value in the MAX_IDL timeout process. IDLC is a down-counter and does not need to be initialized or accessed by the user.

- BRKLN—This value is used to store the length of the last break character received and is the length in bits of that character. For example, If the receive pin is low for 257 bit times, BRKLN shows the value $0101. BRKLN is accurate to within one character unit of bits. For example, for 8 data bits, no parity, 1 stop bit, and 1 start bit, BRKLN is accurate to within 10 bits.
- BRKEC—This counter counts the number of break conditions that occurs on the line. Notice that one break condition can last for hundreds of bit times, yet this counter is only incremented once during that period.
- BRKCR—The SMC UART controller sends a break character sequence whenever a STOP TRANSMIT command is issued. The number of break characters sent by the UART controller is determined by the value in BRKCR. In the case of 8 data bits, no parity, 1 stop bit, and 1 start bit, each break character is 10 bits in length and consists of all zeros.

**16.15.7.4 SMC UART TRANSMISSION PROCESSING.** The UART transmitter is designed to work with almost no intervention from the CPU core. When the CPU core enables the SMC transmitter, it starts transmitting idles. The SMC immediately polls the first BD in the transmit channel BD ring and thereafter once every character time, depending on the character length (every 7 to 16 serial clocks). When there is a message to transmit, the SMC fetches the data from memory and starts transmitting 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 the R-bit. An interrupt is issued if the I-bit in the BD is set. If the next Tx BD is ready, the data from its data buffer is appended to the previous data and transmitted out on the transmit pin, without gaps occurring between buffers. If the next Tx BD is not ready, the SMC starts transmitting idles and waits for the next Tx BD to be ready.

By appropriately setting the I-bit in each BD, interrupts can be generated after the transmission of each buffer, a specific buffer, or each block. The SMC then proceeds to the next BD in the table. If the CM bit is set in the Tx BD, the R-bit is not cleared, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this data buffer. For instance, if a single Tx BD is initialized with the CM bit and W-bit set, the data buffer is continuously transmitted until the user clears the R-bit of the BD.

**16.15.7.5 SMC UART RECEPTION PROCESSING.** When the CPU core enables the SMC receiver in UART mode, it enters hunt mode and waits for the first character to arrive. Once the first character arrives, the CP checks the first Rx BD to see if it is empty and then starts storing characters in the associated data buffer.

When the data buffer is filled or the MAX_IDL timer expires (assuming it was enabled), the SMC clears the E-bit in the BD and generates an interrupt if the I-bit in the BD is set. If the incoming data exceeds the length of the data buffer, the SMC fetches the next BD in the table and, if it is empty, continues transferring data to this BD associated data buffer. If the CM bit is set in the Rx BD, the E-bit is not cleared, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this data buffer.

**16.15.7.6 PROGRAMMING MODEL.** An SMC configured as a UART uses the same data structure as in the other modes. The SMC UART data structure supports multibuffer operation and allows the user to transmit break and preamble sequences. Overrun, parity, and framing errors are reported via the BDs. In it's simplest form, the SMC UART functions in a character-oriented environment, whereas each character is transmitted with accompanying stop bits and parity (as configured by the user) and received into separate 1-byte buffers. Reception of each buffer can generate a maskable interrupt.

Many applications want to take advantage of the message-oriented capabilities that the SMC UART supports by using linked buffers (in either receive or transmit). In this case, data is handled in a message-oriented environment and users can work on entire messages rather than operating on a character-by-character basis. A message can span several linked buffers and each message can be transmitted and received as a linked list of buffers without CPU intervention, which makes it easy to program and saves processor overhead. In the message-oriented environment, the idle sequence is used as the message delimiter. The transmitter can generate an idle sequence before starting a new message and the receiver can close a buffer upon detection of idle sequence.

**16.15.7.7 COMMAND SET.** The following transmit and receive commands are issued to the CPCR.

#### 16.15.7.7.1 Transmit Commands.

**STOP TRANSMIT**

This command disables the transmission of characters on the transmit channel. If the SMC UART controller receives this command during message transmission, the transmission is aborted. The SMC UART completes transmission of any data already transferred to its FIFO and shift register (up to two characters) and then stops transmitting data. The TBPTR is not advanced when this command is issued.

The SMC UART transmits a programmable number of break sequences and then transmits idles. The number of break sequences (which can be zero) should be written to the break count register before this command is given to the SMC UART controller.

**RESTART TRANSMIT**

This command enables the transmission of characters on the transmit channel. This command is expected by the SMC UART controller after disabling the channel in its SMCMR and after the STOP TRANSMIT command. The SMC UART controller resumes transmission from the current TBPTR in the channel's Tx BD table.

**INIT TX PARAMETERS**

This command initializes all the transmit parameters in this serial channel's parameter RAM to their reset state and should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the transmit and receive parameters.

#### 16.15.7.7.2 Receive Commands.

**ENTER HUNT MODE**

This command should not be used for an SMC UART channel. It is recommended that the CLOSE RX BD command be used instead.

**CLOSE RX BD**

This command is used to force the SMC to close the current receive BD 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 in the process of receiving data, no action is taken by this command.

**INIT RX PARAMETERS**

This command Initializes all the receive parameters in this serial channel parameter RAM to their reset state. This command should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the receive and transmit parameters.

**16.15.7.8 SEND BREAK.** A break is an all-zeros character without stop bits and is sent by issuing the STOP TRANSMIT command. The SMC UART completes transmission of any outstanding data and then sends a character with consecutive zeros. The number of zero bits in this character is the sum of the character length, plus the number of start, parity, and stop bits. The SMC UART transmits a programmable number of break characters according to the break count register and then reverts to idle or sends data if the RESTART TRANSMIT command was given before completion. At the completion of the break, the transmitter sends at least one character of idle before transmitting any data to guarantee recognition of a valid start bit.

**16.15.7.9 SENDING A PREAMBLE.** A preamble sequence provides a convenient way for the programmer to ensure that the line is idle before a new message is started. The preamble sequence length is constructed of consecutive ones of one character length and if the preamble bit in a BD is set, the SMC sends a preamble sequence before transmitting that data buffer. For example, for 8 data bits, no parity, 1 stop bit, and 1 start bit, a preamble of 10 ones would be sent before the first character in the buffer. If no preamble sequence is sent, data from two ready transmit buffers can be transmitted without causing a delay on the transmit pin between the two transmit buffers.

**16.15.7.10 SMC UART ERROR-HANDLING PROCEDURE.** The SMC UART reports character reception error conditions via the channel buffer descriptors and the SMC UART event register. However, there are no transmission errors for the SMC UART controller.

**Overrun Error**

The SMC UART maintains a two-character length FIFO for receiving data (shift register plus data register). The 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. Then the channel writes the received character to the buffer, closes the buffer, sets the OV bit in the BD, and generates the RX interrupt if it is enabled. Reception then continues normally.

### NOTE

The SMC UART may occasionally get an overrun error when the line is idle. The user should ignore an overrun when the line state is known to be idle.

### Parity Error

When a parity error occurs, the channel writes the received character to the buffer, closes the buffer, sets the PR bit in the BD, and generates the RX interrupt if it is enabled. Reception then continues normally.

### Idle Sequence Receive

An idle is detected when one character consisting of all ones is received. Once an idle is received, the channel counts the number of consecutive idle characters. If the count reaches the MAX_IDL value, the buffer is closed, and an RX interrupt is generated. If no receive buffer is open, this event does not generate an interrupt or any status information. The idle counter is reset every time a character is received.

### Framing Error

A framing error is detected by the SMC UART controller when a character with no stop bit is received. When this error occurs, the channel writes the received character to the buffer, closes the buffer, sets the FR bit in the BD, and generates the RX interrupt if it is enabled. When this error occurs, parity is not checked for this character.

### Break Sequence

A break sequence is detected by the SMC UART receiver when an all-zero character with a framing error is received. When this error occurs, the channel increments the BRKEC and generates a maskable BRK interrupt in the SMC UART event register. The channel also measures the length of the break sequence and stores this value in the BRKLN counter. If the channel was in the middle of buffer processing when the break was received, the buffer is closed with the BR bit in the Rx BD set and the RX interrupt is generated if it is enabled.

**16.15.7.11 SMC UART MODE REGISTER.** The operating mode of an SMC is defined by the SMC UART mode register as a 16-bit, memory-mapped, read/write register cleared at reset. The function of bits 8–15 is common to each SMC protocol, but bits 0–7 vary according to the protocol selected by the SM bits.

SMCMR REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RES | CLEN | | | | SL | PEN | PM | RES | | SM | | DM | | TEN | REN |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | A82 (SMCMR1), A92 (SMCMR2) | | | | | | | | | | | | | | | |

Bit 0—Reserved

These bits should be cleared by the user.

CLEN—Character Length

The CLEN value should be programmed with the total number of bits in the character minus one. The total number of bits in the character is calculated as the sum of: 1 (start bit always present) + number of data bits (5–14) + 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. Thus, CLEN should be programmed to 9.

The number of data bits in the character ranges from 5 to 14 bits. If the data bit length is less than 8 bits, the MSBs of each byte in memory are not used on transmit and are written with zeros on receive. On the other hand, if the data bit length is more than 8 bits, the MSBs of each 16-bit word in memory are not used on transmit and are written with zeros on receive.

**NOTE**

The total number of bits in the character must never exceed 16. Thus, if a 14-bit data length is chosen, SL must be set to one stop bit and parity should not be enabled. If a 13-bit data length is chosen and parity is enabled, SL must be set to one stop bit. The values 0 to 3 should not be written to CLEN or erratic behavior results.

SL—Stop Length

0 = One stop bit.
1 = Two stop bits.

PEN—Parity Enable

0 = No parity.
1 = Parity is enabled for the transmitter and receiver as determined by the PM bit.

PM—Parity Mode

0 = Odd parity.
1 = Even parity.

Bits 8–9—Reserved

These bits should be cleared by the user.

SM—SMC Mode

00 = GCI or SCIT support.
01 = Reserved.
10 = UART (must be selected for SMC UART operation).
11 = Totally transparent operation.

DM—Diagnostic Mode

00 = Normal operation.
01 = Local loopback mode.
10 = Echo mode.
11 = Reserved.

TEN—SMC Transmit Enable

0 = SMC transmitter disabled.
1 = SMC transmitter enabled.

REN—SMC Receive Enable

0 = SMC receiver disabled.
1 = SMC receiver enabled.

**16.15.7.12 SMC UART RECEIVE BUFFER DESCRIPTOR.** The CP reports information concerning the received data on a per-buffer basis via Rx BDs. The CP closes the current buffer, generates a maskable interrupt, and begins receiving data into the next buffer after one of the following events occurs:

1. An error is detected during message processing.
2. A full receive buffer is detected.
3. 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** | RES | **W** | **I** | RES | RES | CM | ID | RES | RES | BR | FR | PR | RES | OV | CD |
| **OFFSET + 2** | DATA LENGTH | | | | | | | | | | | | | | | |
| **OFFSET + 4** | RX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

An example of the UART Rx BD process is illustrated in Figure 16-106, which shows the resulting state of the Rx BDs after receipt of 10 characters, an idle period, and five characters (one with a framing error). The example assumes that MRBLR = 8 in the SMC parameter RAM.

E—Empty

0 = The data buffer associated with this Rx BD is filled with received data or data reception is aborted due to an error condition. The CPU3 core is free to examine or write to any fields of this Rx BD. The CP does not use this BD as long as the E-bit is zero.

1 = The data buffer associated with this BD is empty or reception is currently in progress. This Rx BD and it's associated receive buffer are owned by the CP. Once the E-bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Rx BD table.

1 = This is the last BD in the Rx BD 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 Rx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

Bits 4–5—Reserved

I—Interrupt

0 = No interrupt is generated after this buffer is filled.

1 = The RX bit in the event register is set when this buffer is completely filled by the CP, indicating the need for the CPU core to process the buffer. The RX bit can cause an interrupt if it is enabled.

CM—Continuous Mode

0 = Normal operation.

1 = The E-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this BD. However, the E-bit is cleared if an error occurs during reception, regardless of the CM bit.

The following status bits are written by the CP after the received data is in the associated data buffer.

ID—Buffer Closed on Reception of Idles

The buffer is closed because a programmable number of consecutive idle sequences are received.

Bits 8–9—Reserved

$\overline{\text{BR}}$—Buffer Closed on Reception of Break

The buffer is closed because a break sequence is received.

FR—Framing Error

A character with a framing error is received and located in the last byte of this buffer. A framing error is a character without a stop bit. A new receive buffer is used for further data reception.

PR—Parity Error

A character with a parity error is received and located in the last byte of this buffer. A new receive buffer is used for further data reception.

Bit 13—Reserved

OV—Overrun

A receiver overrun occurs during message reception.

Data Length

Data length is the number of octets the CP writes into this BD data buffer. It is written only once by the CP as the BD is closed.

**NOTE**

> The actual amount of memory allocated for this buffer should be greater than or equal to MRBLR.

Rx Data Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, must be even. The buffer can reside in internal or external memory.

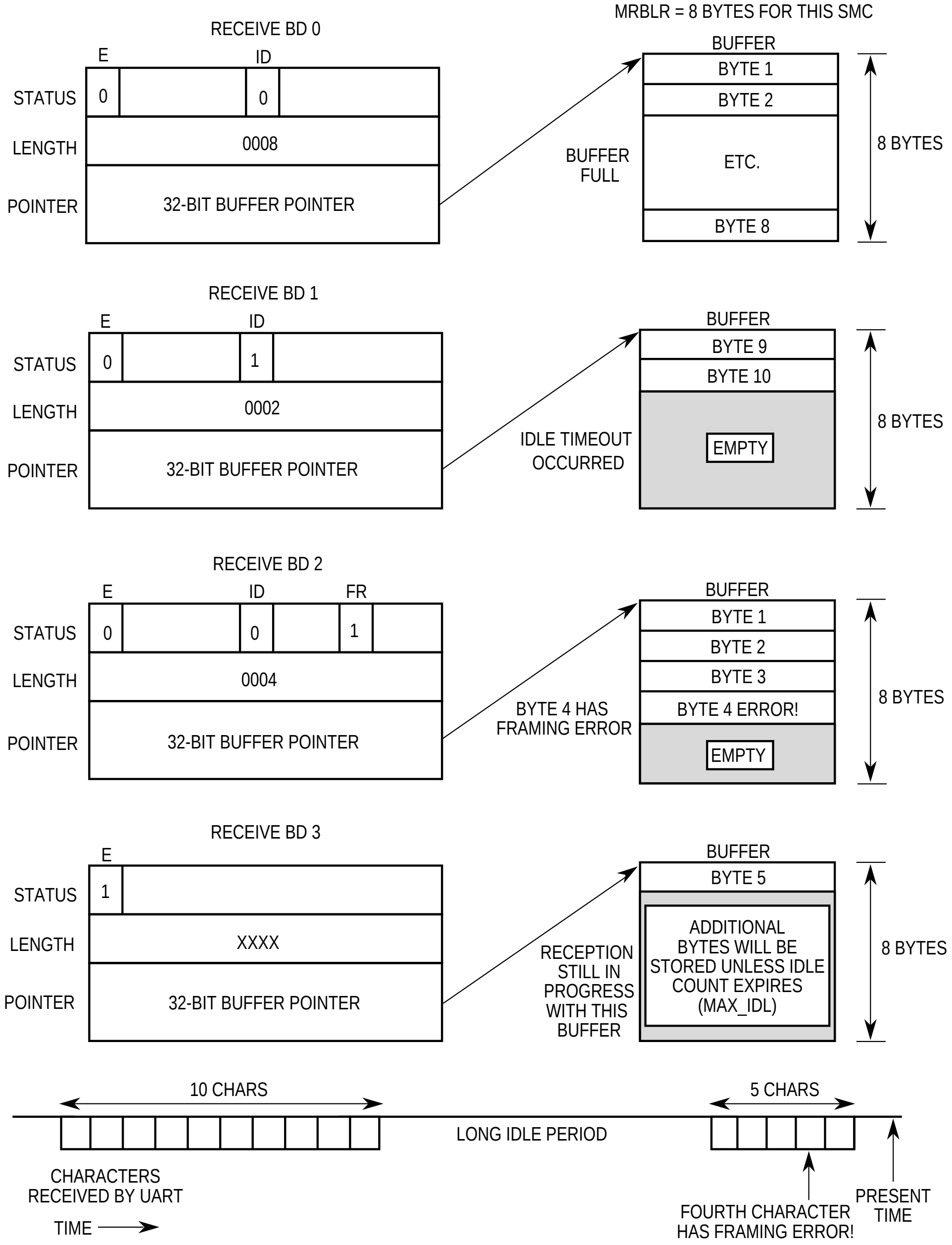


**Figure 16-106. SMC UART Rx BD Example**

**16.15.7.13 SMC UART TRANSMIT BUFFER DESCRIPTOR.** Data is presented to the CP for transmission on an SMC channel by arranging it in buffers referenced by the channel Tx BD ring. Through the BDs, the CP confirms transmission or indicates error conditions to inform the processor that the buffers have been serviced.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **R** | RES | **W** | **I** | RES | | **CM** | **P** | RESERVED | | | | | | | |
| OFFSET + 2 | **DATA LENGTH** | | | | | | | | | | | | | | | |
| OFFSET + 4 | **TX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission, but the user is free to manipulate this BD or it's associated data buffer. The CP clears this bit after the buffer has been transmitted or an error condition is encountered.

1 = The data buffer, which is prepared for transmission by the user, is not transmitted yet or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Tx BD table.

1 = This is the last BD in the Tx BD Table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to in the table. The number of Tx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = The TX bit in the SMC UART event register is set when this buffer is serviced. TX can cause an interrupt if it is enabled.

Bits 4–5—Reserved

CM—Continuous Mode

0 = Normal operation.

1 = The R-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this BD.

P—Preamble

0 = No preamble sequence is sent.
1 = The UART sends one all-ones character before sending 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.

Bits 8–15—Reserved

Data Length

The data length is the number of octets that the CP should transmit from this BD data buffer and it is never modified by the CP. Normally, this value should be greater than zero. However, the data length can be equal to zero with the P-bit set and only a preamble is sent.

If the number of data bits in the UART character is greater than 8, then the data length should be even. For example, to transmit three UART characters of 8-bit data, 1 start, and 1 stop, the data length field should be initialized to 3. However, to transmit three UART characters of 9-bit data, 1 start, and 1 stop, the data length field should be initialized to 6, since the three 9-bit data fields occupy three half-words in memory (the 9 LSBs of each half-word).

Tx Data Buffer Pointer

The transmit buffer pointer, which always points to the first location of the associated data buffer, can be even or odd (unless the number of actual data bits in the UART character is greater than 8 bits, then the transmit 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.

**16.15.7.14 SMC UART EVENT REGISTER.** When the UART protocol is selected, the SMCE register is called the SMC UART event register. It is an 8-bit register used to generate interrupts and report events recognized by the SMC UART channel. On recognition of an event, the UART sets the corresponding bit in the SMC UART event register.

The SMC UART event register is a memory-mapped register that can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset. An example of the timing of various events in the SMC UART event register is illustrated in Figure 16-107.

SMCE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RES | BRKE | RES | BRK | RES | BSY | TX | RX |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | A86 (SMCE1), A96 (SMCE2) | | | | | | | |

Bit 0—Reserved

BRKe—Break End

The end of break sequence was detected. This indication is no sooner than after one idle bit is received after a break sequence.

BRK—Break Character Received

A break character is received. If a very long break sequence occurs, this interrupt only occurs once after the first all-zeros character is received.

Bit 4—Reserved

BSY—Busy Condition

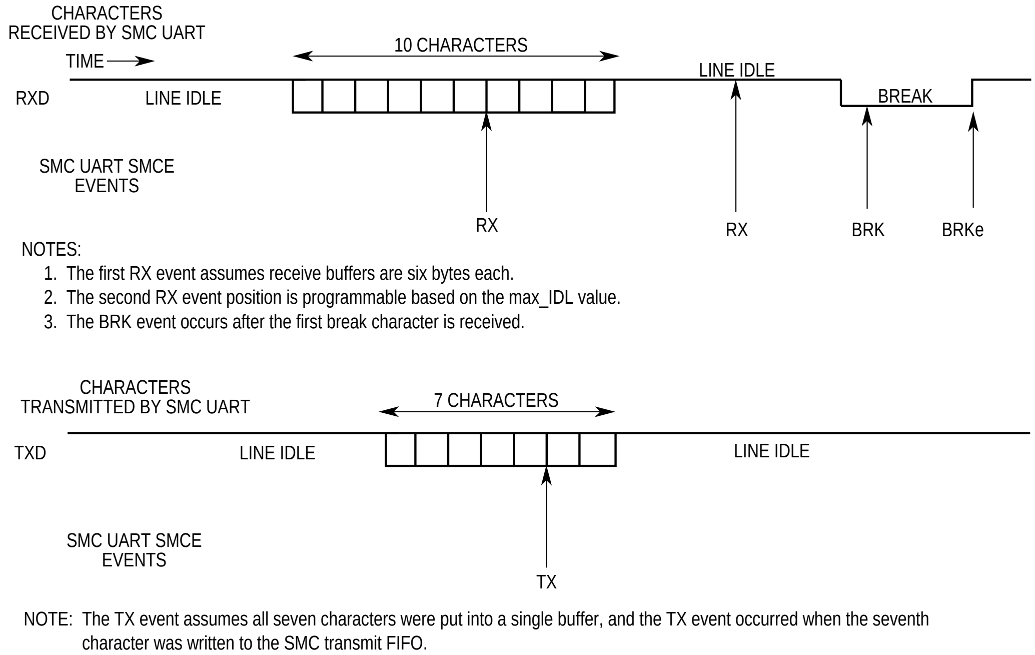
A character is received and discarded due to a lack of buffers. This bit is be set no sooner than the middle of the last stop bit of the first receive character for which there is no available buffer. Reception continues when an empty buffer is provided.

TX—Tx Buffer

A buffer is transmitted over the UART channel. This bit is set once the transmit data of the last character in the buffer is written to the transmit FIFO. The user must wait two character times to be sure that the data is completely sent over the transmit pin.

RX—Rx Buffer

A buffer is received and it's associated Rx BD is now closed. This bit is set no sooner than the middle of the last stop bit of the last character that is written to the receive buffer.



NOTES:
1. The first RX event assumes receive buffers are six bytes each.
2. The second RX event position is programmable based on the max_IDL value.
3. The BRK event occurs after the first break character is received.

NOTE: The TX event assumes all seven characters were put into a single buffer, and the TX event occurred when the seventh character was written to the SMC transmit FIFO.

**Figure 16-107. SMC UART Interrupts Example**

**16.15.7.15 SMC UART MASK REGISTER.** The SMCM is referred to as the SMC UART mask register when the SMC is operating as a UART. It is an 8-bit read/write register with the same bit format as the SMC UART event register. If a bit in the SMC UART mask register is a 1, the corresponding interrupt in the event register is enabled. If the bit is zero, the corresponding interrupt in the event register is masked. This register is cleared at reset.

## 16.15.8 SMC UART Example

The following list is an initialization sequence for 9,600 baud, 8 data bits, no parity, and 1 stop bit operation of an SMC UART assuming a 25 MHz system frequency. BRG1 and SMC1 are used.

1. Configure the port B pins to enable the SMTXD1 and SMRXD1. Write PBPAR bits 25 and 24 with ones and then PBDIR and PBODR bits 25 and 24 with zeros.
2. Configure the BRG1. Write BRGC1 with $010144. The DIV16 bit is not used and the divider is 162 (decimal). The resulting BRG1 clock is 16× the preferred bit rate of the UART.
3. Connect the BRG1 clock to SMC1 using the SI. Write the SMC1 bit in SIMODE with a 0 and the SMC1CS bits in SIMODE with 000.
4. Write RBASE and TBASE in the SMC parameter RAM to point to the Rx BD and Tx BD in the dual-port RAM. Assuming one Rx BD at the beginning of dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.

5. Program the CPCR to execute the INIT RX and TX PARAMS commands. Write $0091 to CPCR.
6. Write $0001 to the SDCR to initialize the SDMA configuration register.
7. Write RFCR and TFCR with $18 for normal operation.
8. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = $0010.
9. Write MAX_IDL with $0000 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 for the clarity.
11. Set BRKCR to $0001, so that if a STOP TRANSMIT command is issued, one break character is sent.
12. Initialize the Rx BD. Assume the Rx data buffer is at $00001000 in main memory. Write $B000 to Rx_BD_Status, $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
13. Initialize the Tx BD. Assume the Tx data buffer is at $00002000 in main memory and contains five 8-bit characters. Then write $B000 to Tx_BD_Status, $0005 to Tx_BD_Length, and $00002000 to Tx_BD_Pointer.
14. Write $FF to the SMCE register to clear any previous events.
15. Write $17 to the SMCM register to enable all possible SMC interrupts.
16. Write $00000010 to the CIMR so the SMC1 can generate a system interrupt. The CICR should also be initialized.
17. Write $4820 to SMCMR to configure normal operation (not loopback), 8-bit characters, no parity, 1 stop bit. Notice that the transmitter and receiver are not enabled yet.
18. Write $4823 to SMCMR to enable the SMC transmitter and receiver. This additional write ensures that the TEN and REN bits are enabled last.

**NOTE**

After 5 bytes are transmitted, the Tx BD is closed. The receive buffer is closed after 16 bytes are received. Any received data beyond 16 bytes caused a busy (out-of-buffers) condition since only one Rx BD is prepared.

### 16.15.9 SMC Interrupt Handling

The following list describes what normally occurs within an SMC interrupt handler:

1. Once an interrupt occurs, read the SMCE register to see which sources caused interrupts. The SMCE bits are usually cleared at this time.
2. Process the Tx BD to reuse it if the TX bit is set in the SMCE register. Extract data from the Rx BD if the RX bit is set in the SMCE. To transmit another buffer, simply set the Tx BD R-bit.
3. Clear the SMC1 bit in the CISR.
4. Execute the RFI instruction.

### 16.15.10 SMC as a Transparent Controller

**16.15.10.1 FEATURES.** The following is a list of the SMC transparent controller's important features:

- Flexible data buffers
- Can be used to connect to a TDM bus using the TSA in the serial interface
- Transmits and receives transparently on it's own set of pins using a sync pin to synchronize the beginning of transmission and reception to an external event
- Programmable character length (4–16)
- Reverse data mode
- Continuous transmission and reception modes
- Four commands

**16.15.10.2 SMC TRANSPARENT COMPARISON.** As compared to the transparent modes supported by the SCCs, the SMCs offer less functionality. This fits with their purpose of providing simpler functions and slower speeds. The following is a list of features the SMC transparent controller does not support:

- Independent transmit and receive clocks, unless connected to a TDM channel of the serial interface
- CRC generation and checking
- Full RTS, $\overline{\text{CTS}}$, and CD pins (supports only one SMSYN pin)
- 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
- Other features for the SCCs as described in the GSMR

The SMCs in transparent mode, however, do provide one feature not provided by the regular SCCs. The SMCs allow a data character length option of 4 to 16 bits; whereas, the SCCs provide a data character length of just 8 or 32 bits (determined by the RFW bit in the GSMR).

**16.15.10.3 SMC TRANSPARENT MEMORY MAP.** There is no protocol-specific parameter RAM for the SMC when it is used as a transparent controller. Only the general SMC parameter RAM is used, which is discussed in more detail in **Section 16.15.4 SMC Parameter RAM**.

**Table 16-33. SMC Transparent-Specific Parameter RAM**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SMC Base + 28 | Reserved | Half-word | Reserved |
| SMC Base + 2A | Reserved | Half-word | Reserved |
| SMC Base + 2C | Reserved | Half-word | Reserved |
| SMC Base + 2E | Reserved | Half-word | Reserved |
| SMC Base + 30 | Reserved | Half-word | Reserved |

NOTE: SMC base = IMMR + 1E80 (SMC1), 1F80 (SMC2).

**16.15.10.4 SMC TRANSPARENT TRANSMISSION PROCESSING.** The transparent transmitter is designed to work with almost no intervention from the CPU core. When the CPU enables the SMC transmitter in transparent mode, it starts transmitting idles. The SMC immediately polls the first BD in the transmit channel BD ring and thereafter, once every character time, depending on the character length (every 4 to 16 serial clocks). When there is a message to transmit, the SMC controller fetches the data from memory and starts transmitting the message once synchronization is achieved.

Synchronization can be achieved in two ways. First, when the transmitter is connected to a TDM channel, it can be synchronized to a time-slot. Once the frame sync is received, the transmitter waits for the first bit of its time-slot to occur before transmission begins. Data is only transmitted during the time-slots defined by the TSA. Secondly, when working with it’s own set of pins (nonmultiplexed mode), the transmitter starts transmitting when the SMSYNx line is asserted (falling edge).

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 (multibuffer mode), only the R-bit 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 the transmission of each buffer, a specific buffer, or each block. The SMC then proceeds to the next BD in the table. 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 transmitting idles.

If the CM bit is set in the Tx BD, the R-bit is not cleared, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this data buffer. For instance, if a single Tx BD is initialized with the CM bit and the W-bit set, the data buffer is continuously transmitted until the user clears the R-bit of the BD.

**16.15.10.5 SMC TRANSPARENT RECEPTION PROCESSING.** When the CPU core enables the SMC receiver in transparent mode, it waits for synchronization before receiving data. Once synchronization is achieved, the receiver transfers the incoming data into memory according to the first Rx BD in the ring. Synchronization can be achieved in two ways. First, when the receiver is connected to a TDM channel, it can be synchronized to a time-slot. Once the frame sync is received, the receiver waits for the first bit of it's time-slot to occur before reception begins. Data is only received during the time-slots defined by the TSA. Secondly, when working with it's own set of pins (nonmultiplexed mode), the receiver starts reception when the SMSYNx line is asserted (falling edge).

When the data buffer is filled, the SMC clears the E-bit in the BD and generates an interrupt if the I-bit in the BD is set. If the incoming data exceeds the length of the data buffer, the SMC fetches the next BD in the table and, if it is empty, continues transferring data to this BD associated data buffer. If the CM bit is set in the Rx BD, the E-bit is not cleared, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this data buffer.

**16.15.10.6 USING THE SMSYN PIN FOR SYNCHRONIZATION.** The SMSYN pin offers a method to externally synchronize the SMC channel. This method differs somewhat from the synchronization options available in the SCCs and should be studied carefully. See Figure 16-108 for an example.

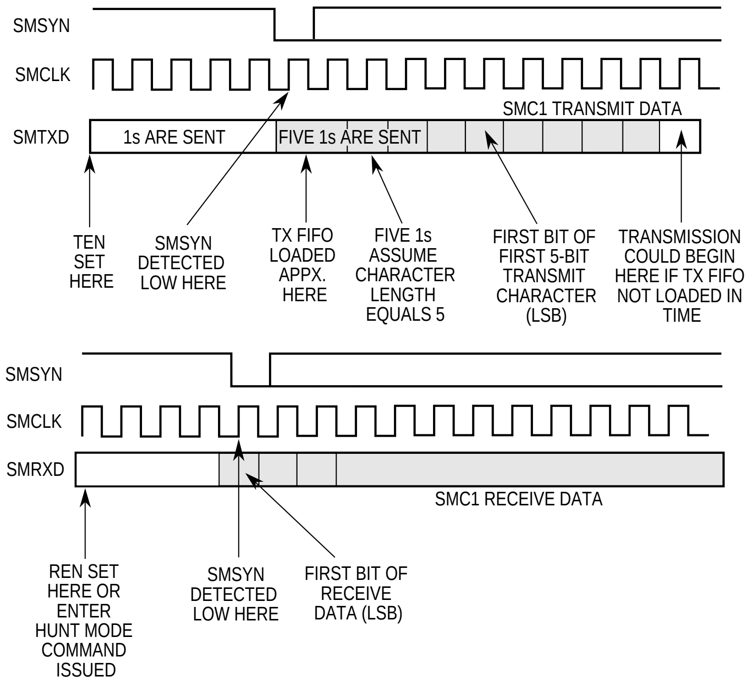
**NOTE**

> Regardless of whether the transmitter or receiver uses the SMSYN signal, it must make glitch-free transitions from high to low or low to high. Glitches on SMSYN can cause errant behavior of the SMC.

Once the REN bit is set in the SMCMR, the first rising edge of SMCLK that detects the SMSYN pin as low causes the SMC receiver to achieve synchronization. Data starts being received (latched) on the same rising edge of SMCLK that latched SMSYN. This is the first bit of data received. The receiver never loses synchronization again, regardless of the state of SMSYN, until the REN bit is cleared by the user.

Once the TEN bit is set in the SMCMR, the first rising edge of SMCLK that detects the SMSYN pin as low causes the SMC transmitter to achieve synchronization. The SMC transmitter begins transmitting ones asynchronously from the falling edge of SMSYN. After one character of ones is transmitted, if the transmit FIFO is loaded (the Tx BD is ready with data), data starts being transmitted on the next falling edge of SMCLK after one character of ones is transmitted. If the transmit FIFO is loaded at some later time, the data begins transmission after some multiple number of all-ones characters is transmitted.

The transmitter never loses synchronization again, regardless of the state of SMSYN, until the TEN bit is cleared by the user or the ENTER HUNT MODE command is issued.


SMSYN
SMCLK
SMC1 TRANSMIT DATA
SMTXD
1s ARE SENT
FIVE 1s ARE SENT
TEN SET HERE
SMSYN DETECTED LOW HERE
TX FIFO LOADED APPX. HERE
FIVE 1s ASSUME CHARACTER LENGTH EQUALS 5
FIRST BIT OF FIRST 5-BIT TRANSMIT CHARACTER (LSB)
TRANSMISSION COULD BEGIN HERE IF TX FIFO NOT LOADED IN TIME
SMSYN
SMCLK
SMRXD
SMC1 RECEIVE DATA
REN SET HERE OR ENTER HUNT MODE COMMAND ISSUED
SMSYN DETECTED LOW HERE
FIRST BIT OF RECEIVE DATA (LSB)

NOTES:
1. SMCLK is an internal clock derived from an external CLKPIN or a baud rate generator.
2. This example shows the SMC receiver and transmitter enabled separately. If the REN and TEN bits were set at the same time, a single falling edge of SMSYN would synchronize both.

**Figure 16-108. Synchronization with the SMSYNx Pin**

If both the REN and TEN bits are set in the SMCMR, the first falling edge of the SMSYN pin causes both the transmitter and receiver to achieve synchronization. To resynchronize the transmitter, the SMC transmitter can be disabled and reenabled and the SMSYN pin can be used again to resynchronize the transmitter itself. Refer to **Section 16.15.5 Disabling the SMCs On-the-Fly** for a description of how to safely disable and reenable the SMC (simply clearing TEN and setting TEN may not be sufficient). The receiver can be resynchronized in a similar fashion.

**16.15.10.7 USING THE TSA FOR SYNCHRONIZATION.** The TSA offers a method to synchronize the SMC channel internally without using the SMSYN pin. This behavior is similar to that of the SMSYN pin, except that the synchronization event is not the falling edge of the SMSYN pin, but rather the first time-slot for this SMC receiver/transmitter following the frame sync indication. Refer to **Section 16.12 Serial Interface with Time-Slot Assigner** for further information on configuring time-slots for the SMCs and SCCs.

The TSA allows the SMC receiver and transmitter to be enabled simultaneously and synchronized separately, a capability not provided by the SMSYN pin. Refer to Figure 16-109 for an example of synchronization using the TSA.

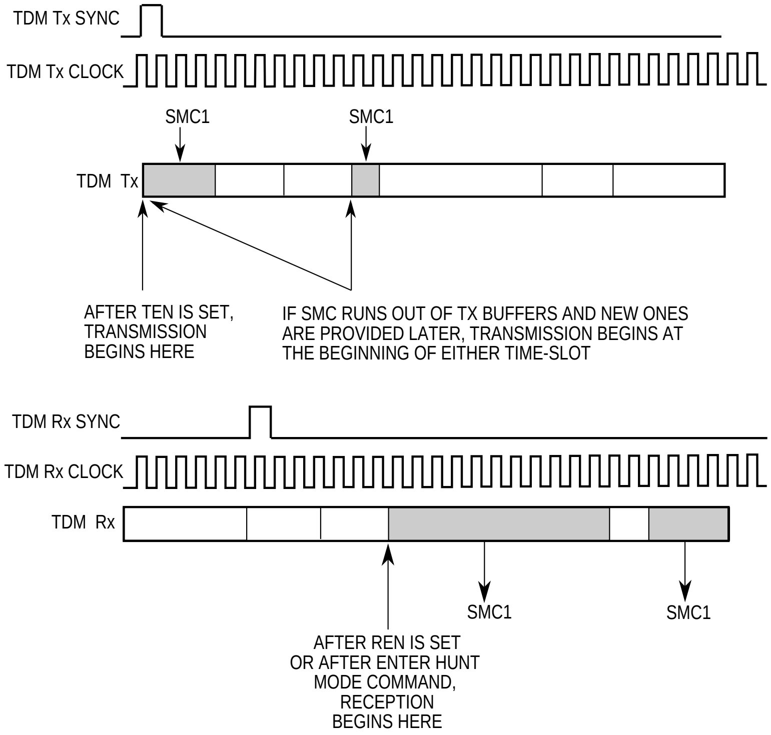


**Figure 16-109. Synchronization with the TSA**

Once the REN bit is set in the SMCMR, the first time-slot after frame sync causes the SMC receiver to achieve synchronization. Data is received immediately, but only during the defined receive time-slots. The receiver continues receiving data during its defined time-slots until the REN bit is cleared by the user. If the ENTER HUNT MODE command is executed, the receiver loses synchronization, closes the current buffer, and resynchronizes to the first time-slot after the frame sync.

Once the TEN bit is set in SMCMR, the SMC waits for the transmit FIFO to be loaded before attempting to achieve synchronization. Once the transmit FIFO is loaded, synchronization and transmission begins according to the following conditions. If a buffer is made ready when the SMC2 is enabled, then the first byte will be placed in time-slot 1 if CLSN is set to 8 and slot 2 if CLSN is set to 16. If a buffer has it’s SMC enabled, then the first byte in the next buffer can appear in any time-slot associated with this channel. If a buffer is ended with the L-bit is 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 transmit buffer is provided later, idles are transmitted during the gap between data buffers and data transmission from the later data buffer begins at the beginning of an SMC time-slot, but not necessarily the first time-slot after the frame sync. Thus, if the user prefers to maintain a certain bit alignment beginning with the first time-slot, they should always make sure that at least one Tx BD is always ready and that no underruns occur. Otherwise, the SMC transmitter should be disabled and reenabled. Refer to **Section 16.15.5 Disabling the SMCs On-the-Fly** for a description of how to safely disable and reenable the SMC (simply clearing TEN and setting TEN may not be sufficient).

**16.15.10.8 COMMAND SET.** The following transmit and receive commands are issued to the CPCR.

#### 16.15.10.8.1 Transmit Commands.

**STOP TRANSMIT**

After the hardware or software is reset and the channel is enabled in the SMC mode register, the channel is in the transmit enable mode and starts polling the first BD in the table. This command disables the transmission of frames on the transmit channel. If this command is received by the transparent controller during frame transmission, transmission of that buffer is aborted after the contents of the FIFO are transmitted (up to 2 characters). The TBPTR is not advanced to the next BD, no new BD is accessed, and no new buffers are transmitted for this channel. The transmitter sends idles until the RESTART TRANSMIT command is issued.

**RESTART TRANSMIT**

This command is used to begin or resume transmission from the current TBPTR in the channel Tx BD table. When this command is received by the channel, it starts polling the R-bit in this BD. This command is expected by the SMC after a STOP TRANSMIT command and the disabling of the channel in it’s mode register or after a transmitter error (underrun) occurs.

**INIT TX PARAMETERS**

This command initializes all the transmit parameters in this serial channel parameter RAM to their reset state and should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the transmit and receive parameters.

#### 16.15.10.8.2 Receive Commands.

**ENTER HUNT MODE**

This command forces the SMC to close the current receive BD 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 in the process of receiving data, the buffer is not closed. Additionally, this command causes the receiver to wait for a resynchronization before further reception continues.

### CLOSE Rx BD

This command is used to force the SMC to close the current receive BD if it is being used and to use the next BD in the list for any subsequently received data. If the SMC is not in the process of receiving data, no action is taken by this command.

### INIT RX PARAMETERS

Initializes all the receive parameters in this serial channel parameter RAM to their reset state. This command should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the receive and transmit parameters.

**16.15.10.9 SMC TRANSPARENT ERROR-HANDLING PROCEDURE.** The SMC reports message reception and transmission error conditions using the channel BDs and the SMC event register.

### Underrun Error

When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the UN bit in the BD, and generates the TXE interrupt if it is enabled. The channel resumes transmission after receiving the RESTART TRANSMIT command. Underrun cannot occur between frames.

### Overrun Error

The SMC maintains an internal FIFO for receiving data (shift register plus data register). The CP begins programming the SDMA channel (if the data buffer is in external memory) when the first character is received into the FIFO. If a FIFO overrun occurs, the SMC writes the received data character to the internal FIFO over the previously received character. The previous character and it's status bits are lost. Following this, the channel closes the buffer, sets the OV bit in the BD, and generates the RX interrupt if it is enabled. Reception then continues normally.

**16.15.10.10 SMC TRANSPARENT MODE REGISTER.** The operating mode of an SMC is defined by the SMCMR. The SMCMR is a 16-bit, memory-mapped, read/write register cleared at reset. The function of bits 8–15 is common to each SMC protocol, but bits 0–7 vary according to the protocol selected by the SM bits.

SMCMR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RES | CLEN | | | | RES | BS | REVD | RES | | SM | | DM | | TEN | REN |
| 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 | A82 (SMCMR1), A92 (SMCMR2) | | | | | | | | | | | | | | | |

Bit 0—Reserved

CLEN—Character Length

CLEN is programmed with a value from 3 to 15 to obtain 4 to 16 bits per character. If the character length is less than 8 bits, the MSBs of the byte in buffer memory are not used on transmit and are written with zeros on receive. On the other hand, if the character length is more than 8 bits, but less than 16 bits, the MSBs of the half-word in buffer memory are not used on transmit and are written with zeros on receive.

**NOTE**

The values 0 to 2 should not be written to CLEN or erratic behavior results. Larger character lengths increase the potential performance of the SMC channel and lower the performance impact on other channels. For instance, the use of 16-bit characters, rather than 8-bit characters is encouraged if 16-bit characters are acceptable in the end application.

Bit 5—Reserved

BS—Byte Sequence

The byte sequence bit controls the sequence of byte transmission if the REVD bit is set for character length greater than 8 bits. It should be set to zero to maintain behavior compatibility with the MC68360 QUICC.

0 = Normal mode (should be selected if character length is less than or equal to 8 bits.
1 = Transmit lower address byte first.

REVD—Reverse Data

0 = Normal mode.
1 = Reverse the character bit order; the MSB is transmitted first.

Bits 8–9—Reserved

SM—SMC Mode

00 = GCI or SCIT support.
01 = Reserved.
10 = UART.
11 = Totally transparent operation (must be selected for SMC transparent operation).

DM—Diagnostic Mode

00 = Normal operation.
01 = Local loopback mode.
10 = Echo mode.
11 = Reserved.

TEN—SMC Transmit Enable

0 = SMC transmitter disabled.
1 = SMC transmitter enabled.

**NOTE**

> Once the SMC transmit enable bit is cleared, the bit must not be reenabled for at least three serial clocks.

REN—SMC Receive Enable

0 = SMC receiver disabled.
1 = SMC receiver enabled.

**16.15.10.11 SMC TRANSPARENT RECEIVE BUFFER DESCRIPTOR.** The CP reports information about the received data for each buffer using Rx BDs and closes the current buffer, generates a maskable interrupt, and starts to receive data into the next buffer after one of the following events occurs:

1. An overrun error is detected.
2. A full receive buffer is detected.
3. The ENTER HUNT MODE command is issued.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **E** | RES | **W** | **I** | RESERVED | | **CM** | RESERVED | | | | | | | OV | RES |
| **OFFSET + 2** | DATA LENGTH | | | | | | | | | | | | | | | |
| **OFFSET + 4** | **RX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

E—Empty

0 = The data buffer associated with this Rx BD is filled with received data or data reception has been aborted due to an error condition. The CPU core is free to examine or write to any fields of this Rx BD. The CP does not use this BD as long as the E-bit is zero.
1 = The data buffer associated with this BD is empty or reception is currently in progress. This Rx BD and it's associated receive buffer are owned by the CP. Once the E-bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Rx BD table.
1 = This is the last BD in the Rx BD 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 Rx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is filled.

1 = The RX bit in the event register is set when this buffer is completely filled by the CP, indicating the need for the CPU core to process the buffer. The RX bit can cause an interrupt if it is enabled.

Bits 4–5—Reserved

CM—Continuous Mode

0 = Normal operation.

1 = The E-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this BD. However, the E-bit is cleared if an error occurs during reception, regardless of the CM bit.

Bits 7–13—Reserved

The following status bit is written by the CP after the received data is placed into the associated data buffer.

OV—Overrun

A receiver overrun occurs during message reception.

Bit 15—Reserved

Data Length

The data length is the number of octets that the CP writes into this BD data buffer. It is written only once by the CP as the buffer is closed.

**NOTE**

> The actual amount of memory allocated for this buffer should be greater than or equal to MRBLR.

Rx Data Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, must be even. The buffer can reside in either internal or external memory.

**16.15.10.12 SMC TRANSPARENT TRANSMIT BUFFER DESCRIPTOR.** Data is presented to the CP for transmission on an SMC channel by arranging it in buffers referenced by the channel Tx BD table. The CP confirms transmission or indicates error conditions using the BDs to inform the processor that the buffers have been serviced.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **R** | RES | **W** | **I** | L | RES | **CM** | RESERVED | | | | | | | UN | RES |
| OFFSET + 2 | **DATA LENGTH** | | | | | | | | | | | | | | | |
| OFFSET + 4 | **TX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission and the user is free to manipulate it or it's associated data buffer. The CP clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which is prepared for transmission by the user, is not transmitted yet or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Tx BD table.

1 = This is the last BD in the Tx BD table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to in the table. The number of Tx BDs in this table is programmable and determined by theW bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = The TX or TXE bit in the event register is set when this buffer is serviced. TX and TXE can cause interrupts if they are enabled.

L— Last in Message

0 = The last byte in the buffer is not the last byte in the transmitted transparent frame. Data from the next transmit buffer (if ready) is transmitted immediately following the last byte of this buffer.

1 = The last byte in this buffer is the last byte in the transmitted transparent frame. After this buffer is transmitted, the transmitter requires synchronization before the next buffer is transmitted.

Bit 5—Reserved

CM—Continuous Mode

0 = Normal operation.

1 = The R-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this BD. However, the R-bit will be cleared if an error occurs during transmission, regardless of the CM bit.

Bits 7–13—Reserved

UN—Underrun

The SMC encountered a transmitter underrun condition while transmitting the associated data buffer.

Bit 15—Reserved

Data Length

The data length is the number of octets that the CP should transmit from this BD data buffer and this value is never modified by the CP. The data length can be even or odd; however, 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 since the three 9-bit characters occupy three half-words in memory (the 9 LSBs of each half-word).

Tx Data Buffer Pointer

The transmit buffer pointer, which always points to the first byte of the associated data buffer, can be even or odd (unless the character length is greater than 8 bits, then 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.

**16.15.10.13 SMC TRANSPARENT EVENT REGISTER.** The SMCE register is referred to as the SMC transparent event register when the SMC is programmed for transparent mode. It is an 8-bit register used to generate interrupts and report events recognized by the SMC channel. On recognition of an event, the SMC controller sets the corresponding bit in the SMCE register. Interrupts generated by this register can be masked in the SMC mask register.

The SMCE register is a memory-mapped register that can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset.

SMCE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | TXE | RES | BSY | TX | RX |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | A86 (SMCE1), A96 (SMCE2) | | | | | | | |

Bits 0–2—Reserved

TXE—Tx Error
An underrun error occurs on the transmitter channel.

Bit 4—Reserved

BSY—Busy Condition
A character is received and discarded due to a lack of buffers. Reception begins after a new buffer is provided. The user might prefer to execute an ENTER HUNT MODE command to cause the receiver to wait for resynchronization.

TX—Tx Buffer
A buffer is transmitted. If the L-bit of the Tx BD is set, this bit is set when the last data character starts being transmitted and the user must wait one character time to be sure that the data is completely sent over the transmit pin. If the L-bit of the Tx BD is cleared, this bit is set when the last data character is written to the transmit FIFO and the user must wait two character times to be sure that the data is completely sent over the transmit pin.

RX—Rx Buffer
A buffer is received on the SMC channel and it’s associated Rx BD is now closed. This bit is set after the last character is written to the buffer.

**16.15.10.14 SMC TRANSPARENT MASK REGISTER.** The SMCM is referred to as the SMC transparent mask register when the SMC is operating in transparent mode. It is an 8-bit read/write register that has the same bit format as the transparent event register. If a bit in the SMCM register is a 1, the corresponding interrupt in the transparent event register is enabled. If the bit is zero, the corresponding interrupt in the transparent event register is masked. This register is cleared at reset.

### 16.15.11 SMC Transparent NMSI Example

The following list is an initialization sequence for operation of the SMC1 transparent channel over it's own set of pins. The transmit and receive clocks are provided from the CLK3 pin (no baud rate generator is used) and the SMSYNx pin is used to obtain synchronization. The SMC UART example shows an example of configuring the baud rate generator.

1. Configure the port B pins to enable the SMTXD1, SMRXD1, and SMSYN1. Write PBPAR bits 25, 24, and 23 with ones and then PBDIR and PBODR bits 25, 24, and 23 with zeros.
2. Configure the port A pins to enable CLK3. Write PAPAR bit 5 with a one and PADIR bit 5 with a zero. The other functions of this pin are the timers or the TSA. These alternate functions cannot be used on this pin.
3. Connect the CLK3 clock to SMC1 using the SI. Write the SMC1 bit in the SIMODE register with a 0 and the SMC1CS bits in the SIMODE register with 110.
4. Write RBASE and TBASE in the SMC parameter RAM to point to the Rx BD and Tx BD in the dual-port RAM. Assuming one Rx BD at the beginning of the dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.
5. Program the CPCR to execute the INIT RX and TX PARAMS commands. Write $0091 to the CPCR.
6. Write $0001 to the SDCR to initialize the SDMA configuration register.
7. Write RFCR and TFCR with $18 for normal operation.
8. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = $0010.
9. Initialize the Rx BD and assume the Rx data buffer is at $00001000 in main memory. Write $B000 to Rx_BD_Status, $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
10. Initialize the Tx BD and assume the Tx data buffer is at $00002000 in main memory and contains five 8-bit characters. Write $B000 to Tx_BD_Status, $0005 to Tx_BD_Length, and $00002000 to Tx_BD_Pointer.
11. Write $FF to the SMCE register to clear any previous events.
12. Write $13 to the SMCM register to enable all possible SMC interrupts.
13. Write $00000010 to the CIMR to allow SMC1 to generate a system interrupt. The CICR should also be initialized.
14. Write $3830 to the SMCMR to configure 8-bit characters, unreversed data, and normal operation (not loopback). Notice that the transmitter and receiver have not been enabled yet.
15. Write $3833 to the SMCMR to enable the SMC transmitter and receiver. This additional write ensures that the TEN and REN bits are enabled last.

**NOTE**

> After 5 bytes are transmitted, the Tx BD is closed. Additionally, the receive buffer is closed after 16 bytes are received. Any received data beyond 16 bytes causes a busy (out-of-buffers) condition since only one Rx BD is prepared.

### 16.15.12 SMC Transparent TSA Example

The following list is an initialization sequence for operation of the SMC1 transparent channel over the TSA. It is assumed that the TSA and the TDM pins already have been set up to route time-slot data to the SMC transmitter and receiver. Refer to **Section 16.12 Serial Interface with Time-Slot Assigner** for examples of how to configure the TSA. The transmit and receive clocks and synchronization signals are provided internally from the TSA.

1. Write RBASE and TBASE in the SMC parameter RAM to point to the Rx BD and Tx BD in the dual-port RAM. Assuming one Rx BD at the beginning of the dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.
2. Program the CPCR to execute the INIT TX and RX PARAMS commands. Write $0091 to the CPCR.
3. Write $0001 to the SDCR to initialize the SDMA configuration register.
4. Write RFCR and TFCR with $18 for normal operation.
5. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = $0010.
6. Initialize the Rx BD and assume the Rx data buffer is at $00001000 in main memory. Write $B000 to Rx_BD_Status, $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
7. Initialize the Tx BD and assume the Tx data buffer is at $00002000 in main memory and contains five 8-bit characters. Write $B000 to Tx_BD_Status, $0005 to Tx_BD_Length, and $00002000 to Tx_BD_Pointer.
8. Write $FF to the SMCE register to clear any previous events.
9. Write $13 to the SMCM register to enable all possible SMC interrupts.
10. Write $00000010 to the CIMR so that SMC1 can generate a system interrupt. The CICR should also be initialized.
11. Write $3830 to the SMCMR to configure 8-bit characters, unreversed data, and normal operation (not loopback). Notice that the transmitter and receiver are not enabled yet.
12. Write $3833 to the SMCMR to enable the SMC transmitter and receiver. This additional write ensures that the TEN and REN bits are enabled last.

### 16.15.13 SMC Interrupt Handling

The following list describes what normally occurs within an SMC interrupt handler.

1. Once an interrupt occurs, read the SMCE register to identify the sources causing the interrupts. Normally, the SMCE bits would be cleared at this time.
2. Process the Tx BD to reuse it if the TX bit is set in the SMCE register. Extract data from the Rx BD if the RX bit is set in the SMCE register. To transmit another buffer, simply set the Tx BD R-bit.
3. Clear the SMC1 bit in the CISR.
4. Execute the RFI instruction.

### 16.15.14 SMC as a GCI Controller

The SMC can be used to control the C/I and to monitor channels of the GCI frame. When using the SCIT configuration of GCI, one SMC can handle SCIT channel 0, and the other SMC can handle SCIT channel 1. The important features of the GCI controller are as follows:

- Each SMC channel 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 SCIT channels 0 and 1
- Full-duplex operation
- Local loopback and echo capability for testing

To use the SMC GCI channels properly, the TSA in the SI must be configured to route the monitor and C/I channels to the preferred SMC. Refer to **Section 16.12 Serial Interface with Time-Slot Assigner** for more details on how to program this configuration. The following SMC discussion assumes that this time-slot routing is programmed properly.

**16.15.14.1 SMC GCI MEMORY MAP.** The GCI parameter RAM area begins at the same offset from each SMC base area. The SMC GCI mode has a very different set of parameter RAM than the SMC UART or SMC transparent modes. In the SMC GCI mode, the general-purpose parameter RAM contains the BDs, rather than pointers, to the BDs. Contrast Table 16-34 with Table 16-31 to see these differences. Additionally, the SMC in GCI mode contains no protocol-specific parameter RAM.

Table 16-34. SMC GCI Parameter RAM

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SMC Base + 00 | **M_RxBD** | Half-word | Monitor Channel Rx BD |
| SMC Base + 02 | **M_TxBD** | Half-word | Monitor Channel Tx BD |
| SMC Base + 04 | **CI_RxBD** | Half-word | C/I Channel Rx BD |
| SMC Base + 06 | **CI_TxBD** | Half-word | C/I Channel Tx BD |
| SMC Base + 08 | Temp1 | Half-word | |
| SMCBase + 0A | Temp2 | Half-word | |
| SMC Base + 0C | Temp3 | Half-word | |
| SMC Base + 0E | Temp4 | Half-word | |

NOTE: Items in bold must be initialized by the user.
SMC base = IMMR + 1E80 (SMC1), 1F80 (SMC2).

**16.15.14.1.1 SMC Monitor Channel Transmission.** The monitor channel 0 is used for data exchange with a layer 1 device (reading and writing internal registers and transferring of the S and Q bits). The monitor channel 1 is used for programming and controlling voice/ data modules such as CODECs. The CPU core writes the data byte into the SMC Tx BD. The SMC transmits the data on the monitor channel and handles the A and E control bits according to the GCI monitor channel protocol. The user can issue the TIMEOUT command to solve deadlocks in case of errors in the A and E bit states on the data line.

**16.15.14.1.2 SMC Monitor Channel Reception.** The SMC receives the data and handles the A and E control bits according to the GCI monitor channel protocol. When a received data byte is stored by the CP in the SMC Rx BD, a maskable interrupt is generated. The user can issue the TRANSMIT ABORT REQUEST command and the MPC821 transmits an abort request on the E-bit.

**16.15.14.2 SMC C/I CHANNEL HANDLING.** The C/I channel (in SCIT configuration, C/I channel 0) is used to control the layer 1 device. The layer 2 device in the TE sends commands and receives indication to or from the upstream layer 1 device via 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).

**16.15.14.2.1 SMC C/I Channel Transmission.** The CPU core writes the data byte into the SMC C/I Tx BD and the SMC transmits the data continuously on the C/I channel to the physical layer device.

**16.15.14.2.2 SMC C/I Channel Reception.** The SMC receiver continuously monitors the C/I channel and when a change in the data is recognized and this value is received in two successive frames, it is interpreted as valid data. This is referred to as the double last-look method. The received data byte is stored by the CP in the C/I Rx BD and a maskable interrupt is generated. If the SMC is configured to support SCIT channel 1, the double last-look method is not used.

**16.15.14.3 SMC COMMANDS IN GCI MODE.** The following commands are issued to the CPCR.

### INIT TX and RX PARAMETERS

This command initializes the transmit and receive parameters in the parameter RAM to their reset state and it is especially useful when switching protocols on a given serial channel.

### TRANSMIT ABORT REQUEST

This receiver command can be issued when the MPC821 implements the monitor channel protocol. When it is issued, the MPC821 sends an abort request on the A-bit.

### TIMEOUT

This transmitter command can be issued when the MPC821 implements the monitor channel protocol and it is issued because the device is not responding or GCI A-bit errors are detected. The MPC821 sends an abort request on the E-bit at the time this command is issued.

**16.15.14.4 SMC GCI MODE REGISTER.** The operating mode of an SMC is defined by the SMCMR, which is a 16-bit, memory-mapped, read/write register cleared at reset. The functions of bits 8–15 are common to each SMC protocol, but bits 0–7 vary according to the protocol selected by the SM bits.

SMCMR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RES | CLEN | | | | ME | RES | C# | RES | | SM | | DM | | TEN | REN |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | A82 (SMCMR1), A92 (SMCMR2) | | | | | | | | | | | | | | | |

Bit 0—Reserved

These bits should be cleared by the user.

CLEN—Character Length

This value is used to define the total number of bits in the C/I and monitor channels of the SCIT channel 0 or channel 1. CLEN ranges from 0 to 15 and specifies values from 1 to 16 bits. CLEN should be written with 13 for the SCIT channel 0 or GCI (8 data bits, plus A and E bits, plus 4 C/I bits = 14 bits) or with 15 for the SCIT channel 1 (8 data, bits, plus A and E bits, plus 6 C/I bits = 16 bits).

ME—Monitor Enable

0 = The SMC does not support the monitor channel.

1 = The SMC supports the monitor channel with the transparent or monitor channel protocol as defined in the MP bit.

C#—SCIT Channel Number

0 = SCIT channel 0.
1 = SCIT channel 1 (required for Siemens ARCOFI and SGS S/T chips).

Bits 8–9—Reserved

These bits should be cleared by the user.

SM—SMC Mode

00 = GCI or SCIT support (required for SMC GCI or SCIT operation).
01 = Reserved.
10 = UART.
11 = Totally transparent operation.

DM—Diagnostic Mode

00 = Normal operation.
01 = Local loopback mode.
10 = Echo mode.
11 = Reserved.

TEN—SMC Transmit Enable

0 = SMC transmitter disabled.
1 = SMC transmitter enabled.

REN—SMC Receive Enable

0 = SMC receiver disabled.
1 = SMC receiver enabled.

**16.15.14.5 SMC MONITOR CHANNEL RX BD.** The CP reports information about the monitor channel receive byte using this BD.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **E** | L | **RE** | **MS** | RESERVED | | | | DATA | | | | | | | |

NOTE: Items in bold must be initialized by the user.

E—Empty

0 = This bit is cleared by the CP to indicate that the data byte associated with this BD is now available to the CPU core.
1 = This bit is set by the CPU core to indicate that the data byte associated with this BD has been read.

When the SMC implements the monitor channel protocol, the SMC waits until this bit is set by the CPU core before acknowledging the monitor channel data. In the transparent mode, additional received data bytes are discarded until the E-bit is set by the CPU core.

L—Last (EOM)

This bit is only valid when the SMC implements the monitor channel protocol and is set when the EOM indication is received on the E-bit.

**NOTE**

When this bit is set, the data byte is invalid.

ER—Error Condition

This bit is only valid when the SMC implements the monitor channel protocol and is set when an error condition occurs on the monitor channel protocol. A new byte is transmitted before the SMC acknowledges the previous byte.

MS—Data Mismatch

This bit is only valid when the SMC implements the monitor channel protocol. It is set when two different consecutive bytes are received and it is cleared when the last two consecutive bytes match. The SMC waits for the reception of two identical consecutive bytes before writing new data to the Rx BD.

Bits 4–7—Reserved

These bits should be cleared by the user.

DATA—Data Field

The data field contains the monitor channel data byte that the SMC received.

**16.15.14.6 SMC MONITOR CHANNEL TX BD.** Using this BD, the CP reports the information about the monitor channel transmit byte.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **R** | L | **AR** | **RESERVED** | | | | | DATA | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0 = This bit is cleared by the CP after transmission. The Tx BD is now available to the CPU core.

1 = This bit is set by the CPU core to indicate that the data byte associated with this BD is ready for transmission.

L—Last (EOM)

This bit is only valid when the SMC implements the monitor channel protocol. When it is set, the SMC first transmits the buffer data and then transmits the EOM indication on the E-bit.

AR—Abort Request

This bit is only valid when the SMC implements the monitor channel protocol and it is set by the SMC when an abort request is received on the A-bit. The SMC transmitter transmits the EOM on the E-bit after an abort request is received.

Bits 3–7—Reserved

These bits should be cleared by the user.

DATA—Data Field

The data field contains the data to be transmitted by the SMC on the monitor channel.

**16.15.14.7 SMC C/I CHANNEL RECEIVE BUFFER DESCRIPTOR.** The CP reports information about the C/I channel receive byte using this BD.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | E | RES | | | | | | | C/I DATA | | | | | | RES | |

E—Empty

0 = This bit is cleared by the CP to indicate that the data byte associated with this BD is now available to the CPU core.

1 = This bit is set by the CPU core to indicate that the data byte associated with this BD has been read.

**NOTE**

Additional data received is discarded until the E-bit is set.

Bits 1–7—Reserved

These bits should be cleared by the user.

C/I DATA—Command/Indication Data Bits

C/I DATA is a 4-bit data field for C/I channel 0 and a 6-bit data field for C/I channel 1. It contains the data received from the C/I channel. For C/I channel 0, bits 10–13 contain the 4-bit data field and bits 8 and 9 are always written with zeros. For C/I channel 1, bits 8–13 contain the 6-bit data field.

Bits 14–15—Reserved

**16.15.14.8 SMC C/I CHANNEL TRANSMIT BUFFER DESCRIPTOR.** Using the BD, the CP reports information about the C/I channel transmit byte.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | R | RESERVED | | | | | | | C/I DATA | | | | | | RES | |

R—Ready

0 = This bit is cleared by the CP after transmission to indicate that the BD is now available to the CPU core.

1 = This bit is set by the CPU core to indicate that the data associated with this BD is ready for transmission.

Bits 1–7—Reserved

These bits should be cleared by the user.

C/I DATA—Command/Indication Data Bits

C/I DATA is a 4-bit data field for C/I channel 0 and a 6-bit data field for C/I channel 1. It contains the data to be transmitted onto the C/I channel. For C/I channel 0, bits 10–13 contain the 4-bit data field and bits 8 and 9 are always written with zeros. For C/I channel 1, bits 8–13 contain the 6-bit data field.

**16.15.14.9 SMC EVENT REGISTER.** The SMCE is an 8-bit register used to generate interrupts and report events recognized by the SMC channel. On recognition of an event, the SMC sets it's corresponding bit in this register. Interrupts generated by this register can be masked in the SMC mode register. The SMCE is a memory-mapped register that can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) more than one bit can be cleared at a time. All unmasked bits must be cleared before the CP clears the internal interrupt request to the CPM interrupt controller. This register is cleared at reset.

SMCE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | CTXB | CRXB | MTXB | MRXB |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | A86 (SMCE1), A96 (SMCE2) | | | | | | | |

Bits 0–3—Reserved

CTXB—C/I Channel Buffer Transmitted

The C/I transmit buffer is now empty.

CRXB—C/I Channel Buffer Received
The C/I receive buffer is full.

MTXB—Monitor Channel Buffer Transmitted
The monitor transmit buffer is now empty.

MRXB—Monitor Channel Buffer Received
The monitor receive buffer is full.

**16.15.14.10 SMC MASK REGISTER.** The SMCM is an 8-bit, memory-mapped, read/write register that has the same bit format as the SMC event register. If a bit in the SMCM is a 1, the corresponding interrupt in the SMC event register is enabled. If the bit is zero, the corresponding interrupt in the SMC event register is masked. The SMCM is clear at reset.

## 16.16 SERIAL PERIPHERAL INTERFACE

The serial peripheral interface (SPI) allows the MPC821 to exchange data between other MPC821 chips, the MC68360, the MC68302, the M68HC11 and M68HC05 microcontroller families, and a number of peripheral devices such as EEPROMs, real-time clock devices, A/D converters, and ISDN devices.

### 16.16.1 Overview

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 transmitted and received simultaneously. Refer to Figure 16-110 for the SPI block diagram.

The SPI receiver and transmitter are double-buffered as illustrated in the block diagram and this corresponds to an effective FIFO size (latency) of 2 characters. The MPC821 SPI most-significant bit is shifted out first. When the SPI is not enabled in the SPMODE, it consumes minimal power.

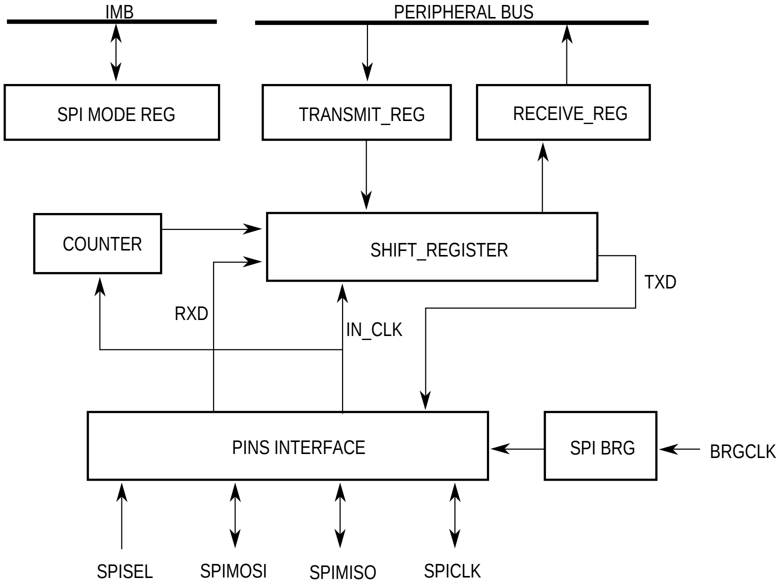


**Figure 16-110. SPI Block Diagram**

## 16.16.2 Features

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

- Four-wire interface (SPIMOSI, SPIMISO, SPICLK, and SPISEL)
- Full-duplex operation
- Works with data characters from 4 to 16 bits in length
- Supports back-to-back character transmission and reception
- Master or slave SPI modes supported
- Multimaster environment support
- Continuous transfer mode for auto scanning of a peripheral
- Supports clock rates up to 6.25 MHz in master mode and up to 12.5 MHz in slave mode (assuming a 25-MHz system clock)
- Independent programmable baud rate generator
- Programmable clock phase and polarity
- Open-drain output pins support multimaster configuration
- Local loopback capability for testing

### 16.16.3 Serial Peripheral Interface Clocking and Pin Functions

The SPI can be configured as a master for the serial channel (it generates both the enable and clock signals) or as slave (the enable and clock signals are inputs to the SPI). The SPI also supports operation in a multimaster environment. When the SPI is a master, the SPI baud rate generator is used to generate the SPI transmit and receive clocks. The SPI baud rate generator takes it's input from the BRGCLK which is generated in the clock synthesizer of the MPC821. It is specifically for the SPI baud rate generator and the other four baud rate generators in the CPM.

The SPI master-in slave-out (SPIMISO) pin is an input in master mode and an output in slave mode. Likewise, the SPI master-out slave-in (SPIMOSI) pin is an output in master mode and an input in slave mode. The reason the pin names SPIMOSI and SPIMISO change functionality between master and slave mode is to support a multimaster configuration that allows communication from one SPI to any other SPI with the same hardware configuration.

When the SPI is working as a master, SPICLK is the clock output signal that shifts in the received data from the SPIMISO pin and shifts out the transmitted data to the SPIMOSI pin. Additionally, an SPI master device must provide a slave select signal output to enable the SPI slave devices. This can be implemented using one of the MPC821 general-purpose I/O pins. The SPISEL pin should not be asserted while the SPI is working as a master or the SPI indicates an error.

When the SPI is working as a slave, SPICLK is the clock input signal that shifts in the received data from the SPIMOSI pin and shifts out the transmitted data to the SPIMISO pin. The SPISEL pin provided by the MPC821 is the enable input to the SPI slave. When the SPI is working in a multimaster environment, the SPISEL pin is still an input and is used to detect an error condition when more then one master is operating.

SPICLK is a gated clock (the clock only toggles while data is being transferred) and the user can select any of four combinations of SPICLK phase and polarity using two bits in the SPI mode register (SPMODE). The SPI pins can also be configured as open-drain pins to support a multimaster configuration where the same SPI pin can be driven by the MPC821 or an external SPI device.

### 16.16.4 Serial Peripheral Interface Transmit/Receive Process

**16.16.4.1 SPI MASTER MODE.** When the SPI functions in master mode, the SPI transmits a message to the peripheral (SPI slave), which in turn sends back a simultaneous reply. When the MPC821 works with more than one slave, it can use the general-purpose parallel I/O pins to selectively enable different slaves.

To begin the data exchange, the CPU core writes the data to be transmitted into a data buffer, configures a Tx BD with it's R-bit set and configures one or more Rx BDs. The CPU core should then set the STR bit in the SPCOM to start transmission of data. The data begins transmitting once the SDMA channel loads the transmit FIFO with data.

The SPI controller then generates programmable clock pulses on the SPICLK pin for each character and shifts the data out on the SPIMOSI pin. At the same time, the SPI shifts received data in from the SPIMISO pin. This received data is written into a receive buffer using the next available Rx BD. The SPI continues transmitting and receiving characters until the transmit buffer has been completely transmitted or an error has occurred (SPISEL pin unexpectedly asserted). The CP then clears the R and E bits in the Tx BD and Rx BD and issues a maskable interrupt to the CPM interrupt controller.

When multiple Tx BDs are ready for transmission, the Tx BD L-bit determines whether or not the SPI continues transmitting without waiting for the STR bit to be set again. If the L-bit is cleared, the data from the next Tx BD begins transmitting after the transmission of data from the first Tx BD. In most cases, the user should see no delay on the SPIMOSI pin between buffers. If the L-bit is set, transmission stops after data from this Tx BD has completed transmission. In addition, the current Rx BD that is used to receive data is closed after the transmission completes, even if the receive buffer is not full. Thus, the user does not need to provide receive buffers of the same length as the transmit buffers. If the SPI is the only master in a system, then the SPISEL pin can be used as a general-purpose I/O, and the internal SPISEL signal to the SPI is always forced internally inactive, eliminating the possibility of a multimaster error.

**16.16.4.2 SPI SLAVE MODE.** When the SPI functions in slave mode, it receives messages from an SPI master and, in turn, sends back a simultaneous reply. The SPISEL pin must be asserted before receive clocks are recognized and once SPISEL is asserted, the SPICLK pin becomes an input from the master to the slave. SPICLK can be any frequency from DC to the BRGCLK/2 (12.5 MHz for a 25-MHz system).

Before the data exchange, the CPU core writes the data to be transmitted into a data buffer, configures a Tx BD with it's R-bit set, and configures one or more Rx BDs. The CPU core should then set the STR bit in the SPCOM to enable the SPI to prepare the data for transmission and wait for the SPISEL pin to be asserted. Data is shifted out from the slave on the SPIMISO pin and shifted in through the SPIMOSI pin. A maskable interrupt is issued upon complete transmission or reception of a full buffer or after an error occurs (receive overrun, transmit underrun, or out of receive buffers). The SPI then continues reception using the next Rx BD in the ring until it runs out of receive buffers or the SPISEL pin is negated.

Transmission continues until no more data is available for transmission or the SPISEL pin is negated. If the this pin is negated before all the transmit data is transmitted, transmission stops, but the Tx BD remains open. Further transmission from that point continues once the SPISEL pin is reasserted and SPICLK begins toggling. After completing transmission of characters in the Tx DB, the SPI transmits ones if SPISEL is not negated.

**16.16.4.3 SPI MULTIMASTER OPERATION.** The SPI can operate in a multimaster environment in which some SPI devices are connected on the same bus. In this configuration, the SPIMOSI, SPIMISO, and SPICLK pins of all SPIs are connected together and the SPISEL input pins are connected separately. In this environment, only one SPI device can work as a master at a time; all the others must be slaves. When the SPI is configured as a master and it's SPISEL input goes active (low), a multimaster error has occurred since more than one SPI device is currently a bus master. The SPI sets the MME bit in the event register and a maskable interrupt is issued to the CPU core. It also disables the SPI operation and the output drivers of the SPI pins. The CPU core should clear the EN bit the SPMODE register before using the SPI again. After the problems are corrected, the MME bit should be cleared and the SPI should be enabled in the same procedure as after a reset.

**NOTE**

The user should notice that the maximum sustained data rate supported on the SPI is SYSTEMCLK/50. The SPI can transfer a single character at much higher rates (SYSTEMCLK/4 in master mode and SYSTEMCLK/2 in slave mode). If multiple characters are to be transmitted, gaps should be inserted between them so that it will not exceed the maximum sustained data rate.

## 16.16.5 Programming Model

**16.16.5.1 SPI MODE REGISTER.** SPMODE is a read/write register that controls both the SPI operation mode and clock source. The SPMODE register is cleared by a reset.

SPMODE REGISTER

| BITS | 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[0:3] | | | |
| 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 | AA0 | | | | | | | | | | | | | | | |

Bit 0—Reserved

This bit should be cleared by the user.

LOOP—Loop Mode

When set, this bit selects the local loopback operation. The transmitter output is internally connected to the receiver input; the receiver and transmitter operate normally except that the received data is ignored.

0 = Normal operation.
1 = The SPI is in loopback mode.

CI—Clock Invert

The CI bit inverts the SPI clock polarity (refer to Figure 16-111 and Figure 16-112 for details).

0 = The inactive state of SPICLK is low.
1 = The inactive state of SPICLK is high.

CP—Clock Phase

The CP bit selects one of two fundamentally different transfer formats (refer to Figure 16-111 and Figure 16-112 for details).

0 = SPICLK begins toggling at the middle of the data transfer.
1 = SPICLK begins toggling at the beginning of the data transfer.

DIV16—Divide by 16

The DIV16 bit selects the clock source for the SPI baud rate generator when configured as an SPI master. In slave mode, the clock source is the SPICLK pin.

0 = Use the BRGCLK as the input to the SPI baud rate generator.
1 = Use the BRGCLK/16 as the input to the SPI baud rate generator.

REV—Reverse Data

The REV bit determines the receive and transmit character bit order.

0 = Reverse data—LSB of character transmitted and received first.
1 = Normal operation—MSB of character transmitted and received first.

M/S—Master/Slave

The M/S bit configures the SPI to work as a master or slave.

0 = SPI is a slave.
1 = SPI is a master.

EN—Enable SPI

The EN bit enables the SPI operation. Notice that SPIMOSI, SPIMISO, SPICLK, and SPISEL should be configured to connect to the SPI as described in **Section 16.18.8 Port B Registers**. When the EN bit is cleared, the SPI is in a reset state and consumes minimal power (the SPI baud rate generator is not functioning and the input clock is disabled).

0 = SPI is disabled.
1 = SPI is enabled.

**NOTE**

Other bits of the SPMODE register should not be modified while EN is set.

LEN—Character Length

The LEN field specifies how many bits are in a character. The value 0000 corresponds to 1 bit and the value 1111 corresponds to 16 bits. Acceptable values are in the range of 4 to 16 bits inclusive. Programming a value less than 4 bits can cause erratic behavior.

If the LEN value is less than or equal to a byte, there will be LEN number of valid bits in every byte (8 bits) in memory. On the other hand, if the LEN value is greater than a byte, there is LEN number of valid bits in every half-word (16 bits) in memory.

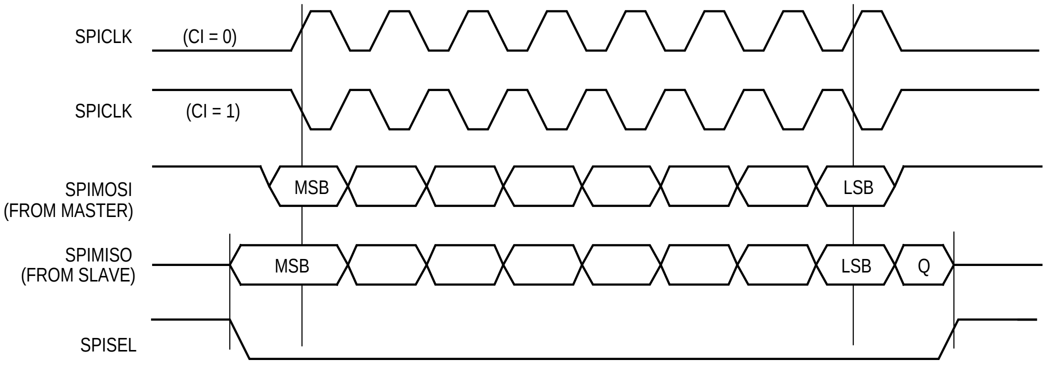
PM[0:3]—Prescale Modulus Select

These four bits specify the divide ratio of the prescale divider in the SPI clock generator. The BRGCLK is divided by 4 * ([PM0–PM3] + 1) giving a clock divide ratio of 4 to 64 and the clock has a 50% duty cycle.

**16.16.5.1.1 SPI Examples With Different LEN Values.** The examples below use LEN to illustrate using the bits described above. To help map the process, let *g* though *v* be binary symbols, *x* indicates a deleted bit, __ indicates original byte boundaries, and _ indicates original nybble (4-bit) boundaries (both are used to aid in readability and to help the user understand the process. Once the data string image is determined, it is always transmitted byte by byte with the LSB first. Let the memory contain the following binary image:

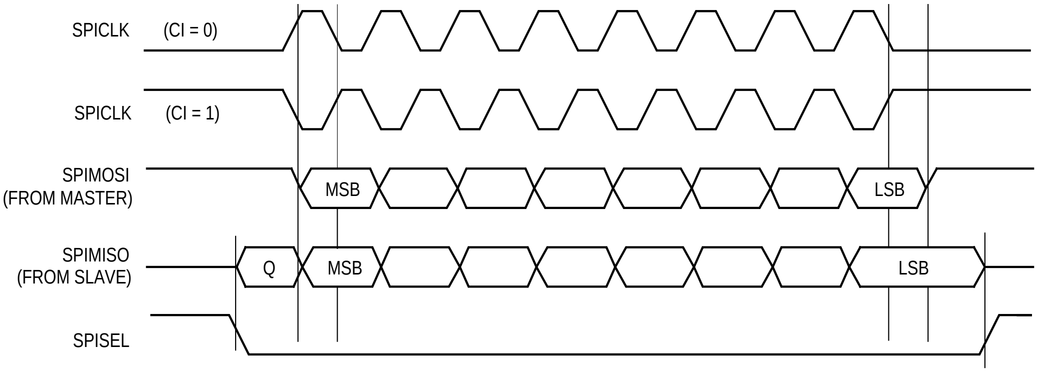
```
             msb         ghij_klmn__opqr_stuv             lsb
Example 1:
with LEN=4 (data size=5), the following data is selected:
             msb         xxxj_klmn__xxxr_stuv             lsb
with REV=0, the data string image is:
             msb         j_klmn__r_stuv                   lsb
the order of the string appearing on the line, a byte at a time is:
                         nmlk_j__vuts_r
with REV=1,the string has each byte reversed
the data string image is:
             msb         nmlk_j__vuts_r                   lsb
the order of the string appearing on the line, one byte at a time is:
                         j_klmn__r_stuv

Example 2:
with LEN=7 (data size=8), the following data is selected:
             msb         ghij_klmn__opqr_stuv             lsb
the data string is selected:
             msb         ghij_klmn__opqr_stuv             lsb
with REV=0, the string transmitted, a byte at a time with lsb first is:
             nmlk_jihg__vuts_rqpo
with REV=1, the string is byte reversed and transmitted, a byte at a time, with
lsb first:
             ghij_klmn__opqr_stuv
```

```
Example 3:
with LEN=0cH(12), (data size=0dH(13)), the following data is selected:
              msb         ghij_klmn__xxxr_stuv             lsb
the data string selected is:
              msb         r_stuv__ghij_klmn                lsb
with REV=0, the string transmitted, a byte at a time with lsb first is:
                          vuts_r__nmlk_jihg
with REV=1, the string is WORD reversed
                          nmlk_jihg__vuts_r
and transmitted at byte at a time with lsb first:
                          ghij_klmn__r_stuv
```



NOTE: Q = Undefined Signal

**Figure 16-111. SPI Transfer Format With CP = 0**



NOTE: Q = Undefined Signal

**Figure 16-112. SPI Transfer Format With CP = 1**

**16.16.5.2 SPI COMMAND REGISTER.** The SPCOM is an 8-bit read/write register that is used to start SPI operation.

SPCOM REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | STR | RESERVED | | | | | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | AAD | | | | | | | |

STR—Start Transmit

When the SPI is configured as a master, setting the STR bit to 1 causes the SPI controller to start the transmission and reception of data to and from the SPI transmit/receive buffers (if configured as ready by the user). When the SPI is configured as a slave, setting the STR bit to 1 when the SPI is idle (between transfers) causes the SPI to load the transmit data register from the SPI transmit buffer and start transmission as soon as the next SPI input clocks and select signal are received.

The STR bit is cleared automatically after one system clock cycle.

Bits 1–7—Reserved.

These bits should be written with zeros by the user.

**16.16.5.3 SPI PARAMETER RAM MEMORY MAP.** The SPI parameter RAM area begins at the SPI base address and a is used for the general SPI parameters. Notice that it is similar to the SCC general-purpose parameter RAM. Refer to Table 16-35 for details.

**Table 16-35. SPI Parameter RAM Memory Map**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| SPI Base + 00 | **RBASE** | Half-word | Rx BD Base Address |
| SPI Base+ 02 | **TBASE** | Half-word | Tx BD Base Address |
| SPI Base+ 04 | **RFCR** | Byte | Rx Function Code |
| SPI Base+ 05 | **TFCR** | Byte | Tx Function Code |
| SPI Base+ 06 | **MRBLR** | Half-word | Maximum Receive Buffer Length |
| SPI Base+ 08 | RSTATE | Word | Rx Internal State |
| SPI Base+ 0C | | Word | Rx Internal Data Pointer |
| SPI Base+ 10 | RBPTR | Half-word | Rx BD Pointer |
| SPI Base+ 12 | | Half-word | Rx Internal Byte Count |
| SPI Base+ 14 | | Word | Rx Temp |
| SPI Base+ 18 | TSTATE | Word | Tx Internal State |
| SPI Base+ 1C | | Word | Tx Internal Data Pointer |
| SPI Base+ 20 | TBPTR | Half-word | Tx BD Pointer |
| SPI Base+ 22 | | Half-word | Tx Internal Byte Count |
| SPI Base+ 24 | | Word | Tx Temp |

NOTE: Items in bold must be initialized by the user.
SPI base = IMMR + 1D80.

Certain parameter RAM values need to be initialized by the user before the SPI is enabled; other values are initialized by the CP. Once initialized, the parameter RAM values do not normally need to be accessed by user software. They should only be modified when no SPI activity is in progress.

**16.16.5.3.1 BD Table Pointer.** The RBASE and TBASE entries define the starting location in the dual-port RAM for the set of BDs for receive and transmit functions of the SPI. This provides a great deal of flexibility in how BDs for an SPI are partitioned. By setting the W-bit in the last BD in each BD list, the user can select how many BDs to allocate for the transmit and receive side of the SPI, but the user must initialize these entries before enabling the SPI. Furthermore, the user should not configure BD tables of the SPI to overlap any other serial channel BDs or erratic operation occurs.

**NOTE**

RBASE and TBASE should contain a value that is divisible by eight.

**16.16.5.3.2 SPI Function Code Registers.** The FC entry contains the value that the user wants to appear on the address type pins (AT0–AT3) when the associated SDMA channel accesses memory. It also controls the byte-ordering convention to be used in the transfers.

RFCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | SPI BASE + 04 | | | | | | | |

Bits 0–2—Reserved

These bits should be set to zero by the user.

BO—Byte Ordering

This bit field should be set by the user to select the required byte ordering of the data buffer.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

TFCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | SPI BASE + 05 | | | | | | | |

Bits 0–2—Reserved

These bits should be set to zero by the user.

BO—Byte Ordering

This bit field should be set by the user to select the required byte ordering of the data buffer. If this bit field is modified on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next BD.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.
01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.
1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

**16.16.5.3.3 Maximum Receive Buffer Length Register.** The SPI has one MRBLR to define the receive buffer length for that SPI and it defines the maximum number of bytes that the MPC821 writes to a receive buffer on that SPI before moving to the next buffer. The MPC821 can write fewer bytes to the buffer than the MRBLR value if a condition such as an error or end-of-frame occurs, but it never writes more bytes than the MRBLR value. It follows, then, that buffers supplied by the user for use by the MPC821 should always be the size of MRBLR (or greater) in length.

The transmit buffers for an SPI are not affected in any way by the value programmed into MRBLR and can be individually chosen to have varying lengths, as needed. The number of bytes to be transmitted is chosen by programming the data length field in the Tx BD.

**NOTE**

MRBLR is not intended to be changed dynamically while an SPI is operating. However, if it is modified in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back), then a dynamic change in receive buffer length can be successfully achieved. This takes place when the CP moves control to the next Rx BD in the table. Thus, a change to MRBLR does not have an immediate effect. To guarantee the exact Rx BD on which the change occurs, the user should only change MRBLR while the SPI receiver is disabled. The MRBLR value should be greater than zero and should be even if the character length of the data is greater than 8 bits.

**16.16.5.3.4 Receiver Buffer Descriptor Pointer.** The RBPTR for each SPI channel points to the next BD 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 this pointer to the value programmed in the RBASE entry. Although in most applications the user should not write RBPTR, it can be modified when the receiver is disabled or when the user is sure that no receive buffer is currently in use.

**16.16.5.3.5 Transmitter Buffer Descriptor Pointer.** The TBPTR for each SPI channel points to the next BD that the transmitter transfers data from when it is in idle state or to the current BD during frame transmission. After a reset or when the end of BD table is reached, the CP initializes this pointer to the value programmed in the TBASE entry. Although in most applications the user should not write TBPTR, it can be modified when the transmitter is disabled or when the user is sure that no transmit buffer is currently in use.

**16.16.5.3.6 Other General Parameters.** These parameters do not need to be accessed by the user in normal operation. They are only listed because they provide helpful information for experienced users and for debugging purposes. Additional parameters are listed in Table 16-35.

The Rx and Tx internal data pointers are updated by the SDMA channels to show the next address in the buffer to be accessed. The Tx internal byte count is a down-count value that is initialized with the Tx BD data length and decremented with every byte read by the SDMA channels. The Rx internal byte count is a down-count value that is initialized with the MRBLR value and decremented with every byte written by the SDMA channels.

**NOTE**

To extract data from a partially full buffer, the CLOSE Rx BD command can be used.

The Rx internal state, Tx internal state, Rx temp, Tx temp, and reserved areas are only for RISC use.

**16.16.5.4 SPI COMMANDS.** The following transmit and receive commands are issued to the CPCR.

**INIT TX PARAMETERS**

This command initializes all transmit parameters in this serial channel parameter RAM to their reset state and should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the transmit and receive parameters.

**CLOSE Rx BD**

This command is used to force the SPI controller to close the current Rx BD, if it is currently being used and to use the next BD for any subsequently received data. If the SPI controller is not in the process of receiving data, no action is taken by this command.

### INIT RX PARAMETERS

This command initializes all the receive parameters in this serial channel parameter RAM to their reset state and should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the receive and transmit parameters.

**16.16.5.5 SPI BUFFER DESCRIPTOR RING.** The data associated with the SPI is stored in buffers, which are referenced by BDs organized in a BD ring located in the dual-port RAM (see Figure 16-113). This ring has the same basic configuration as those used by the SCCs and SMCs.

The BD ring allows the user to define buffers for transmission and buffers for reception and each BD ring forms a circular queue. The CP confirms reception and transmission or indicates error conditions using the BDs to inform the processor that the buffers have been serviced. The actual buffers can reside in either external memory or internal memory and the data buffers can reside in the parameter area of an SCC if it is not enabled.

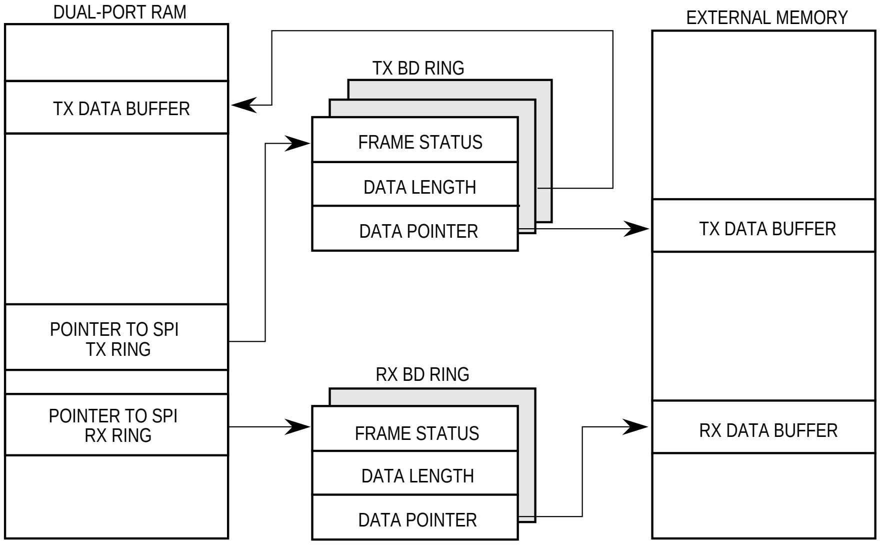


**Figure 16-113. SPI Memory Structure**

**16.16.5.5.1 SPI Receive Buffer Descriptor.** Using Rx BDs, the CP reports information about each buffer of received data and closes the current buffer, generates a maskable interrupt, and starts receiving data in the next buffer once the current buffer is full. Additionally, it closes the buffer when the SPI is configured as a slave and the SPISEL pin goes to an inactive state, indicating that the reception process is terminated.

The first word of the Rx BD contains status and control bits. These bits are prepared by the user before reception and are set by the CP after the buffer has been closed. The second word contains the data length (in bytes) that is received. The third and fourth words contain a pointer that always points to the beginning of the received data buffer.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **E** | RES | **W** | **I** | L | RES | **CM** | RESERVED | | | | | | | OV | ME |
| OFFSET + 2 | DATA LENGTH | | | | | | | | | | | | | | | |
| OFFSET + 4 | **RX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

The following bits should be written by the CPU core before enabling the SPI.

E—Empty

0 = The data buffer associated with this Rx BD is filled with received data or data reception is aborted due to an error condition. The CPU core is free to examine or write to any fields of this Rx BD. The CP does not use this BD as long as the E-bit is zero.

1 = The data buffer associated with this BD is empty or reception is currently in progress. This Rx BD and it's associated receive buffer are owned by the CP. Once the E-bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Rx BD table.

1 = This is the last BD in the Rx BD 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 Rx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 No interrupt is generated after this buffer is filled.

1 The RXB bit in the SPI event register is set when this buffer is completely filled by the CP, indicating the need for the CPU core to process the buffer. The RXB bit can cause an interrupt if it is enabled.

Bit 5—Reserved

CM—Continuous Mode

This bit is valid only when the SPI is configured as a master; it should be written as a zero in slave mode.

0 = Normal operation.
1 = The E-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this BD. This allows continuous reception from an SPI slave into one buffer for auto scanning of a serial A/D peripheral with no CPU overhead.

Bits 7–13—Reserved

The following status bits are written by the SPI after the received data is placed into the associated data buffer.

L—Last

This bit is set by the SPI controller when the buffer is closed due to negation of the SPISEL pin. This only occurs when the SPI is a slave; otherwise, the ME bit is set.

0 = This buffer does not contain the last character of the message.
1 = This buffer contains the last character of the message.

OV—Overrun

A receiver overrun occurs during reception. This error can only occur when the SPI is a slave.

ME—Multimaster Error

This buffer is closed because the SPISEL pin is asserted when the SPI is operating as a master. This indicates a synchronization problem between multiple masters on the SPI bus.

Data Length

Data length is the number of octets that the CP writes into this BD data buffer. It is written once by the CP as the BD is closed.

**NOTE**

The actual amount of memory allocated for this buffer should be greater than or equal to the MRBLR.

Rx Data Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, must be even. The buffer can reside in internal or external memory.

**16.16.5.5.2 SPI Transmit Buffer Descriptor.** Data to be transmitted with the SPI is presented to the CP by arranging it in buffers referenced by the Tx BD ring. The first word of the Tx BD contains status and control bits.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **R** | RES | **W** | **I** | **L** | RES | **CM** | RESERVED | | | | | | | UN | ME |
| OFFSET + 2 | **DATA LENGTH** | | | | | | | | | | | | | | | |
| OFFSET + 4 | **TX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

The user should prepare the following bits before transmission.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission and the user is free to manipulate this BD or it's associated data buffer. The CP clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which is prepared for transmission by the user, is not transmitted yet or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Tx BD table.

1 = This is the last BD in the Tx BD table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to in the table. The number of Tx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = The TXB or TXE bit in the event register is set when this buffer is serviced. TXB and TXE can cause interrupts if they are enabled.

L—Last

0 = This buffer does not contain the last character of the message.

1 = This buffer contains the last character of the message.

Bit 5—Reserved

CM—Continuous Mode

This bit is only valid when the SPI is configured as a master; it should be written as a zero in slave mode.

0 Normal operation.
1 The R-bit is not cleared by the CP after this BD is closed, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this BD.

Bits 7–13—Reserved

The following status bits are written by the SPI after it finishes transmitting the associated data buffer.

UN—Underrun

The SPI encounters a transmitter underrun condition while transmitting the associated data buffer. This error condition is only valid when the SPI is configured as a slave.

ME—Multimaster Error

This buffer is closed because the SPISEL pin is asserted when the SPI operates as a master. This indicates a synchronization problem between multiple masters on the SPI bus.

Data Length

The data length is the number of octets that the CP should transmit from this BD data buffer and it is never modified by the CP. Normally, this value should be greater than zero. If the number of data bits in the character is greater than 8, then the data length should be even. For example, to transmit three characters of 8-bit data, 1 start, and 1 stop, the data length field should be initialized to 3. However, to transmit three characters of 9-bit data, the data length field should be initialized to 6 since the three 9-bit data fields occupy three half-words in memory.

Tx Data Buffer Pointer

The transmit buffer pointer, which always points to the first location of the associated data buffer, can be even or odd (unless the number of actual data bits in the character is greater than 8 bits, in which case the transmit buffer pointer must be even). The buffer can reside in internal or external memory.

**16.16.5.6 SPI EVENT REGISTER.** The SPIE is an 8-bit register used to generate interrupts and report events recognized by the SPI. On recognition of an event, the SPI sets it's corresponding bit in the SPIE register. Interrupts generated by this register can be masked in the SPI mask register. The SPIE register is a memory-mapped register that can be read at any time and a bit is cleared by writing a 1 (writing a zero does not affect a bit value). More than one bit can be cleared at a time. However, all unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset.

SPIE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | MIME | TXE | RES | BSY | TXB | RXB |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | AA6 | | | | | | | |

Bits 0–1—Reserved

MME—Multi Master Error

The SPI detects that the SPISEL pin is asserted externally while the SPI is in master mode.

TXE—Tx Error

An error occurs during transmission (underrun in SPI slave mode).

Bit 4—Reserved

BSY—Busy Condition

Received data is discarded due to a lack of buffers. This bit is set after the first character is received for which there is no receive buffer available.

TXB—Tx Buffer

A buffer is transmitted. This bit is set once the transmit data of the last character in the buffer is written to the transmit FIFO. The user must wait two character times to be sure that the data is completely sent over the transmit pin.

RXB—Rx Buffer

A buffer is received. This bit is set after the last character is written to the receive buffer and the Rx BD is closed.

**16.16.5.7 SPI MASK REGISTER.** The SPIM is an 8-bit read/write register that has the same bit formats as the SPI event register. If a bit in the SPIM register is 1, the corresponding interrupt in the SPIE register is enabled. If the bit is zero, the corresponding interrupt in the SPIE register is masked. This register is cleared at reset.

### 16.16.6 SPI Master Example

The following list is an initialization sequence for a high-speed use of the SPI as a master.

1. Configure the port B pins to enable the SPIMOSI, SPIMISO, and SPICLK pins. Write PBPAR and PBDIR bits 30, 29 and 28 with ones and then PBODR bits 30, 29 and 28 with zeros.

   **NOTE**

   In the case of multimaster operation, the SPISEL pin should also be enabled to internally connect to the SPI.

2. Configure a parallel I/O pin to operate as the SPI select pin if needed. Supposing PB15 is chosen, write PBODR bit 15 with a zero, PBDIR bit 15 with a one, and PBPAR bit 15 with a zero. Then write PBDAT bit 15 with a zero to constantly assert the select pin.
3. Write RBASE and TBASE in the SPI parameter RAM to point to the Rx BD and Tx BD in the dual-port RAM. Assuming one Rx BD at the beginning of the dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.
4. Program the CPCR to execute the INIT RX and TX PARAMS commands. Write $0051 to the CPCR.
5. Write $0001 to the SDCR to initialize the SDMA configuration register.
6. Write RFCR and TFCR with $18 for normal operation.
7. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = $0010.
8. Initialize the Rx BD and assume the Rx data buffer is at $00001000 in main memory. Write $B000 to Rx_BD_Status, $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
9. Initialize the Tx BD and assume the Tx data buffer is at $00002000 in main memory and contains five 8-bit characters. Write $B800 to Tx_BD_Status, $0005 to Tx_BD_Length, and $00002000 to Tx_BD_Pointer.
10. Write $FF to the SPIE register to clear any previous events.
11. Write $37 to the SPIM register to enable all possible SPI interrupts.
12. Write $00000020 to the CIMR to allow the SPI to generate a system interrupt. The CICR should also be initialized.
13. Write $0370 to the SPMODE register to enable normal operation (not loopback), master mode, SPI enabled, 8-bit characters, and the fastest speed possible.
14. Write PBDAT Bit 15 with zero to assert the SPI select pin.
15. Set the STR bit in the SPCOM register to start the transfer.

NOTE

After 5 bytes are transmitted, the Tx BD is closed. Additionally, the receive buffer is closed after 5 bytes are received because the L-bit of the Tx BD is set.

### 16.16.7 SPI Slave Example

The following list is an initialization sequence for use of the SPI as a slave. It is very similar to the SPI master example except that the SPISEL pin is used, rather than a general-purpose I/O pin.

1. Configure the port B pins to enable the SPIMOSI, SPIMISO, SPISEL, and SPICLK pins. Write PBPAR and PBDIR bits 31, 30, 29, and 28 with ones and then PBODR bits 31, 30, 29, and 28 with zeros.
2. Write RBASE and TBASE in the SPI parameter RAM to point to the Rx BD and Tx BD in the dual-port RAM. Assuming one Rx BD at the beginning of the dual-port RAM and one Tx BD following that Rx BD, write RBASE with $0000 and TBASE with $0008.
3. Write RFCR and TFCR with $18 for normal operation.
4. Program the CPCR to execute the INIT RX and TX PARAMS commands. Write $0051 to the CPCR.
5. Write $0001 to the SDCR to initialize the SDMA configuration register.
6. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = $0010.
7. Initialize the Rx BD and assume the Rx data buffer is at $00001000 in main memory. Write $B000 to Rx_BD_Status, $0000 to Rx_BD_Length (not required), and $00001000 to Rx_BD_Pointer.
8. Initialize the Tx BD and assume the Tx data buffer is at $00002000 in main memory and contains five 8-bit characters. Write $B800 to Tx_BD_Status, $0005 to Tx_BD_Length, and $00002000 to Tx_BD_Pointer.
9. Write $FF to the SPIE register to clear any previous events.
10. Write $37 to the SPIM register to enable all possible SPI interrupts.
11. Write $00000020 to the CIMR to allow the SPI to generate a system interrupt. The CICR should also be initialized.
12. Write $0170 to the SPMODE register to enable normal operation (not loopback), master mode, SPI enabled, and 8-bit characters. The SPI baud rate generator speed is ignored because the SPI is in slave mode.
13. Set the STR bit in the SPCOM register to enable the SPI to be ready once the master begins the transfer.

NOTE

If the master transmits 3 bytes and negates the SPISEL pin, the Rx BD is closed, but the Tx BD remains open. If the master transmits 5 or more bytes, the Tx BD is closed after the fifth byte. If the master transmits 16 bytes and negates the SPISEL pin, the Rx BD is closed with no errors and no out-of-buffers error occurs. If the master transmits more than 16 bytes, the Rx BD is closed (completely full) and the out-of-buffers error occurs after the 17th byte is received.

### 16.16.8 SPI Interrupt Handling

The following list describes what normally occurs within an SPI interrupt handler.

1. Once an interrupt occurs, the SPIE register should be read by the user to determine the sources causing the interrupts. Normally, the SPIE bits should be cleared at this time.
2. Process the Tx BD to reuse it and the Rx BD to extract the data from it. To transmit another buffer, simply set the Tx BD R-bit, Rx BD E-bit, and STR bit in the SPCOM register.
3. Clear the SPI bit in the CISR.
4. Execute the RFI instruction.

## 16.17 I$^2$C CONTROLLER

The I$^2$C controller allows the MPC821 to exchange data with a number of other I$^2$C devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCD displays.

### 16.17.1 Overview

The I$^2$C is a synchronous, multimaster bus that is used to connect several ICs on a board and it uses two wires, serial data (SDA) and serial clock (SCL) to carry information between the ICs connected to it.

The I$^2$C controller consists of transmitter and receiver sections, an independent baud rate generator, and a control unit. The transmitter and receiver sections use the same clock, which is derived from the I$^2$C controller baud rate generator in master mode and generated externally in slave mode. According to the I$^2$C spec, wait states are inserted during a data transfer if the SCL signal is held low by a slave device. As a master in the middle of a data transfer, the I$^2$C controller recognizes wait states by monitoring the SCL signal. The I$^2$C controller does not begin counting down from a specific timeout value upon SCL assertion and therefore monitoring SCL assertion times for bus timeout should be performed in the software. Refer to Figure 16-114 for the I$^2$C controller block diagram.

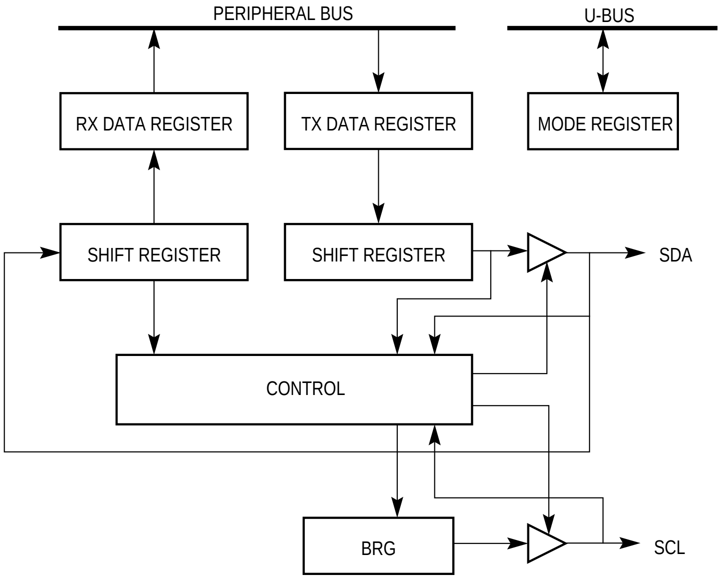


**Figure 16-114. I$^2$C Controller Block Diagram**

**NOTE**

The I$^2$C receiver and transmitter are double-buffered as shown in the block diagram. This corresponds to an effective FIFO size (latency) of 2 characters. The MPC821 I$^2$C bit 0 (MSB) is shifted out first. When the I$^2$C is not enabled in the I2MOD, it consumes minimal power.

### 16.17.2 Features

The following is a list of the I$^2$C controller's important features:

- Two-wire interface (SDA and SCL)
- Full-duplex operation
- Master or slave I$^2$C modes supported
- Multimaster environment support
- Continuous transfer mode for auto scanning of a peripheral
- Supports clock rates up to 520 KHz (assuming a 25-MHz system clock)
- Independent programmable baud rate generator
- Open-drain output pins support multimaster configuration
- Local loopback capability for testing

### 16.17.3 I$^2$C Controller Clocking and Pin Functions

The I$^2$C controller can be configured as a master for the serial channel (generates both the clock signal, initiates and terminates the transfer) or as slave (the clock signal is inputs to the I$^2$C controller). When the I$^2$C controller is a master, the controller's baud rate generator is used to generate the transmit and receive clocks. The I$^2$C baud rate generator takes it's input from the BRGCLK.

The BRGCLK is generated in the clock synthesizer of the MPC821 specifically for the I$^2$C baud rate generator and the other four baud rate generators in the CPM. Both serial data (SDA) and serial clock (SCL) are bidirectional pins connected to a positive supply voltage via an external pull-up resistor. When the bus is free both lines are high.

When the I$^2$C controller is working as a master, SCL is the clock output signal that shifts in the received data and shifts out the transmitted data to or from the SDA pin. Additionally, the transmitter arbitrates for the bus during the transmission and aborts the transmission in case it loses arbitration. When the I$^2$C controller is working as a slave, SCL is the clock input signal that shifts in the received data pin and shifts out the transmitted data to or from the SDA pin.

### 16.17.4 I$^2$C Controller Transmit/Receive Process

**16.17.4.1 I$^2$C MASTER MODE.** When the I$^2$C controller functions in master mode, the I$^2$C master initiates a transaction by transmitting a message to the peripheral (I$^2$C slave). The message specifies a read or write operation. If a read operation is specified, the direction of the transfer is changed at the first acknowledge and the slave receiver becomes a slave transmitter.

To begin the data exchange, the CPU core writes the data to be transmitted into a data buffer, configures a Tx BD with it's R-bit set, and configures one or more Rx BDs. The CPU core should then sets the STR bit in the I$^2$C command register to start transmitting data. The data starts transmitting once the SDMA channel loads the transmit FIFO with data and the I$^2$C bus is not busy.

The I$^2$C controller then generates a start condition on the SDA and SCL lines and a programmable clock pulses on the SCL pin for each data bit shifted out on the SCL pin. For each bit shifted out, the transmitter monitors the level of the SDA pin, to detect a possible collision with other I$^2$C master transmitter. If a collision is detected (the data bit transmitted was ‘1‘, but the SDA pin was ‘0’) the transmission is aborted and the channel reverts to the slave mode. A maskable interrupt is generated to the CPU, to enable the software to attempt retransmission later. After each byte, the master transmitter monitors the acknowledge indication. If the receiver fails to acknowledge a byte, the transmission is aborted and the stop condition is generated by the master. If a I$^2$C slave read operation is to be performed, the CPU should prepare transmit buffer which is *N*+1 bytes is size. The first byte should be initialized to the slave address with the read/write bit set. The following *N* byte values are ignored by the I$^2$C controller and they can be left uninitialized. *N* is the number of bytes needed to be read from the slave.

Reception begins when the start condition is detected on the SDA and SCL lines. After the first byte is shifted in, the receiver compares the received data to the slave address as programmed in the I$^2$C address register. If a match is found, the received data is acknowledged and written into a receive buffer using the next available Rx BD until a new start or stop condition is detected. On the other hand, if a mismatch is found, the receiver aborts reception and searches again for a new start condition. The receiver acknowledges each data byte as long as an overrun condition did not occur.

When multiple Tx BDs are ready for transmission, the Tx BD L-bit determines whether or not the I$^2$C controller continues to transmit without waiting for the STR bit to be set again. If the L-bit is cleared, the data from the next Tx BD begins it’s transmission following the transmission of data from the first Tx BD. If the L-bit is set, transmission stops after data from this Tx BD finishes transmitting.

**16.17.4.2 I$^2$C SLAVE MODE.** When the I$^2$C controller functions in slave mode, it receives messages from an I$^2$C master and, in turn, sends back a reply. Once a slave mode operation is selected in I$^2$C mode register, the SCL pin becomes an input from the master to the slave. SCL can be any frequency from DC to the BRGCLK/48 (520 KHz for a 25-MHz system). Before the data exchange, the CPU core writes the data to be transmitted into a data buffer, configures a Tx BD with it’s R-bit set, and configures one or more Rx BDs. The CPU core should then set the STR bit in the I$^2$C command to enable the I$^2$C controller to prepare the data for transmission and wait for a read request from the master.

On detection of a match between the first byte received following a start condition and the slave address, the read/write bit (bit 0 of the address byte) is evaluated. If a write operation is requested by the master (R/W =0), the received data is acknowledged and written into a receive buffer using the next available Rx BD until a new start or stop condition is detected. If a read operation is requested by the I$^2$C master (R/W=1), the newly received address byte is acknowledged only if the transmitter FIFO has been loaded by the SDMA channel. If the transmitter is ready, transmission starts on the next clock pulse following the acknowledge. Otherwise the transaction is aborted and a maskable Tx Error interrupt is set to notify the software to prepare data for transmission on the next attempt.

After each byte, the transmitter checks the acknowledge bit. If the master receiver fails to acknowledge a byte the transmission is aborted and a maskable interrupt is issued. Data is shifted out from the slave on the SDA pin. A maskable interrupt is issued upon complete transmission of a full buffer or after an error occurs. If an underrun condition occurred, the slave transmits ones until a stop condition is detected.

### 16.17.5 Programming Model

**16.17.5.1 I$^{2}$C MODE REGISTER.** The I$^{2}$C mode (I2MOD) register is a read/write register cleared at reset that controls both the I$^{2}$C operation mode and clock source.

I2MOD REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | REVD | GCD | FLT | PDIV | | EN |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 860 | | | | | | | |

Bits 0–1—Reserved

These bits should be cleared by the user.

REVD—Reverse Data

The REVD bit determines the receive and transmit character bit order.

0= Reverse data—LSB of character transmitted and received first.
1= Normal operation—MSB of character transmitted and received first.

GCD—General Call Disable

The GCD bit determines if the receiver will acknowledge a general call address.

0= General Call Address is enabled.
1= General Call Address is disabled.

FLT—Clock Filter

The FLT bit determines if the I$^{2}$C input clock is filtered to prevent spikes in case of a noisy environment.

0 = I2CLK is not filtered.
1 = I2CLK is filtered by a digital filter.

PDIV—Pre Divider

The PDIV bit field determines the division factor of the clock before it is fed into the BRG. The clock source for the I$^2$C is the BRGCLK that is generated by the SIU.

00= Use the BRGCLK/32 as the input to the I$^2$C baud rate generator.
01= Use the BRGCLK/16 as the input to the I$^2$C baud rate generator.
10= Use the BRGCLK/8 as the input to the I$^2$C baud rate generator.
11= Use the BRGCLK/4 as the input to the I$^2$C baud rate generator.

EN—Enable I$^2$C

The EN bit enables the I$^2$C operation. When the EN bit is cleared, the I$^2$C is in a reset state and consumes minimal power (the I$^2$C baud rate generator is not functioning and the input clock is disabled).

0 = I$^2$C is disabled.
1 = I$^2$C is enabled.

**NOTE**

Other bits of the I2MOD should not be modified by the user while the EN bit is set.

**16.17.5.2 I$^2$C ADDRESS REGISTER.** The I$^2$C address (I2ADD) register is an 8-bit, memory mapped, read/write register that holds the address for this I$^2$C port.

II2ADD REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | SAD[0:7] | | | | | | | RESERVED |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | A64 | | | | | | | |

SAD0–SAD7—Slave Address

This bit field holds the slave address for the I$^2$C port.

Bit 7—Reserved

**16.17.5.3 I$^2$C BRG REGISTER.** The I$^2$C BRG (I2BRG) register is an 8-bit, memory mapped, read/write register that sets the divide ratio of the BRG. This register is set to all ones on a hard reset.

I2BRG REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | DIV[0:7] | | | | | | | |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 868 | | | | | | | |

DIV0–DIV7—Division Ratio

These eight bits specify the divide ratio of the BRG divider in the I$^2$C clock generator. The output of the prescaler is divided by 2 * ([DIV0–DIV7] + 3) and the clock has a 50% duty cycle.

**NOTE**

The minimum value for DIV is three if the digital filter is disabled and six if it is enabled.

**16.17.5.4 I$^2$C COMMAND REGISTER.** The I$^2$C command (I2COM) register is an 8-bit read/write register that is used to start I$^2$C operation.

I2COM REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | STR | RESERVED | | | | | | M/S |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 86C | | | | | | | |

STR—Start Transmit

When the I$^2$C is configured as a master, setting the STR bit to 1 causes the I$^2$C controller to start the transmission of data from the I$^2$C transmit buffers (if they are configured as ready by the user).

When the I$^2$C is configured as a slave, setting the STR bit to 1 when the I$^2$C is idle (between transfers) causes the I$^2$C to load the transmit data register from the I$^2$C transmit buffer and start transmission when an address byte is received that matches the slave address with the R/W bit set to 1. The STR bit is always read as a zero.

Bits 1–6—Reserved.

These bits should be written with zeros by the user.

M/S—Master/Slave

The M/S bit configures the I$^2$C to work as a master or a slave.

0 = I$^2$C is a slave.
1 = I$^2$C is a master.

**16.17.5.5 I$^2$C PARAMETER RAM MEMORY MAP.** The I$^2$C controller parameter RAM area (see the table below) begins at the I$^2$C base address. This area is used for the general I$^2$C parameters. The user should notice that it is similar to the SCC general-purpose parameter RAM.

**Table 16-36. I$^2$C Parameter RAM Memory Map**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| I$^2$C Base + 00 | **RBASE** | Half-word | Rx BD Base Address |
| I$^2$C Base+ 02 | **TBASE** | Half-word | Tx BD Base Address |
| I$^2$C Base+ 04 | **RFCR** | Byte | Rx Function Code |
| I$^2$C Base+ 05 | **TFCR** | Byte | Tx Function Code |
| I$^2$C Base+ 06 | **MRBLR** | Half-word | Maximum Receive Buffer Length |
| I$^2$C Base+ 08 | RSTATE | Word | Rx Internal State |
| I$^2$C Base+ 0C |  | Word | Rx Internal Data Pointer |
| I$^2$C Base+ 10 | RBPTR | Half-word | Rx BD Pointer |
| I$^2$C Base+ 12 |  | Half-word | Rx Internal Byte Count |
| I$^2$C Base+ 14 |  | Word | Rx Temp |
| I$^2$C Base+ 18 | TSTATE | Word | Tx Internal State |
| I$^2$C Base+ 1C |  | Word | Tx Internal Data Pointer |
| I$^2$C Base+ 20 | TBPTR | Half-word | Tx BD Pointer |
| I$^2$C Base+ 22 |  | Half-word | Tx Internal Byte Count |
| I$^2$C Base+ 24 |  | Word | Tx Temp |

NOTE: Items in bold must be initialized by the user.
!$^2$C base = IMMR + 1C80.

Certain parameter RAM values need to be initialized by the user before the I$^2$C is enabled; other values are initialized by the CP. Once initialized, the parameter RAM values do not need to be accessed by the user software. They should only be modified when no I$^2$C activity is in progress.

**16.17.5.5.1 BD Table Pointer.** The RBASE and TBASE entries define the starting location in the dual-port RAM for the set of BDs with receive and transmit functions of the I$^2$C controller. This provides flexibility in how BDs for an I$^2$C are partitioned. By setting the W-bit in the last BD in each list, the user can select how many BDs to allocate for the transmit and receive side of the I$^2$C controller. However, the user must initialize these entries before enabling the I$^2$C controller. Furthermore, the user should not configure BD tables of the I$^2$C to overlap with any other serial channel BDs or erratic operation occurs.

**NOTE**

RBASE and TBASE should contain a value that is divisible by eight.

**16.17.5.5.2 I$^2$C Function Code Registers.** The FC entry contains the value that the user would like to appear on the address type pins (AT0–AT3) when the associated SDMA channel accesses memory. It also controls the byte-ordering convention to be used in the transfers.

RFCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | J2C BASE + 04 | | | | | | | |

Bits 0–2—Reserved

These bits should be set to zero by the user.

BO—Byte Ordering

This bit field should be set by the user to select the required byte ordering of the data buffer.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

TFCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | I2C BASE + 05 | | | | | | | |

Bits 0–2—Reserved

These bits should be set to zero by the user.

BO—Byte Ordering

This bit field should be set by the user to select the required byte ordering of the data buffer.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering—normal operation. It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

**16.17.5.5.3 Maximum Receive Buffer Length Register.** The I$^2$C has a MRBLR to define it's receive buffer length. MRBLR defines the maximum number of bytes that the MPC821 writes to a receive buffer on that I$^2$C before moving to the next buffer. The MPC821 writes fewer bytes to the buffer than the MRBLR value if a condition such as an error or end-of-frame occurs, but it never writes more bytes than the MRBLR value. It follows, then, that buffers supplied by the user that the MPC821 uses should always be the size of MRBLR (or greater) in length. The I$^2$C transmit buffers are not affected in any way by the value programmed into MRBLR and they can be individually chosen to have varying lengths, as needed. The number of bytes to be transmitted is chosen by programming the data length field in the Tx BD.

**NOTE**

MRBLR is not intended to be dynamically changed while an I$^2$C is operating. However, if it is modified in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back), then a dynamic change in receive buffer length can be successfully achieved. This takes place when the CP moves control to the next Rx BD in the table. Thus, a change to MRBLR does not have an immediate effect. To guarantee the exact Rx BD on which the change occurs, the user should only change MRBLR while the I$^2$C receiver is disabled. The MRBLR value should be greater than zero and even if the character length of the data is greater than eight bits.

**16.17.5.5.4 Receiver Buffer Descriptor Pointer.** The RBPTR for each I$^2$C channel points to the next BD 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 this pointer to the value programmed in the RBASE entry. Although in most applications the user should not need to write the RBPTR, it can be modified when the receiver is disabled or when the user is sure no receive buffer is currently in use.

**16.17.5.5.5 Transmitter Buffer Descriptor Pointer.** The TBPTR for each I$^2$C channel points to the next BD that the transmitter transfers data from when it is in idle state or to the current BD during frame transmission. After a reset or when the end of BD table is reached, the CP initializes this pointer to the value programmed in the TBASE entry. Although in most applications the user should not need to write the TBPTR, it can be modified when the transmitter is disabled or when the user is sure no transmit buffer is currently in use.

**16.17.5.5.6 Other General Parameters.** These parameters do not need to be accessed by the user in normal operation. They are only listed because they provide helpful information for experienced users and for debugging purposes. Additional parameters are listed in Table 16-36. The Rx and Tx internal data pointers are updated by the SDMA channels to show the next address to be accessed. The Tx internal byte count is a down-count value that is initialized with the Tx BD data length and decremented with every byte read by the SDMA channels. The Rx internal byte count is a down-count value that is initialized with the MRBLR value and decremented with every byte written by the SDMA channels.

**NOTE**

> To extract data from a partially full buffer, the CLOSE Rx BD command can be used.

The Rx internal state, Tx internal state, Rx temp, Tx temp, and reserved areas are only for RISC use.

**16.17.5.6 I$^2$C COMMANDS.** The following transmit and receive commands are issued to the CPCR.

**INIT TX PARAMETERS**

This command initializes all transmit parameters in this serial channel parameter RAM to their reset state and should only be issued when the transmitter is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the transmit and receive parameters.

**CLOSE Rx BD**

This command is used to force the I$^2$C controller to close the current Rx BD (if it is currently being used) and to use the next BD for any subsequently received data. If the I$^2$C controller is not in the process of receiving data, no action is taken by this command.

**INIT RX PARAMETERS**

This command initializes all the receive parameters in this serial channel parameter RAM to their reset state and should only be issued when the receiver is disabled. Notice that the INIT TX and RX PARAMETERS commands can also be used to reset the receive and transmit parameters.

**16.17.5.7 I$^2$C BUFFER DESCRIPTOR RING.** The data associated with the I$^2$C is stored in buffers that are referenced by buffer descriptors organized in a BD ring located in the dual-port RAM. Additional parameters are listed in Table 16-36. This ring has the same basic configuration as those used by the SCCs and SMCs.

The BD ring allows the user to define buffers for transmission or reception and each BD ring forms a circular queue. The CP confirms reception and transmission or indicates error conditions using the BDs to inform the processor that the buffers have been serviced. The actual buffers can reside in external memory or internal memory and data buffers can reside in the parameter area of an SCC if it is not enabled.

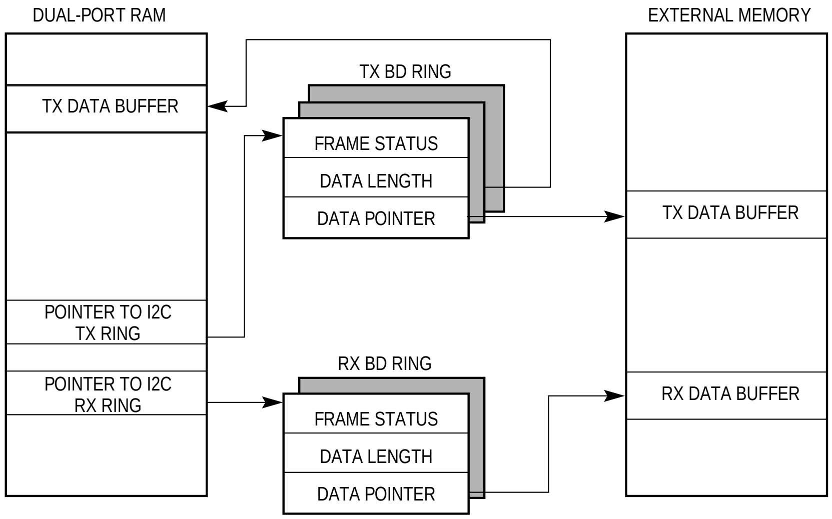


**Figure 16-115. I$^2$C Memory Structure**

**16.17.5.7.1 I$^2$C Receive Buffer Descriptor.** Using Rx BDs, the CP reports information about each buffer of received data. The CP closes the current buffer, generates a maskable interrupt, and starts receiving data in the next buffer when the current buffer is full. In addition, it closes the buffer on the following conditions:

- A stop or start condition is detected on the I$^2$C bus (end of frame is detected)
- An overrun error occurs

The first word of the Rx BD contains status and control bits that are prepared by the user before reception and set by the CP after the buffer closes. The second word contains the data length (in bytes) that is received and the third and fourth words contain a pointer that always points to the beginning of the received data buffer.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **E** | RES | **W** | **I** | L | RESERVED | | | | | | | | | OV | RES |
| **OFFSET + 2** | DATA LENGTH | | | | | | | | | | | | | | | |
| **OFFSET + 4** | **RX DATA BUFFER POINTER** | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

The CPU core should write the following bits before enabling the I²C.

E—Empty

0 = The data buffer associated with this Rx BD is filled with received data or data reception is aborted due to an error condition. The CPU core is free to examine or write to any fields of this Rx BD. The CP does not use this BD as long as the E-bit is zero.

1 = The data buffer associated with this BD is empty or reception is currently in progress. This Rx BD and it’s associated receive buffer are owned by the CP. Once the E-bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last BD in the Rx BD table.

1 = This is the last BD in the Rx BD 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 Rx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 No interrupt is generated after this buffer is filled.

1 The RXB bit in the I²C event register is set when this buffer is completely filled by the CP, indicating the need for the CPU core to process the buffer. The RXB bit can cause an interrupt if it is enabled.

The following status bits are written by the I²C after the received data is placed into the associated data buffer.

L—Last

This bit is set by the I²C controller when the buffer is closed due to detection of a stop (or start) condition on the bus or as a result of an overrun.

0 = This buffer does not contain the last character of the message.

1 = This buffer contains the last character of the message.

Bits 5–13—Reserved

OV—Overrun

A receiver overrun occurs during reception.

Bit 15—Reserved

Data Length

Data length is the number of octets that the CP writes into this BD data buffer. It is written once by the CP as the BD is closed.

**NOTE**

> The actual amount of memory allocated for this buffer should be greater than or equal to the MRBLR.

Rx Data Buffer Pointer

The receive buffer pointer, which always points to the first location of the associated data buffer, must be even. The buffer can reside in internal or external memory.

**16.17.5.7.2 I²C Transmit Buffer Descriptor.** Data to be transmitted with the I²C is presented to the CP by arranging it in buffers referenced by the Tx BD ring. The first word of the Tx BD contains status and control bits.

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFSET + 0** | **R** | RES | **W** | **I** | **L** | **S** | RESERVED | | | | | | | NAK | UN | CL |
| **OFFSET + 2** | DATA LENGTH | | | | | | | | | | | | | | | |
| **OFFSET + 4** | TX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| **OFFSET + 6** | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

The user should prepare the following bits before transmission.

R—Ready

0 = The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or it’s associated data buffer. The CP clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which is prepared for transmission by the user, is not transmitted yet or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0= This is not the last BD in the Tx BD table.

1= This is the last BD in the Tx BD table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to in the table. The number of Tx BDs in this table is programmable and determined only by the W-bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0= No interrupt is generated after this buffer is serviced.

1= The TXB or TXE bit in the event register is set when this buffer is serviced. TXB and TXE can cause interrupts if they are enabled.

L—Last

0= This buffer does not contain the last character of the message.
1= This buffer contains the last character of the message.

S—Transmit Start Condition

When this bit is set to 1, the I$^2$C controller transmits a start condition before the first byte of the buffer. If this BD is the first one in the frame, a start condition is transmitted regardless to the value of ST. This bit provides the ability to transmit a start byte or back-to-back frames.

0= A start condition is not transmitted before the first byte of the buffer, unless it is the first byte of a frame.
1= A start condition is transmitted before the first byte of the buffer.

Bits 5–12—Reserved

The following status bits are written by the I$^2$C controller after it finishes transmitting the associated data buffer.

NAK—No Acknowledge

The transmission is aborted because the last transmitted byte was not acknowledged.

UN—Underrun

The I$^2$C encounters a transmitter underrun condition while transmitting the associated data buffer.

CL—Collision

Transmission is aborted because the transmitter lost while arbitrating for the bus.

Data Length

The data length is the number of octets the CP should transmit from this BD data buffer. It is never modified by the CP. Normally, this value should be greater than zero.

Tx Data Buffer Pointer

The transmit buffer pointer, which always points to the first location of the associated data buffer, can be even or odd (unless the number of actual data bits in the character is greater than 8 bits, in which case the transmit buffer pointer must be even). The buffer can reside in internal or external memory.

**16.17.5.8 I$^2$C EVENT REGISTER.** The I$^2$C event register (I2CER) is an 8-bit register used to generate interrupts and report events recognized by the I$^2$C. On recognition of an event, the I$^2$C sets it's corresponding bit in the I2CER. Interrupts generated by this register can be masked in the I$^2$C mask register.

The I2CER is a memory-mapped register that can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset.

I2CER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | TXE | RES | BSY | TXB | RXB |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 870 | | | | | | | |

Bits 0–2—Reserved

TXE—Tx Error

An error occurs during transmission (bus arbitration lost, byte not acknowledged or underrun).

Bit 4—Reserved

BSY—Busy Condition

Received data is discarded due to a lack of buffers. This bit is set after the first character is received for which there is no receive buffer available.

TXB—Tx Buffer

A buffer is transmitted. This bit is set once the transmit data of the last character in the buffer is written to the transmit FIFO. The user must wait two character times to be sure that the data is completely sent over the transmit pin.

RXB—Rx Buffer

A buffer is received. This bit is set after the last character is written to the receive buffer and the Rx BD is closed.

**16.17.5.9 I$^2$C MASK REGISTER.** The I$^2$C mask register (I2CMR) is an 8-bit read/write register that has the same bit formats as the I2CER. If a bit in the I2CMR is 1, the corresponding interrupt in the I2CER is enabled. If the bit is zero, the corresponding interrupt in the I2CER is masked. This register is cleared at reset.

## 16.18 PARALLEL INTERFACE PORT

The parallel interface port (PIP) is a function of the CPM that allows data to be transferred to and from the MPC821 over 8 or 16 parallel data pins. The pins of the PIP are multiplexed with the 18-bit port B parallel I/O port. The PIP supports the Centronics interface and a fast parallel connection between the MPC821, but when the PIP is used, the SMC2 channel is not available.

### 16.18.1 Features

The following is a list of the PIP’s important features:

- Eighteen general-purpose I/O pins
- Three handshake modes
- Programmable handshake timing attributes
- Supports Centronics and receiver/transmitter interface
- Allows bidirectional Centronics (P1284) operation to be implemented (with some CPU intervention)
- Supports fast connection between MPC821
- Can be controlled by the CPU core or by the CPM RISC

### 16.18.2 Overview

The PIP can be operated as an 18-bit general-purpose I/O port or in one of three handshake modes:

- 8- or 16-bit strobed I/O port with two interlocked handshake signals
- 8- or 16-bit strobed I/O port with two pulsed handshake signals
- 8- or 16-bit transparent I/O port with no handshake signals

The block diagram for the PIP is illustrated in Figure 16-116.

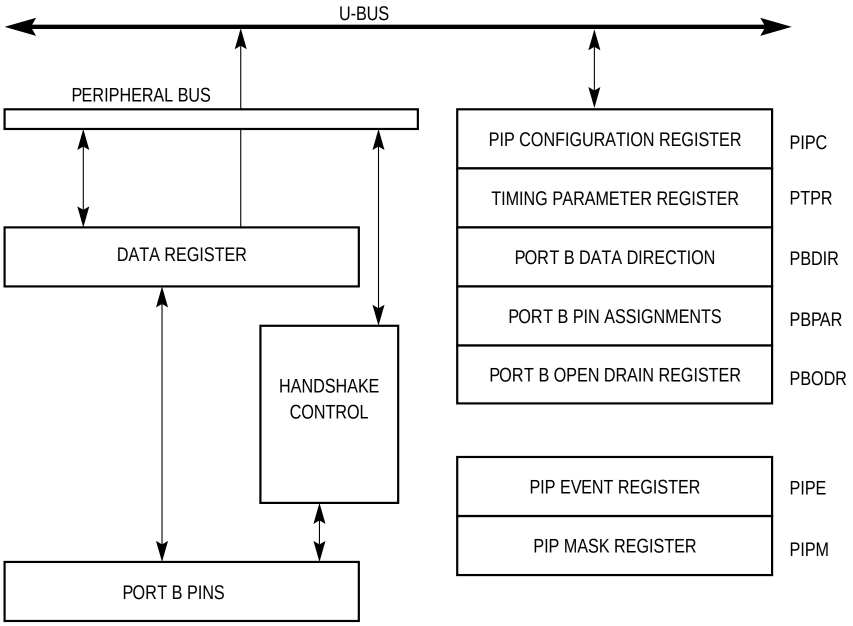


**Figure 16-116. PIP Block Diagram**

While in one of the handshake modes, the PIP is controlled either by the RISC controller or the CPU core. When it is under RISC control, data is prepared by the CPU core (or other host processor) using the same general BD structures that are used for the SCCs. Thus, the PIP can transfer or receive blocks of characters without interrupting the host processor. The data block can span several linked buffers and an entire block can be received or transmitted without CPU core intervention. When the PIP is under CPU core control (or the control of an external processor), it is controlled one byte/half-word at a time.

When the interlocked or pulsed handshake modes are used, the PIP offers programmable timing attributes, such as setup time and pulse width. The interlocked handshake mode supports level-sensitive handshake control signals and the pulsed handshake mode supports edge-sensitive handshakes like those used for the Centronics interface. The PIP mode of operation can be configured independently for two groups of port B pins—PB24–PB31 and PB14–PB23. This configuration allows an 8-bit PIP data port to be defined, rather than a full 16-bit data port.

In addition, the PIP shares several registers with the SMC2 serial channel. However, SMC2 is not available and should not be enabled if the PIP is used. If SMC2 is enabled, erratic behavior occurs.

### 16.18.3 General-Purpose I/O Pins (Port B)

In this configuration, the PIP is not used, but rather operates as general-purpose parallel I/O port B. Refer to **Section 16.18.8 Port B Registers** for more details.

### 16.18.4 Interlocked Data Transfers

In the interlocked handshake mode, the PIP can be configured as a transmitter or receiver. This configuration allows a fast connection between MPC821s and can be used for the P1284-protocol advanced byte transfer mode. This mode can either be controlled by the RISC or CPU core. Operation using the RISC requires BDs and parameter RAM initialization very similar to the other serial channels. Data is then stored in the buffers using one of the SDMA channels (from SMC2). Operation by the CPU core is performed by software-controlled reads and writes to or from the PIP data register upon interrupt request.

When configured as a transmitter, the STBO pin (PB15) is used as a strobe output (STB) handshake control signal and the STBI pin (PB14) is used as an acknowledge (ACK) input. When configured as a receiver, the PIP generates the ACK signal on the STBO pin and inputs the STB signal on the STBI pin. Bits PB15 and PB14 in the port B data direction register (PBDIR) and the port B data register (PBDAT) corresponding to STBO and STBI are invalid and ignored by the PIP in the interlocked handshake mode.

When the PIP is in this mode and configured as a transmitter, the RISC controller loads data into the output latch when it receives a request to begin transfers from the host processor Refer to Figure 16-117 for more details. Once data is loaded, after a programmable setup time, the STB signal is asserted (low). Then when ACK is sampled as low, the data is transmitted, followed by the STB being negated (high). STB remains high until new data is loaded into the output latch and ACK is negated (high). When the PIP is configured as a receiver, input data is latched when the STB signal is sampled as low. The ACK signal is then asserted. ACK is negated (high) when the data is removed from the input latch. To connect to MPC821s using this interface, connect the STBO pin of each MPC821 to the STBI pin of the other, and connect the preferred data pins (either PB24–PB16 or PB31–PB16 are connected between MPC821s).

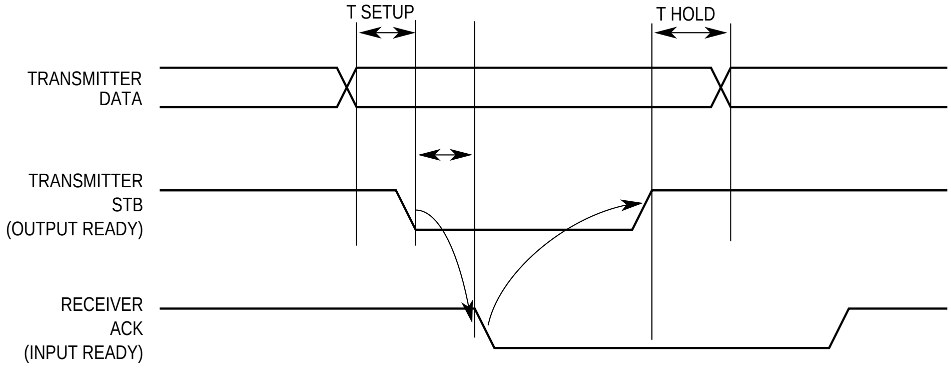


Figure 16-117. Interlock Handshake Mode

### 16.18.5 Pulsed Data Transfers

In the pulsed handshake mode, the PIP can be configured as a transmitter or receiver. This configuration allows a Centronics-compatible interface to be implemented. The pulsed handshake mode can be controlled by the RISC or CPU core. Operation using the RISC requires BDs and parameter RAM initialization very similar to the other serial channels. Data is then stored in the buffers using one of the SDMA channels (from SMC2). Operation by the CPU core is performed by software-controlled reads and writes to or from the PIP data register on an interrupt request. When configured as a transmitter, the STBO pin (PB15) is used as a strobe output (STB) handshake control signal and the STBI pin (PB14) is used as an acknowledge (ACK) input. When configured as a receiver, the PIP generates the ACK signal on the STBO pin and inputs the STB signal on the STBI pin.

Bits PB15 and PB14 in the port B data direction register (PBDIR) and the port B data register (PBDAT) corresponding to STBO and STBI are invalid and ignored by the PIP when the pulsed handshake mode is selected. When configured as a transmitter, the PIP generates the STB signal when data is ready in the PIP output latch and the previous transfer is acknowledged. Refer to Figure 16-118 for more details. The setup time and the strobe pulse width are user programmable. When configured as a receiver, the PIP uses the STB signal to latch the input data and acknowledges the transfer with the ACK signal. The timing of the ACK signal is user programmable.

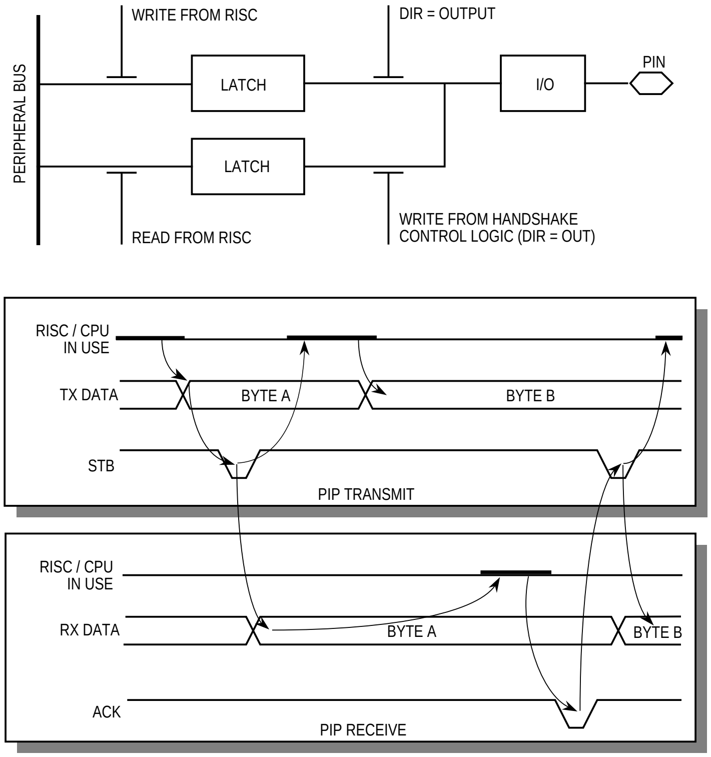


**Figure 16-118. Pulsed Handshake Full Cycle**

**16.18.5.1 BUSY SIGNAL.** In the pulsed handshake mode, the PIP receiver can generate an additional BUSY handshake signal that is useful when implementing the Centronics reception interface. Refer to Figure 16-119 for more details. The BUSY signal is an output indication of a transfer in service. It is asserted by the Centronics receiver as soon as the data is latched into the PIP data register. The timing of BUSY negation in relation to the ACK signal is user programmable. Two bits in the PIP configuration register enable the assertion and negation of the BUSY signal via the host processor software.

The BUSY signal is multiplexed onto PB31. It is not possible to use the BUSY signal with a full 16-bit PIP interface, but it can be used with the standard 8-bit PIP interface to implement Centronics functions. When in the pulsed handshake mode, the PIP transmitter can be configured to ignore the BUSY signal or suspend the assertion of the STB output until the receiver BUSY signal is negated.

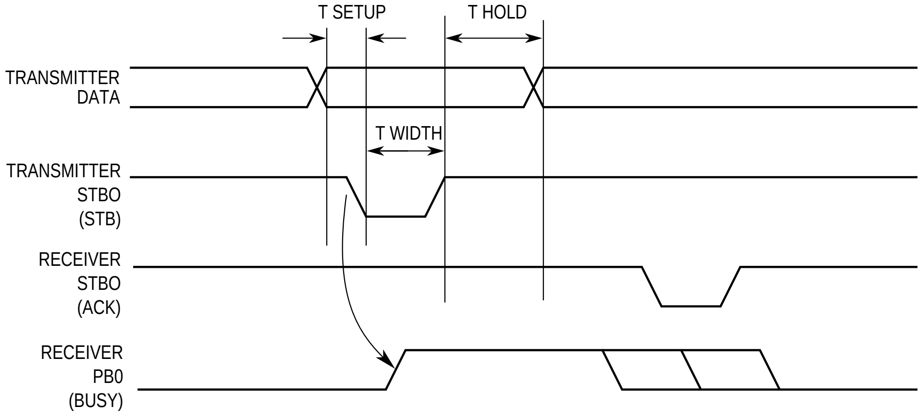


**Figure 16-119. Pulsed Handshake Busy Signal**

**16.18.5.2 PULSED HANDSHAKE TIMING.** In this mode, four Centronics receive timing options select the relative timing of the BUSY signal to the ACK signal. The pulsed handshake mode transmitter timing is illustrated in Figure 16-120. The timing parameters for the pulsed handshake mode are governed by two user-programmable timing parameters—TPAR1 and TPAR2. Each parameter defines an interval from 1 to 256 system clocks. Figure 16-121 through Figure 16-124 illustrate the definition of TPAR1 and TPAR2 in the four receive modes.

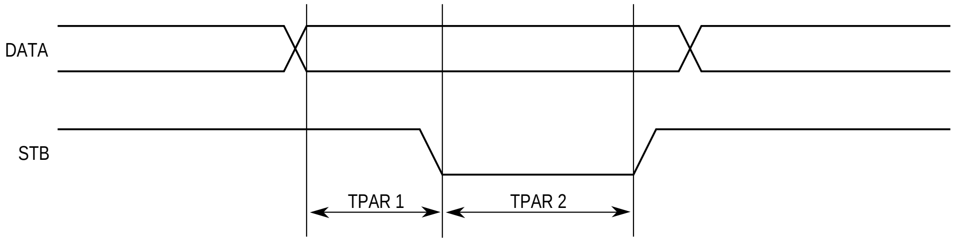


**Figure 16-120. Centronics Transmitter Timing Diagram**

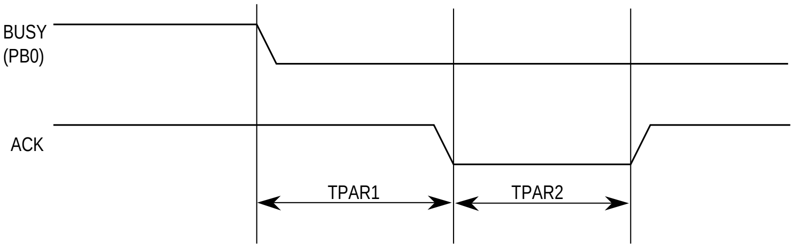


Figure 16-121. Centronics Receiver Timing Diagram Mode 0

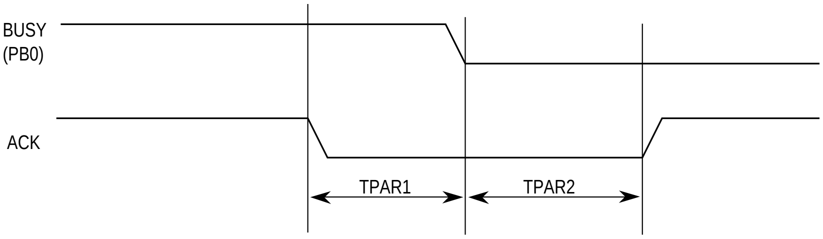


Figure 16-122. Centronics Receiver Timing Diagram Mode 1

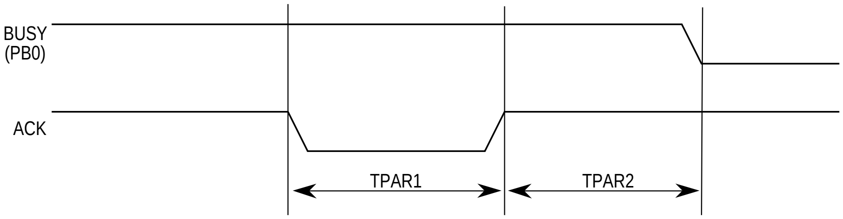


Figure 16-123. Centronics Receiver Timing Diagram Mode 2

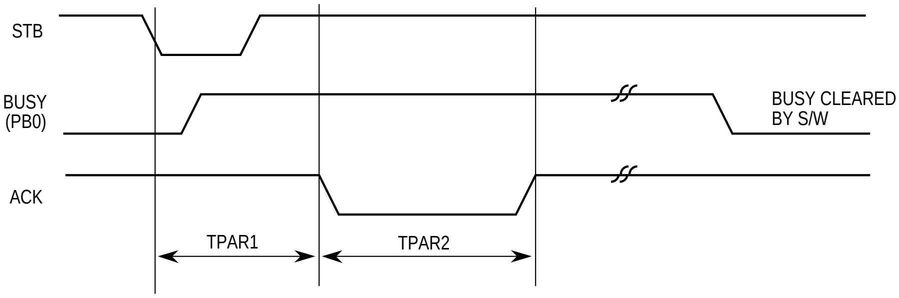


Figure 16-124. Centronics Receiver Timing Diagram Mode 3

## 16.18.6 Transparent Data Transfers

In the transparent handshake mode, the PIP can be configured as a transmitter or receiver. This configuration only has one handshake pin. The transparent mode is only controlled by the RISC, which requires BDs and parameter RAM initialization very similar to the other serial channels. Data is then stored in the buffers using one of the SDMA channels (from SMC2).

In this mode, the B14 pin falling edge generates the request to the RISC that causes the it to receive or transmit data. The direction of the pins is controlled by the port B data direction register (PBDIR). The transparent handshake mode is illustrated in Figure 16-125.

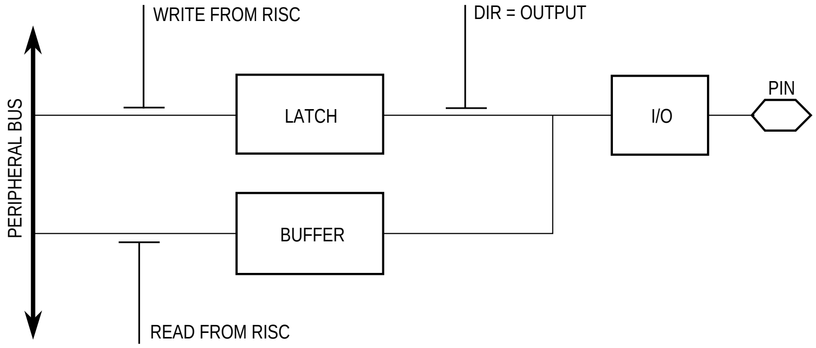


**Figure 16-125. PIP Transparent Handshake Mode**

## 16.18.7 Centronics Controller

**16.18.7.1 OVERVIEW.** Centronics is a parallel peripheral interface bus that is generally used as a communication channel between a host computer and printing equipment. The interface uses an 8-bit data bus, handshake signals that control the data exchange, and some status lines that reflect the peripheral device status. Traditionally, the direction of data transfer is from the host computer to the peripheral device. New standards, such as IEEE P1284, allow reverse channel operation (data can be transferred from the peripheral to the host).

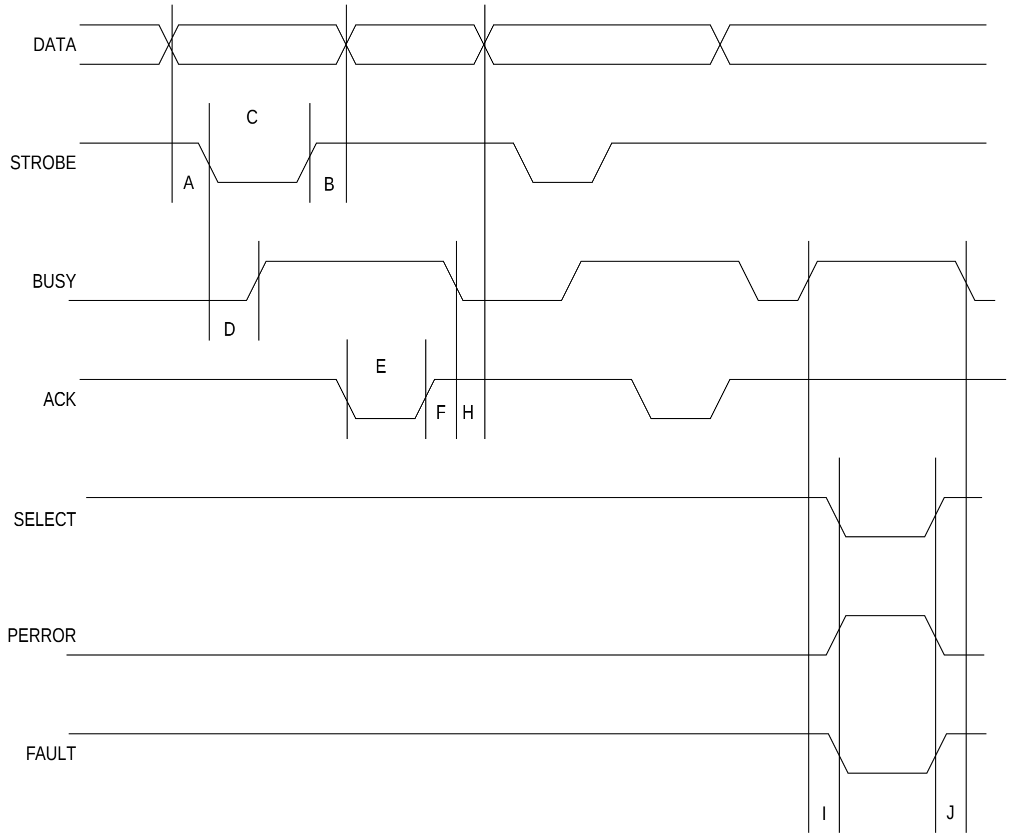


**Figure 16-126. Centronics Interface Signals**

The Centronics controller can be operated as a host port (transmitter), peripheral port (receiver), or can support bidirectional data transfer using some software support and a combination of the two modes.

### 16.18.7.2 HARDWARE CONFIGURATION.

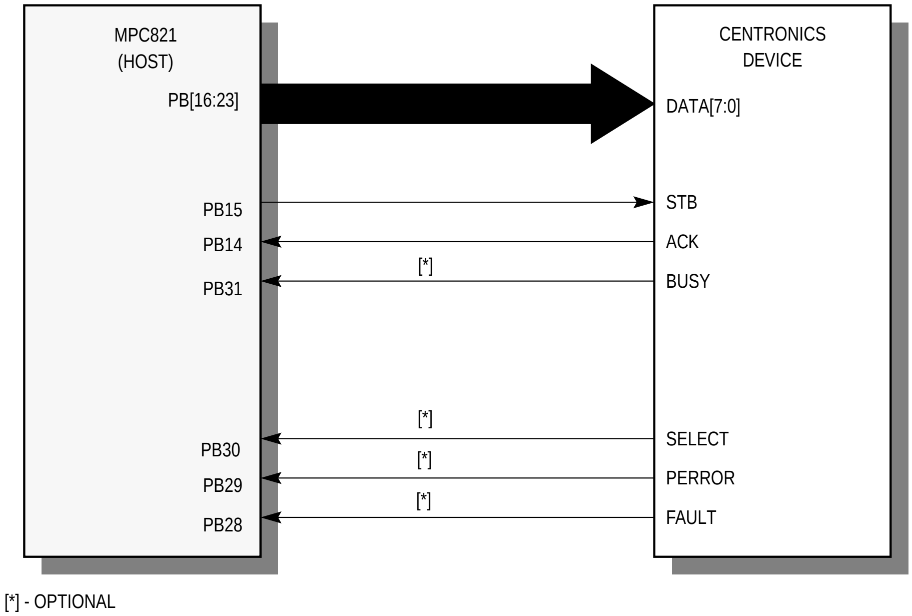


**Figure 16-127. Centronics Transmitter Configuration**

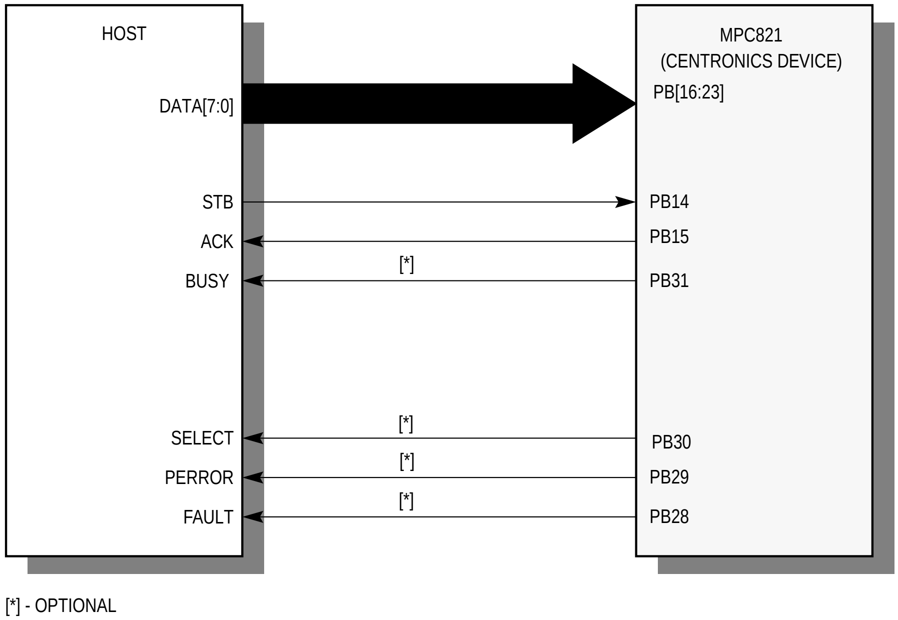


**Figure 16-128. Centronics Receiver Configuration**

**16.18.7.3 FEATURES.** The following is a list of the Centronics controller's important features:

- Superset of the Centronics standard
  - — 8- or 16-bit data transfer
  - — Supports closed loop handshake for higher data transfer rates
- Supports Centronics transmitter and receiver operating modes
- Bidirectional Centronics (P1284) support
- Message-oriented data structure flexibility
- Flexible control character comparison (receiver)
- Flexible timing modes
- Programmable timing parameters

**16.18.7.4 CENTRONICS CHANNEL TRANSMISSION.** The Centronics transmitter supports the same general data structure that is used by the SCCs for other protocols. When the STR bit in the PIP configuration register is set, the Centronics controller processes the next buffer descriptor in the Centronics transmitter BD table. If the BD is ready, the Centronics transmitter fetches the data from the memory and starts sending it to the printer. If the status mask bits are set in the SMASK register, the printer status line (Select, PError and Fault*) are checked before each transfer. In this case, the user should configure PB30, 29, and 28 pins as general-purpose inputs and connect them to Select, PError, and Fault respectively.

For each transfer, the Centronics controller outputs the data on the Centronics interface data lines and generates the strobe pulse if previous data is acknowledged and the minimum setup time is met. The strobe pulse width and the setup time parameters are programmed by the PTPR. A single data frame can span several BDs. A maskable interrupt can be generated after the processing of each BD.

**16.18.7.5 CENTRONICS TRANSMITTER MEMORY MAP.** When configured to operate in Centronics transmit mode, the MPC821 overlays the structure shown in Table 16-37 with the SMC2 parameter RAM area.

### Table 16-37. Centronics Transmitter Parameter RAM

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| PIP Base + 00 | Res | Half-word | Reserved |
| PIP Base+ 02 | **TBASE** | Half-word | Tx BD Base Address |
| PIP Base+ 04 | **CFCR** | Byte | Centronics Function Code |
| PIP Base+ 05 | **SMASK** | Byte | Status Mask |
| PIP Base+ 06 | Res | Half-word | Reserved |
| PIP Base+ 08 | Res | Word | Reserved |
| PIP Base+ 0C | Res | Word | Reserved |
| PIP Base+ 10 | Res | Half-word | Reserved |
| PIP Base+ 12 | Res | Half-word | Reserved |
| PIP Base+ 14 | Res | Word | Reserved |
| PIP Base+ 18 | TSTATE | Word | Tx Internal State |
| PIP Base+ 1C | T_PTR | Word | Tx Internal Data Pointer |
| PIP Base+ 20 | TBPTR | Half-word | Tx BD Pointer |
| PIP Base+ 22 | T_CNT | Half-word | Tx Internal Byte Count |
| PIP Base+ 24 | TTEMP | Word | Tx Temp |

NOTE: Items in bold must be initialized by the user.
PIP base = IMMR + 1F80 (SMC2).

Certain parameter RAM values need to be initialized by the user before the PIP is enabled; the others are initialized or written by the CP. Once initialized, most parameter RAM values do not need to be accessed in the user software since most of the activity is centered around the transmit buffer descriptors and not the parameter RAM.

**16.18.7.5.1 Buffer Descriptor Table Pointer.** The TBASE entry defines the starting location in the dual-port RAM for the PIP transmitter set of buffer descriptors. This provides flexibility in how BDs are partitioned. By programming the TBASE entry and setting the “wrap” bit in the last BD, the user can select how many BDs to allocate for the transmit function. The user must initialize TBASE before enabling the channel.

**NOTE**

TBASE should contain a value that is divisible by eight.

**16.18.7.5.2 Status Mask Register.** The status mask (SMASK) register controls which of the printer status lines are checked before each transfer.

SMASK REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | 0 | 0 | 0 | 0 | F | PE | S | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | PIP BASE + 05 | | | | | | | |

F—Fault

0 = The Fault status is ignored.
1 = The Fault status line is checked during transmission.

PE—Printer Error

0 = The PError status is ignored.
1 = The PError status line is checked during transmission.

S—Select Error

0 = The Select status is ignored.
1 = The Select status line is checked during transmission.

**16.18.7.5.3 Centronics Function Code Register.** The FC entry contains the value that the user would like to appear on the address type pins (AT0-3) when the associated SDMA channel accesses memory. It also controls the byte ordering convention to be used in the transfers.

CFCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | PIP BASE + 04 | | | | | | | |

Bits 0–2—Reserved

These bits should be initialized to zero.

BO—Byte Ordering

This bit field should be set by the user to select the required byte ordering of the data buffer.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

**16.18.7.5.4 Transmitter Buffer Descriptor Pointer.** The TBPTR points to the next BD that the transmitter transfers data from when it’s in IDLE state or to the current BD during frame transmission. After a reset or when the end of BD table is reached, the CP initializes this pointer to the value programmed in the TBASE entry. Although in most applications the user should not need to write TBPTR, it can be modified when the transmitter is disabled or when the user is sure that no transmit buffer is currently in use. For instance, after the STOP TRANSMIT command is issued.

**16.18.7.6 TRANSMITTER PROGRAMMING MODEL.** The host configures the PIP to operate as a Centronics controller by programming the PIP configuration (PIPC) register. Timing attributes (minimum data setup time and strobe pulse width) are set by programming the PIP timing parameters register (PTPR). The transmit errors are reported through the Tx BD.

**16.18.7.7 COMMAND SET.** The Centronics transmitter uses SMC2 transmit commands (same opcodes and channel number).

**STOP TRANSMIT**

This command disables the transmission of frames on the transmit channel. If this command is received by the Centronics controller during frame transmission, buffer transmission is aborted and the TBPTR is not advanced to the next BD. No new BD is accessed and no new buffers are transmitted for this channel. The transmitter idles until the RESTART TRANSMIT command is issued.

### RESTART TRANSMIT

This command is used to begin or resume transmission from the current TBPTR in the channel Tx BD table. When this command is received by the channel followed by the STR bit in the PIPC being set, it starts processing the current BD. This command is expected by the Centronics controller after a STOP TRANSMIT command, after the disabling of the channel in it’s mode register, or after a transmitter error occurs.

### INIT TX PARAMETERS

This command initializes all the transmit parameters in the Centronics channel parameter RAM to their reset state and should only be issued when the transmitter is disabled.

## 16.18.7.8 TRANSMISSION ERRORS.

### Buffer Descriptor Not Ready

This error occurs if the Centronics transmitter is active (STR bit in the PIP mode register is asserted by the host) and the current BD that should be processed by the Centronics controller is not ready (R bit in the BD = 0). When this condition occurs, the TXE (transmit error) interrupt is set. The channel resumes transmission after the software prepares the BD and asserts the STR bit.

### Printer Off-Line Error

This error occurs if the printer is off-line (select line is negated) and if the printer status check option is enabled. The S bit is set in the BD and the TX Error (TXE) interrupt is also set. The channel resumes transmission after the RESTART TRANSMIT command.

### Printer Fault

This error occurs if the printer has a fault condition (Fault* line asserted) and if the printer status check option is enabled. The F bit is set in the BD and TX Error (TXE) interrupt is also set. The channel resumes transmission after the RESTART TRANSMIT command.

### Paper Error

This error occurs if the printer has an error in it’s paper path (PError line asserted) and if the printer status check option is enabled. The PE bit is set in the BD and the TX Error (TXE) interrupt is set. The channel resumes transmission after the RESTART TRANSMIT command.

**16.18.7.9 CENTRONICS TRANSMITTER BUFFER DESCRIPTOR.** The CP confirms transmission (or indicates error conditions) via the buffer descriptors to inform the processor that the buffers are serviced.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **R** | RES | **W** | **I** | L | RES | **CM** | RESERVED | | | | | F | PE | S | RES |
| OFFSET + 2 | DATA LENGTH | | | | | | | | | | | | | | | |
| OFFSET + 4 | TX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

R—Ready

0= The data buffer associated with this BD is not currently ready for transmission. The user is free to manipulate this BD or it's associated data buffer. The CP clears this bit after the buffer is transmitted or after an error condition is encountered.

1= The data buffer, which is prepared for transmission by the user, is not transmitted yet or is currently being transmitted. No fields of this BD can be written by the user once this bit is set.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0= This is not the last buffer descriptor in the Tx BD table.

1= This is the last buffer descriptor in the Tx BD table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to in the table. The number of Tx BDs in this table is programmable and determined only by the wrap bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0= No interrupt is generated after this buffer is serviced.

1= The TX bit in the PIP event register is set when this buffer is serviced by the CP, which can cause an interrupt.

L—Last

0= This buffer is not the last buffer of the frame.

1= This buffer is the last buffer of the frame.

Bit 5—Reserved

CM—Continuous Mode

0= Normal Operation.

1= The R-bit is not cleared by the CP after this buffer is closed, allowing the associated data buffer to be automatically retransmitted the next time the CP accesses this BD. However, the R bit is cleared if an error occurs during transmission.

Bits 7–11—Reserved

F—Fault

0= The Fault status remains negated during transmission.
1= The Fault status is asserted during transmission.

PE—Printer Error

0= The PError status remains negated during transmission.
1= The PError status is asserted during transmission.

S—Select Error

0= The Select status remains asserted during transmission.
1= The Select status is negated during transmission.

Bit 15—Reserved

**16.18.7.10 CENTRONICS TRANSMITTER EVENT REGISTER.** When the Centronics transmitter protocol is selected, the SMC2 event register is called the Centronics transmitter event register. It is an 8-bit register which is used to generate interrupts and report events recognized by the Centronics channel. On recognition of an event, the Centronics controller sets it's corresponding bit in the Centronics event register.

This memory-mapped register can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset.

PIPE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | TXE | RES | | CHR | TX |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | A96 | | | | | | | |

Bits 0–2—Reserved

TXE—Transmit Error
An error condition is detected. This error status is reported in the buffer descriptor.

Bits 4–5—Reserved

CHR—Character Transmitted
This error acknowledges that the last character is strobed into the receiver input latch (the transmitter asserts STB) and a new character is written to the data register.

TX—Tx Buffer

A buffer is transmitted over the Centronics channel. This bit is only set after the last character of the buffer is strobed into the receiver input latch (the transmitter asserts STB).

**16.18.7.11 CENTRONICS CHANNEL RECEPTION.** The Centronics receiver supports the same general data structure that the SCCs use for other protocols. On receiving a character from the Centronics interface, the receiver checks if the current buffer descriptor in the Centronics receiver BD table is ready for use. If the BD is ready, the Centronics receiver compares the character against a user-defined control character table. If no match is found, the character is written to the BD associated buffer. If a match is found, the character is either written to the receive buffer (on which the buffer is closed and a new receive buffer taken) or rejected, depending on the R bit in the control character table. If rejected, the character is written to the RCCR in internal RAM and a maskable interrupt is generated. A maskable interrupt is generated when the BD finishes processing. A single received data frame can span several BDs.

For each transfer, the Centronics controller generates ACK and BUSY handshake signals on the Centronics interface. The ACK pulse width and the timing of BUSY with respect to the ACK signal are determined by the PTPR’s setting.

**16.18.7.12 CENTRONICS RECEIVER MEMORY MAP.** When configured to operate in Centronics receive mode, the MPC821 overlays the structure shown in Table 16-38 with the SMC2 parameter RAM area.

**Table 16-38. Centronics Receiver Parameter RAM**

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| PIP Base + 00 | **RBASE** | Half-word | Rx Buffer Descriptors Base Address |
| PIP Base+ 02 | Res | Half-word | Reserved |
| PIP Base+ 04 | **CFCR** | Byte | Centronics Function Code |
| PIP Base+ 05 | Res | Byte | Reserved |
| PIP Base+ 06 | **MRBLR** | Half-word | Maximum Receive Buffer Length |
| PIP Base+ 08 | RSTATE | Word | Rx Internal State |
| PIP Base+ 0C | R_PTR | Word | Rx Internal Data Pointer |
| PIP Base+ 10 | RBPTR | Half-word | Rx Buffer Descriptor Pointer |
| PIP Base+ 12 | R_CNT | Half-word | Rx Internal Byte Count |
| PIP Base+ 14 | RTEMP | Word | Rx Temp |
| PIP Base+ 18 | RES | Word | Reserved |
| PIP Base+ 1C | RES | Word | Reserved |
| PIP Base+ 20 | RES | Half-word | Reserved |
| PIP Base+ 22 | RES | Half-word | Reserved |
| PIP Base+ 24 | RES | Word | Reserved |

### Table 16-38. Centronics Receiver Parameter RAM (Continued)

| ADDRESS | NAME | WIDTH | DESCRIPTION |
|---|---|---|---|
| PIP Base+28 | **MAX_SL** | Half-word | Maximum Silence period |
| PIP Base+2a | SL_CNT | Half-word | Silence counter |
| PIP Base+2c | **Charcter1** | Half-word | CONTROL character 1 |
| PIP Base+2E | **Charcter2** | Half-word | CONTROL character 2 |
| PIP Base+30 | **Charcter3** | Half-word | CONTROL character 3 |
| PIP Base+32 | **Charcter4** | Half-word | CONTROL character 4 |
| PIP Base+34 | **Charcter5** | Half-word | CONTROL character 5 |
| PIP Base+36 | **Charcter6** | Half-word | CONTROL character 6 |
| PIP Base+38 | **Charcter7** | Half-word | CONTROL character 7 |
| PIP Base+3A | **Charcter8** | Half-word | CONTROL character 8 |
| PIP Base+3C | **RCCM** | Half-word | Receive Control Character Mask |
| PIP Base+3E | **RCCR** | Half-word | Receive Character Control Register |

NOTE: Items in bold must be initialized by the user.
PIP base = IMMR + 1F80 (SMC2).

Certain parameter RAM values need to be initialized by the user before the PIP is enabled; the others are initialized or written by the CP. Once initialized, most parameter RAM values do not need to be accessed in the user software since most of the activity is centered around the transmit buffer descriptors and not the parameter RAM.

**16.18.7.12.1 Buffer Descriptor Table Pointer.** The RBASE entry defines the starting location in the dual-port RAM for the PIP receiver set of buffer descriptors. This provides flexibility in how BDs are partitioned. By programming the RBASE entry and setting the “wrap” bit in the last BD, the user can select how many BDs to allocate for the receive function. However, the user must initialize RBASE before enabling the channel.

**NOTE**

RBASE should contain a value that is divisible by eight.

**16.18.7.12.2 Centronics Function Code Register.** The FC entry contains the value that the user wants to appear on the address type pins (AT0-3) when the associated SDMA channel accesses memory. It also controls the byte-ordering convention to be used in the transfers.

CFCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | BO | | AT1–AT3 | | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | PIP BASE + 04 | | | | | | | |

Bits 0–2—Reserved

BO—Byte Ordering

The user should set this bit field to select the required byte ordering of the data buffer.

00 = DEC (and Intel) convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

**16.18.7.12.3 Receiver Buffer Descriptor Pointer.** The RBPTR points to the next BD that the receiver transfers data to when it is in IDLE state or to the current BD during frame reception. After a reset or when the end of BD table is reached, the CP initializes this pointer to the value programmed in the RBASE entry. Although in most applications the user should not need to write the RBPTR, it can be modified when the receiver is disabled.

**16.18.7.13 RECEIVER PROGRAMMING MODEL.** The host configures the PIP to operate as a Centronics controller by programming the PIPC. Timing attributes (ACK pulse width and the timing between ACK and BUSY) are set by programming the PTPR. The receive errors are reported through the Rx BD.

**16.18.7.13.1 Centronics Control Characters.** The Centronics receiver has the capability to recognize special control characters that can be used when the Centronics functions in a message-oriented environment. The user can define up to eight control characters in the control characters table and each of these characters can be written to the receive buffer (on which the buffer is closed and a new receive buffer taken) or rejected. If rejected, the character is written to the RCCR in internal RAM and a maskable interrupt is generated. This method is useful for notifying the user of the arrival of control characters that are not part of the received messages. The Centronics receiver uses a table of 16-bit entries to support control character recognition and each entry consists of the control character, valid bit, and reject character bit.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PIP BASE + 2C | E | R | | | | | | | CHARACTER1 | | | | | | | |
| PIP BASE + 2E | E | R | | | | | | | CHARACTER2 | | | | | | | |
| PIP BASE + 30 | E | R | | | | | | | CHARACTER3 | | | | | | | |
| PIP BASE + 32 | | | | | | | | •<br>•<br>•<br>•<br>• | | | | | | | | |
| PIP BASE + 3A | E | R | | | | | | | CHARACTER8 | | | | | | | |
| PIP BASE + 3C | 1 | 1 | | | | | | | RCCM | | | | | | | |
| PIP BASE + 3E | | | | | | | | | RCCR | | | | | | | |

CHARACTER1–8 —Control Character Values

These fields define control characters that should be compared to the incoming character. For less than 8 bits characters, the MSBs should be zero.

RCCM—Received Control Character Mask

The value in this register is used to mask the comparison of CHARACTER1 through CHARACTER8. The lower eight bits of this register correspond to the lower eight bits of CHARACTER1–8 and are decoded as follows.

0 = Mask this bit in the comparison of the incoming character and CHARACTER1 through CHARACTER8.

1 = The address comparison on this bit proceeds normally and no masking occurs.

RCCR—Received Control Character Register

At the time of a control character match for which the reject bit is set, the Centronics writes the control character into the RCCR and generates a maskable interrupt. The core must process the interrupt and read the RCCR before a second control character arrives. Failure to do so results in the Centronics overwriting the first control character.

E—End of Table

0 = This entry is valid. The lower 8 bits are checked against the incoming character.
1 = The entry is not valid. This must be the last entry in the control characters table.

**NOTE**

In tables with 8 control characters this bit is always 0.

R—Reject Character

0 = The character is not rejected, but written into the receive buffer. The buffer is then closed and a new receive buffer is used if there is more data in the message. A maskable (I bit in the receive BD) interrupt is generated.
1 = If this character is recognized it is not written to the receive buffer. Instead, it is written to the RCCR and a maskable interrupt is generated. The current buffer is not closed when a control character is received with R set.

**NOTE**

The two most-significant bits (bits 0 and 1) of the RCCM register must be set or erratic operation can occur during the control character recognition process.

**16.18.7.13.2 Centronics Silence Period.** The Centronics controller can be programmed to close the receive data buffer after a programmable silence period. The length of the silence period is determined by the MAX_SL register value. The Centronics controller decrements the MAX_SL value every 1,024 system clocks. If it reaches zero before any data is received, the receive buffer is automatically closed. Setting MAX_SL value to zero disables this function.

### 16.18.7.14 COMMAND SET.

### INIT RX PARAMETERS

This command initializes all the receive parameters in the Centronics parameter RAM to their reset state and should only be issued when the receiver is disabled.

### CLOSE RX BD

This command is used to force the Centronics controller to close the current receive BD if it is currently being used and to use the next BD in the list for any subsequently received data. If the Centronics controller is not in the process of receiving data, no action is taken by this command.

### 16.18.7.15 RECEIVER ERRORS.

#### Buffer Descriptor Busy

This error occurs if a character is received from the Centronics interface and the current BD that should be processed by the Centronics controller is not empty (E-bit in the BD = 0). The channel resumes reception after the software prepares the BD.

**16.18.7.16 CENTRONICS RECEIVE BUFFER DESCRIPTOR.** The CP confirms transmission (or indicates error conditions) via the buffer descriptors to inform the processor that the buffers have been serviced.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET + 0 | **E** | RES | **W** | **I** | C | RES | **CM** | SL | RESERVED | | | | | | | |
| OFFSET + 2 | DATA LENGTH | | | | | | | | | | | | | | | |
| OFFSET + 4 | RX DATA BUFFER POINTER | | | | | | | | | | | | | | | |
| OFFSET + 6 | | | | | | | | | | | | | | | | |

NOTE: Items in bold must be initialized by the user.

E—Empty

0 = The data buffer associated with this BD is filled with received data or data reception is aborted due to an error condition. The core is free to examine or write to any fields of this Rx BD. The CP does not use this BD as long as the empty bit is zero.

1 = The data buffer associated with this BD is empty or reception is currently in progress. This Rx BD and it's associated receive buffer are owned by the CP. Once the E bit is set, the CPU core should not write any fields of this Rx BD.

Bit 1—Reserved

W—Wrap (Final BD in Table)

0 = This is not the last buffer descriptor in the Rx BD table.

1 = This is the last buffer descriptor in the Rx BD 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 Rx BDs in this table is programmable and determined only by the wrap bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is filled.

1 = The RX bit in the Centronics event register is set when this buffer is completely filled by the CP, indicating the need for the CPU core to process the buffer. The RX bit can cause an interrupt if it is enabled.

C—Control Character

0 = This buffer does not contain a control character.

1 = This buffer contains a control character. The last byte in the buffer is one of the user-defined control characters.

Bit 5—Reserved

CM—Continuous Mode

0 = Normal operation.

1 = The E-bit is not cleared by the CP after this buffer is closed, allowing the associated data buffer to be automatically overwritten the next time the CP accesses this BD.

SL—Silence

The buffer is closed due to the expiration of the programmable silence period timer (defined in MAX_SL).

Bits 8–15—Reserved

**16.18.7.17 PIP EVENT REGISTER.** The PIPE register is an 8-bit register used to generate interrupts and report events recognized by the PIP. It shares the same address as the SMC2 event register thus, SMC2 cannot be used simultaneously with the PIP. On recognition of an event, the PIP sets it's corresponding bit in the PIPE. Interrupts generated by this register can be masked in the PIP mask register.

The PIPE is a memory-mapped register that can be read at any time. A bit is cleared by writing a 1 (writing a zero does not affect a bit value) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the CP clears the internal interrupt request. This register is cleared at reset.

PIPE REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | CCR | BSY | CHR | BD |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | A96 | | | | | | | |

Bits 0–3—Reserved

CCR—Control Character Received

A control character is received (with reject (R) = 1) and stored in the receive control character register.

BSY—Busy Condition

A data byte/half-word is not received or transmitted by the PIP due to a lack of buffers.

CHR—Character Received/Transmitted

A data character is transmitted or received. This event can be used to generate interrupts to the CPU core if the PIP is programmed to be controlled by the host software.

BD—Rx/Tx Buffer

A complete buffer is received or transmitted on the PIP channel that closes the receive buffer when one of the following events occur:

- A user-defined control character (and reject (R) = 0 for that character in the Centronics control characters table) is received.
- Data length is terminated (the receive buffer is filled or the transmit buffer finishes transmitting).
- A programmable silence period is received.

**16.18.7.18 PIP MASK REGISTER.** The PIPM register is an 8-bit read/write register that shares the same address as the SMC2 mask register; thus, SMC2 cannot be used simultaneously with the PIP. Each bit in the PIPM register corresponds to a bit in the PIPE register. If a bit in the PIPM register is a 1, the corresponding interrupt in the PIPE register is enabled. If the bit is a zero, the corresponding interrupt in the PIPE register is masked. This register is cleared at reset.

**16.18.7.19 PIP CONFIGURATION REGISTER.** The PIPC register is a 16-bit read/write register cleared at reset that determines all PIP options.

PIPC REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | STR | RESERVED | | | SACK | CBSY | SBSY | EBSY | TMOD | | MODL | | MODH | | HSC | T/R |
| 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 | AB2 | | | | | | | | | | | | | | | |

STR —Start (valid for a transmitter only)

This bit is only valid when the T/R bit is set to 1 (transmitter). Setting this bit to 1 causes the RISC controller to poll the Tx BD. Thus, the user should prepare a Tx BD and set it’s R bit before setting STR, which is cleared after one system clock.

Bits 1–3—Reserved

SACK—Set Acknowledge

When set, this bit asserts the receiver’s ACK output (low voltage), regardless of the receiver state. SACK should be used when implementing the IEEE P1284 bidirectional Centronics protocol.

CBSY—Clear BUSY

This bit is used by host software to force the BUSY signal low for a Centronics receiver. When CBSY is set, the BUSY signal outputs at 0 (low voltage). CBSY is cleared after the PIP negates the BUSY signal.

**NOTE**

The T/R bit should be set to 0 (receiver) if CBSY is used.

SBSY—Set BUSY

This bit is used by host software to force the BUSY signal high for a Centronics receiver. When SBSY is set, the BUSY signal outputs a 1 (high voltage). SBSY is cleared after the PIP asserts the BUSY signal.

**NOTE**

The T/R bit should be set to 0 (receiver) if SBSY is used. Also, EBSY is normally set by the user before SBSY is set. If EBSY is cleared, the PIP ignores the STB signal until CBSY is set in the software.

EBSY—Enable BUSY (receiver)

This bit has a different definition depending on whether T/R is set to receiver or transmitter.

When T/R = 0 (PIP is a receiver), the definition is as follows:

0 = Disable BUSY signal generation on PB31 for the receiver.
1 = Enable the BUSY output signal on PB31. EBSY only takes effect if Bit 31 of the PBPAR is 0 to configure this pin to belong to the PIP and Bit 31of the PBDIR is 1 to make this pin an output.

When T/R = 1 (PIP is a transmitter), the definition is as follows:

0 = Ignore the BUSY signal input on PB31 for the transmitter.
1 = Assertion of STB is conditioned by BUSY is negation. STB is not asserted until the BUSY signal, input on PB31, is negated. EBSY only takes effect if Bit 31of the PBPAR is 0 to configure this pin to belong to the PIP and Bit 31of the PBDIR is 0 to make this pin an input.

**NOTE**

The programming of MODL has no effect on the BUSY pin if EBSY is set.

TMOD—Timing Mode (Centronics receiver)

These bits are only valid when T/R is set to receive and MODH is set to pulsed handshake mode. Otherwise, they are ignored.

00 = Centronics receiver timing mode 0 (BUSY is negated before ACK is asserted).
01 = Centronics receiver timing mode 1 (BUSY is negated after ACK assertion, but before ACK negation).
10 = Centronics receiver timing mode 2 (BUSY is negated after ACK negation).
11 = Centronics receiver timing mode 3 (BUSY is negated by host software).

MODL—Mode Low

These bits determine the mode of the PIP's lower 8 pins (PB24–PB31).

00 = Port B general-purpose I/O mode (under host control).
01 = Transparent handshake mode (under RISC or host control).
1x = Mode of operation is controlled by MODH.

**NOTE**

The BUSY pin (PB31) is not affected by MODL programming if EBSY is set.

MODH—Mode High

These bits determine the mode of the PIP upper 10 pins (PB14–PB23). MODH can be changed when the RISC processor is not currently receiving or transmitting data.

00 = Port B general-purpose I/O (under host control).
01 = Transparent handshake mode (under RISC or host control).
10 = Interlocked handshake mode (under RISC or host control).
11 = Pulsed handshake mode (under RISC or host control).

HSC—Host Control

0 = The PIP data transfers are controlled by the RISC in the CPM, using the PIP parameter RAM, BDs, and SDMA channels.
1 = The PIP data transfers are controlled by the CPU core (software).

T/R—Transmit/Receive Select

This bit selects transmitter or receiver operation for the PIP when it is using the interlocked, pulsed, or transparent handshake modes.

0 = Data is input to the PIP.
1 = Data is output from the PIP.

**16.18.7.20 PIP TIMING PARAMETERS REGISTER.** The PTPR is a 16-bit read/write register that is cleared at reset. It holds two timing parameters (TPAR1 and TPAR2) that are used in the pulsed handshake modes for both a PIP transmitter and receiver.

PTPR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | TPAR2 | | | | | | | | TPAR1 | | | | | | | |
| 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 | | | | | | | | | | | | | | | | |

TPAR1—Timing Parameter 1

This 8-bit value defines the number of system clocks for TPAR1 in the transmitter or receiver pulsed handshake mode. The value $00 corresponds to one MPC821 general system clock, and the value $FF corresponds to 256 MPC821 general system clocks. A general system clock defaults to 40 nanoseconds, assuming a 25-MHz MPC821 system.

TPAR2—Timing Parameter 2

This 8-bit value defines the number of system clocks for TPAR2 in the transmitter or receiver pulsed handshake mode. The value $00 corresponds to one MPC821 general system clock, and the value $FF corresponds to 256 MPC821 general system clocks. A general system clock defaults to 40 nanoseconds, assuming a 25-MHz MPC821 system.

## 16.18.8 Port B Registers

The PIP is associated with parallel I/O port B and it’s basic operation of the port is illustrated in Figure 16-129. The registers are described in the port B description.

**16.18.8.1 PORT B ASSIGNMENT REGISTERS.** The port B assignment register (PBPAR) is an 18-bit, memory-mapped, read/write register. To use port B pins as PIP pins, the corresponding PBPAR bits must be cleared and the MODH and MODL bits in the PIP configuration register can be configured as preferred.

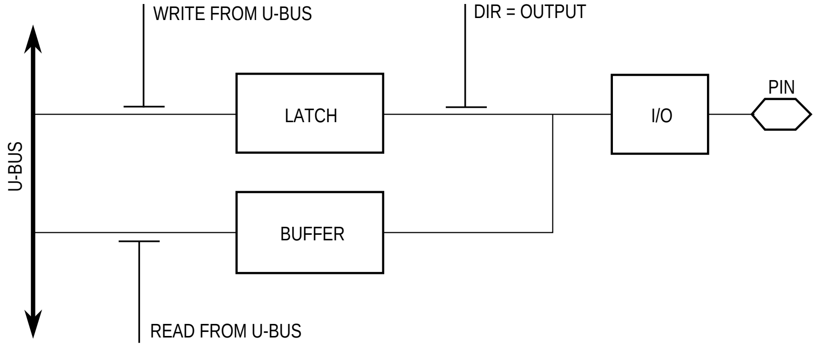


**Figure 16-129. Port B General-Purpose I/O**

**16.18.8.2 DATA DIRECTION REGISTER.** The port B data direction (PBDIR) register is an 18-bit, memory-mapped, read/write register. The description of PBDIR in **Section 16.18.8 Port B Registers** is also valid for the PIP.

**16.18.8.3 DATA REGISTER.** The port B data (PBDAT) register functions as the PIP data register when the PIP is operational. This register is used to receive/transmit PIP data when the PIP is under host software control. The description of PBDAT in **Section 16.18.8 Port B Registers** is also valid for the PIP.

**16.18.8.4 OPEN-DRAIN REGISTER.** A description of the port B open-drain register (PBODR) in **Section 16.18.8 Port B Registers** is also valid for the PIP.

## 16.19 PARALLEL I/O PORTS

### 16.19.1 Overview

The CPM supports four general-purpose I/O ports—A, B, C, and D. Each pin in the I/O ports can be configured as a general-purpose I/O pin or as a dedicated peripheral interface pin. Port A is shared with the SCC RXD and TXD pins, the bank of clocks pins, and some TDM pins. Port B is shared with the PIP and other functions such as the IDMA, SMC, SPI, and I$^2$C pins. Port C is shared with the RTS, $\overline{\text{CTS}}$, and CD pins of the SCCs as well as some TDM pins. Port C is unique in that it's pins can generate interrupts to the CPM interrupt controller.

Each pin can be configured as an input or output and has a latch for data output, read or written at any time, and configured as general-purpose I/O or as a dedicated peripheral pin. Ports A and B have pins that can be configured as open-drain (the pin can be configured in a wired-OR configuration on the board). The pin drives a zero voltage, but three-states when driving a high voltage.

**NOTE**

The port pins do not have internal pull-up resistors. Due to the significant flexibility of the MPC821 CPM, many dedicated peripheral functions are multiplexed onto ports A, B, and C. The functions are grouped in such a way as to maximize the usefulness of the pins in the greatest number of MPC821 applications. The reader may not obtain a full understanding of the pin assignment capability described in this section until attaining an understanding of the CPM peripherals themselves.

### 16.19.2 Features

The following is a list of the parallel I/O port's important features:

- Port A is 16 bits
- Port B is 18 bits
- Port C is 12 bits
- Port D is 13 bits
- All ports are bidirectional
- All ports have alternate on-chip peripheral functions
- All ports are three-stated at system reset
- All pin values can be read while the pin is connected to an on-chip peripheral
- Ports A and B offer open-drain capability
- Port C offers 12 interrupt input pins

### 16.19.3 Port A Pin Functions

Each of the 16 port A pins is independently configured as a general-purpose I/O pin if the corresponding port A pin assignment register (PAPAR) bit is cleared. Each pin is configured as a dedicated on-chip peripheral pin if the corresponding PAPAR bit is set.When the port A pin is configured as a general-purpose I/O pin, the signal direction for that pin is determined by the corresponding control bit in the port A data direction register (PADIR). The port A I/O pin is configured as an input if the corresponding PADIR bit is cleared; it is configured as an output if the corresponding PADIR bit is set. All PAPAR and PADIR bits are cleared on total system reset, configuring all port A pins as general-purpose input pins. Refer to Table 16-39 for the default description of all port A pin options (the pins in bold can have open-drain capability).

### Table 16-39. Port A Pin Assignment

| SIGNAL | PIN FUNCTION | | | |
|---|---|---|---|---|
| | PAPAR = 0 | PAPAR = 1 | | INPUT TO ON-CHIP PERIPHERALS |
| | | PADIR = 0 | PADIR = 1 | |
| PA15 | PORT A15 | RXD1 | _ | GND |
| PA14 | PORT A14 | **TXD1** | _ | — |
| PA13 | PORT A13 | RXD2 | — | GND |
| PA12 | PORT A12 | **TXD2** | — | — |
| PA11 | PORT A11 | — | **L1TXDB** | Undefined |
| PA10 | PORT A10 | — | L1RXDB | GND |
| PA9 | PORT A9 | — | **L1TXDA** | Undefined |
| PA8 | PORT A8 | — | L1RXDA | L1RXDA = GND |
| PA7 | PORT A7 | CLK1/TIN1/L1RCLKA | BRGO1 | CLK1/TIN1/L1RCLKA = BRGO1 |
| PA6 | PORT A6 | CLK2 | TOUT1 | CLK2 = GND |
| PA5 | PORT A5 | CLK3/TIN2/L1TCLKA | BRGO2 | CLK3/TIN2/L1TCLKA = BRGO2 |
| PA4 | PORT A4 | CLK4 | TOUT2 | CLK4 = CLK8 |
| PA3 | PORT A3 | CLK5/TIN3 | BRGO3 | CLK5/TIN3 = BRGO3 |
| PA2 | PORT A2 | CLK6/L1RCLKB | TOUT3 | CLK6/L1RCLKB = GND |
| PA1 | PORT A1 | CLK7/TIN4 | BRGO4 | CLK7/TIN4 = BRGO4 |
| PA0 | PORT A0 | CLK8/L1TCLKB | TOUT4 | CLK8/L1TCLKB = GND |

NOTE: 1. Items in bold must be initialized by the user.

If a port A pin is selected as a general-purpose I/O pin, it can be accessed through the port A data register (PADAT). Data written to the PADAT is stored in an output latch. If a port A pin is configured as an output, the output latch data is gated onto the port pin. In this case, when PADAT is read, the port pin itself is read. If a port A pin is configured as an input, data written to PADAT is still stored in the output latch, but is prevented from reaching the port pin. In this case, when PADAT is read, the state of the port pin is read. If an input to a peripheral is not supplied from a pin, then a default value is supplied to the on-chip peripheral as listed in Table 16-39.

## 16.19.4 Port A Registers

Port A has four memory-mapped, read/write, 16-bit control registers.

**16.19.4.1 PORT A OPEN-DRAIN REGISTER.** The port A open-drain register (PAODR) indicates a normal or wired-OR configuration of the port pins. Four of the PAODR bits can be open-drain to correspond to those pins that have serial channel output capability. The other bits are always zero. PAODR is cleared at system reset.

PAODR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | OD9 | 0 | OD11 | OD12 | 0 | OD14 | 0 |
| 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 | 954 | | | | | | | | | | | | | | | |

For each ODx bit, the definition is as follows:

0 = The I/O pin is actively driven as an output.
1 = The I/O pin is an open-drain driver. As an output, the pin is actively driven low, otherwise it is three-stated.

**16.19.4.2 PORT A DATA REGISTER.** A read of the port A data (PADAT) register returns the data at the pin, independent of whether the pin is defined as an input or output. This allows detection of output conflicts at the pin by comparing the written data with the data on the pin. A write to the PADIR is latched and if that bit in the PADIR is configured as an output, the value latched for that bit is driven onto it's respective pin. PADAT can be read or written at any time, is not initialized, and is undefined at reset.

PADAT REGISTER

| BITS | 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 | 956 | | | | | | | | | | | | | | | |

**16.19.4.3 PORT A DATA DIRECTION REGISTER.** The port A data direction (PADIR) register is cleared at system reset.

PADIR

| BITS | 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 | 950 | | | | | | | | | | | | | | | |

For each DRx bit, the definition is as follows:

0 = The corresponding pin is an input.
1 = The corresponding pin is an output.

**16.19.4.4 PORT A PIN ASSIGNMENT REGISTER.** The port A pin assignment register (PAPAR) is cleared at system reset.

PAPAR

| BITS | 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 | 952 | | | | | | | | | | | | | | | |

For each DDx bit, the definition is as follows:

0 = General-purpose I/O. The peripheral functions of the pin are not used.
1 = Dedicated peripheral function. The pin is used by the internal module. The on-chip peripheral function to which it is dedicated can be determined by other bits such as those is the PADIR.

## 16.19.5 Port A Examples

The port A pins can be configured in several ways. Figure 16-130 and Figure 16-131 illustrate the block diagrams of the PA15 and PA14 pins. PA15 can be configured as a general-purpose I/O pin, but not an open-drain pin. It can also be the RXD1 pin for SCC1 in the NMSI mode. If PA15 is configured as a general-purpose I/O pin, then the RXD1 input is internally grounded. If SCC1 is connected to a TDM or is not used, then PA15 can be used as general-purpose I/O.

PA14 can be configured as a general-purpose I/O pin, either open-drain or not. It can also be the TXD1 pin for SCC1 in the NMSI mode. If TXD1 is configured as an output on PA14 and the OD14 bit is set in the PAODR, then TXD1 is output from SCC1 as an open-drain output. If PA14 is configured as a general-purpose I/O pin, then the TXD1 output is not externally connected. If SCC1 is connected to a TDM or is not used, then PA14 can be used as a general-purpose I/O.

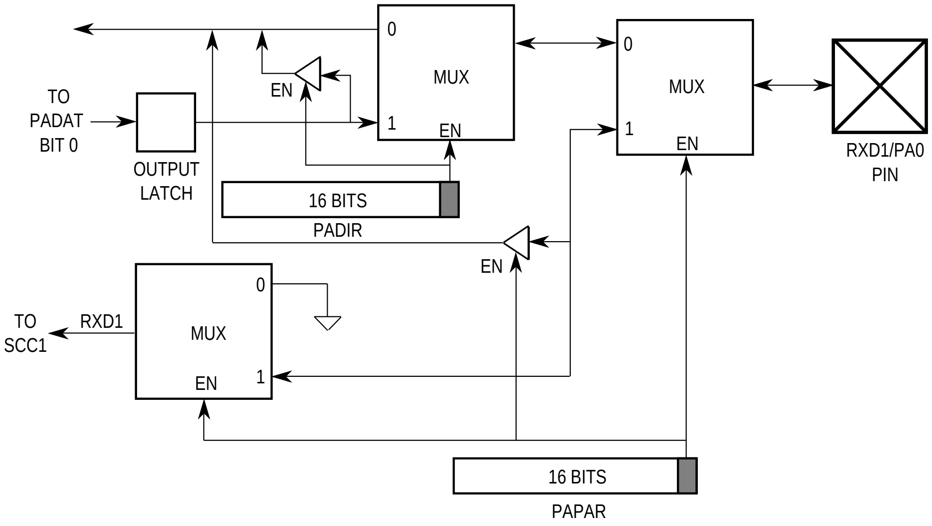


**Figure 16-130. Parallel Block Diagram For PA15**

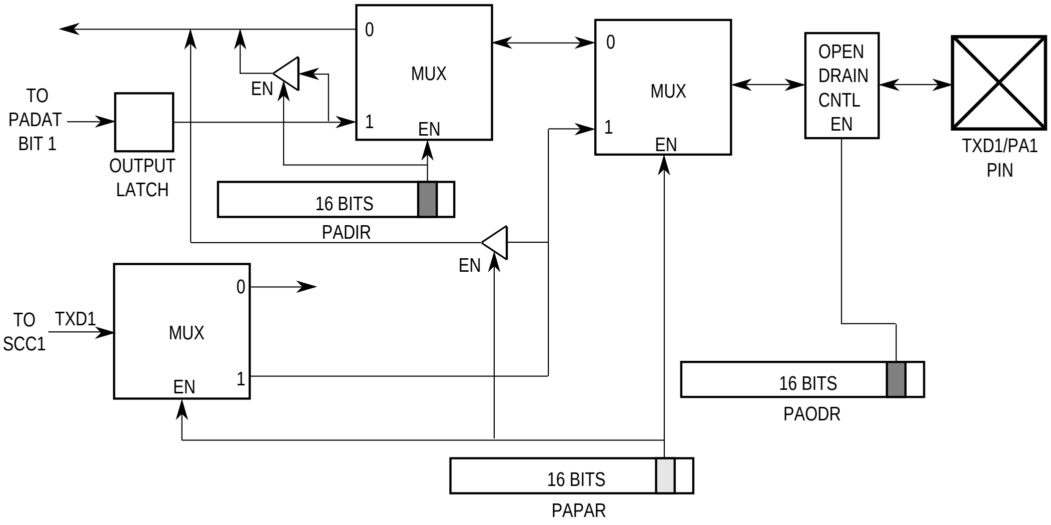


**Figure 16-131. Parallel Block Diagram For PA14**

PA11 can be configured as a general-purpose I/O and an open-drain pin. It can also be the L1TXDB pin for TDMb if the PADIR bit is a 1. PA7 can be configured as a general-purpose I/O pin, but not an open-drain pin. If the corresponding PADIR bit is a zero, it can also be the CLK1 pin (part of the bank of clocks in the SI), the TIN1 pin (input to timer 1), or the L1RCLKA pin (receive clock to TDMa) or all three at once.

There is no selection between these three inputs in port A, because the connections are made separately in the SI and the timer mode registers. If the PADIR bit is a 1, this pin can also be the BRGO1 pin (output from BRG1). If the PA7 pin is a general-purpose I/O pin, then the input to the on-chip peripheral (CLK1, TIN1, or L1RCLKa) is internally connected to BRG01. Refer to **Section 16.12 Serial Interface with Time-Slot Assigner** for more details on the use of the CLK1 and L1RCLKA pins.

PA4 can be configured as a general-purpose I/O pin, but not an open-drain pin. If the PADIR bit is a zero, PA4 can also be the CLK4 pin (part of the bank of clocks). If the PADIR bit is a 1, PA4 can also be the $\overline{\text{TOUT2}}$ pin (output from timer 2). If the PA4 pin is a general-purpose I/O pin, then the input to the on-chip CLK4 function is the value supplied on the CLK8 pin. This interesting option is useful because not all CLK pins can be routed to all serial channels in all situations. The ability to send a clock from CLK8 to CLK4 increases the flexibility of this assignment process. Refer to **Section 16.12 Serial Interface with Time-Slot Assigner** for more details.

### 16.19.6 Port B Pin Functions

All port B pins except PB14 and PB15 can be open-drain. Port B pins are independently configured as a general-purpose I/O pins if the corresponding bit in the PBPAR is cleared. They are configured as dedicated on-chip peripheral pins if the corresponding PBPAR bit is set. Refer to Table 16-40 for a description of all port B pin options. When acting as a general-purpose I/O pin, the signal direction for that pin is determined by the corresponding control bit in the PBDIR. The port I/O pin is configured as an input if the corresponding PBDIR bit is cleared; it is configured as an output if the corresponding PBDIR bit is set. All PBPAR bits and PBDIR bits are cleared on total system reset, configuring all port B pins as general-purpose input pins.

If a port B pin is selected as a general-purpose I/O pin, it can be accessed through the PBDAT. Data written to the PBDAT is stored in an output latch. If a port B pin is configured as an output, the output latch data is gated onto the port pin. In this case, when PBDAT is read, the port pin itself is read. If a port B pin is configured as an input, data written to PBDAT is still stored in the output latch, but is prevented from reaching the port pin. In this case, when PBDAT is read, the state of the port pin is read.

All of the port B pins have more than one option. These options include on-chip peripheral functions relating to the Ethernet CAM interface, SPI, I$^2$C, SMC1, SMC2, TDMa, and TDMb. Port B is also multiplexed with the PIP, which is a CPM parallel port that can implement fast parallel interfaces, such as Centronics. For a functional description of the dedicated pin functions of the PIP, refer to **Section 16.18 Parallel Interface Port**.

### NOTE

> If the user does not use the PIP, the description in this section is sufficient to describe the features of port B and the PIP description does not need to be studied. The PIP STRBI and STRBO pins are not listed in Table 16-40. Refer to **Section 16.18 Parallel Interface Port** for instructions on how to enable them.

PB28–PB26 and PB23 have an unusual property in that their on-chip peripheral functions (BRGO4, BRGO3, BRGO2, and BRGO1) are repeated in port A. This gives an alternate way to output the BRGO pins if other functions are used on port A. PB19–PB16 have an unusual property in that their on-chip peripheral functions (RTSx or L1ST1) are repeated in port C. This gives an alternate location to output these pins if other functions on port C are used.

**Table 16-40. Port B Pin Assignment**

| SIGNAL | PIN FUNCTION | | | |
|---|---|---|---|---|
| | PBPAR = 0 | PBPAR = 1 | | INPUT TO ON-CHIP PERIPHERALS |
| | | PBDIR = 0 | PBDIR = 1 | |
| PB31 | PORT B31 | **REJECT1** | SPISEL | $V_{DD}$ |
| PB30 | PORT B30 | RSTRT2 | SPICLK | SPICLK = GND |
| PB29 | PORT B29 | REJECT2 | SPIMOSI | $V_{DD}$ |
| PB28 | PORT B28 | BRGO4 | SPIMISO | SPIMISO = SPIMOSI |
| PB27 | PORT B27 | BRGO1 | I2CSDA | $V_{DD}$ |
| PB26 | PORT B26 | BRGO2 | I2CSCL | I2CSCL = GND |
| PB25 | PORT B25 | SMTXD1 | — | — |
| PB24 | PORT B24 | SMRXD1 | — | — |
| PB23 | PORT B23 | SMSYN1 | SDACK1 | SMSYN1 = GND |
| PB22 | PORT B22 | SMSYN2 | SDACK2 | SMSYN2 = GND |
| PB21 | PORT B21 | SMTXD2 | L1CLKOB | — |
| PB20 | PORT B20 | SMRXD2 | L1CLKOA | SMRXD2 = GND |
| PB19 | PORT B19 | L1ST1 | RTS1 | — |
| PB18 | PORT B18 | L1ST2 | RTS2 | — |

**Table 16-40. Port B Pin Assignment**

| SIGNAL | PIN FUNCTION | | | |
|---|---|---|---|---|
| | PBPAR = 0 | PBPAR = 1 | | INPUT TO ON-CHIP PERIPHERALS |
| | | PBDIR = 0 | PBDIR = 1 | |
| PB17 | PORT B17 | L1ST3 | L1RQB | — |
| PB16 | PORT B16 | L1ST4 | L1RQA | — |
| PB15 | PORT B15 | — | BRGO3 | — |
| PB14 | PORT B14 | — | $\overline{\text{RSTRT1}}$ | — |

**NOTE**

The user can freely configure any of the previous functions to be output onto two pins at once, although there is typically no advantage in doing this (except in the case of a large fanout, where it is advantageous to share the load between two pins).

## 16.19.7 Port B Registers

Port B has four memory-mapped, read-write, 16-bit control registers.

**16.19.7.1 PORT B OPEN-DRAIN REGISTER.** The port B open drain register (PBODR) is a 16-bit register that indicates a normal or wired-OR configuration of the port pins. Bits 14 and 15 of PBODR do not exist. PBODR is cleared at system reset.

PBODR

| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | OD16 | OD17 | OD18 | OD19 | OD20 | OD21 | OD22 | OD23 | OD24 | OD25 | OD26 | OD27 | OD28 | OD29 | OD30 | OD31 |
| RESET | | | | | | | | | | | | | | | | |
| R/W | | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | | |

For each ODx bit, the definition is as follows:

0 = The I/O pin is actively driven as an output.
1 = The I/O pin is an open-drain driver. As an output, the pin is actively driven low. Otherwise, it is three-stated.

**NOTE**

SMTXD1 cannot be set as an open-drain driver.

**16.19.7.2 PORT B DATA REGISTER.** A read of the port B data register (PBDAT) returns the data at the pin, independent of whether the pin is defined as an input or an output. This allows detection of output conflicts at the pin by comparing the written data with the data on the pin. A write to the PBDAT is latched and if that bit in the PBDIR is configured as an output, the value latched for that bit is driven onto it's respective pin. PBDAT can be read or written at any time, is not initialized, and is undefined at reset.

PBDAT REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | | | | | | | | D14 | D15 |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | AC4 | | | | | | | | | | | | | | | |
| 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 |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | AC6 | | | | | | | | | | | | | | | |

**16.19.7.3 PORT B DATA DIRECTION REGISTER.** The port B data direction register (PBDIR) is cleared at system reset.

PBDIR REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | | | | | | | | 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 | AB8 | | | | | | | | | | | | | | | |
| 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 | ABA | | | | | | | | | | | | | | | |

For each DRx bit, the definition is as follows:

0 = The corresponding pin is an input.
1 = The corresponding pin is an output.

**16.19.7.4 PORT B PIN ASSIGNMENT REGISTER.** The port B pin assignment register (PBPAR) is cleared at system reset.

PBPAR REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | | | | | | | | | | | 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 | ABC | | | | | | | | | | | | | | | |

| 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 | 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 | ABE | | | | | | | | | | | | | | | |

For each DDx bit, the definition is as follows:

0 = General-purpose I/O. The peripheral functions of the pin are not used.
1 = Dedicated peripheral function. The pin is used by the internal module. The on-chip peripheral function to which it is dedicated can be determined by other bits such as those in the PBDIR.

## 16.19.8 Port B Example

PB31 can be configured as a general-purpose I/O pin or an open-drain pin. It can also be the receiver reject pin for the SCC1 Ethernet CAM interface (REJECT1) or the SPI select input ($\overline{\text{SPISEL}}$). If the PB31 pin is not configured to connect to the REJECT or $\overline{\text{SPISEL}}$ signal, then the SCC and/or SPI receives $V_{DD}$ on that signal.

**NOTE**

In the description of the PIP, the PB31 pin, as well as other port B pins, can also be used as PIP functions. However, the PIP does not affect the operation of port B unless it is enabled. Therefore, the PIP description does not need to be studied by users of port B unless the PIP is used in the application.

### 16.19.9 Port C Pin Functions

Port C consists of 12 general-purpose I/O pins with interrupt capability on each pin. Refer to Table 16-41 for a description of all port C pin options.

**Table 16-41. Port C Pin Assignment**

| SIGNALS | PCPAR = 0 | | PCPAR = 1 | | INPUT TO ON-CHIP PERIPHERALS |
|---|---|---|---|---|---|
| | PCDIR = 1 OR PCSO = 0 | PCDIR = 0 AND PCSO = 1 | PCDIR = 0 | PCDIR = 1 | |
| PC15 | Port C15 | DREQ0 | RTS1 | L1ST1 | EXT0 = $V_{DD}$ |
| PC14 | Port C14 | DREQ1 | RTS2 | L1ST2 | EXT1 = $V_{DD}$ |
| PC13 | Port C13 | — | L1RQB | L1ST3 | — |
| PC12 | Port C12 | — | L1RQA | L1ST4 | — |
| PC11 | Port C11 | $\overline{\text{CTS1}}$ | — | | GND |
| PC10 | Port C10 | CD1 | TGATE1 | | GND |
| PC9 | Port C9 | $\overline{\text{CTS2}}$ | — | | GND |
| PC8 | Port C8 | CD2 | TGATE2 | | GND |
| PC7 | Port C7 | — | L1TSYNCB | SDACK2 | L1TSYNCB = PORT D |
| PC6 | Port C6 | — | L1RSYNCB | | L1RSYNCB=Port D |
| PC5 | Port C5 | — | L1TSYNCA | SDACK1 | L1TSYNCA = PORT D |
| PC4 | Port C4 | — | L1RSYNCA | | CD4=GNDPORT<br>DL1RSYNCA=Port D |

All PCDIR and PCPAR bits are cleared on total system reset, configuring all port pins as general-purpose input pins. Notice that the global CIMR is also cleared on total system reset so, if any PCIO pin is left floating, it does not cause a false interrupt.

If a port C pin is selected as a general-purpose I/O pin, it can be accessed through the PCDAT register where written data is stored in an output latch. If a port C pin is configured as an output, the output latch data is gated onto the port pin. In this case, when the PCDAT register is read, the port pin itself is read. If a port C pin is configured as an input, data written to PCDAT register is still stored in the output latch, but is prevented from reaching the port pin. In this case, when PCDAT register is read, the state of the port pin is read.

To configure a port C pin as a general-purpose output pin, these steps should be followed. Notice that when the pin is configured as an output, port C interrupts are not possible.

1. Write the corresponding PCPAR bit with a zero.
2. Write the corresponding PCDIR bit with a 1.
3. Write the corresponding PCSO bit with a zero (for clarity).
4. The corresponding PCINT bit is a don't care.
5. Write the pin value using the PCDAT register.

To configure a port C pin as a general-purpose input pin that does not generate an interrupt, these steps should be followed:

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.
5. Write the corresponding CIMR bit with a zero to prevent interrupts from being generated to the CPU core.
6. Read the pin value using the PCDAT register.

When a port C pin is configured as a general-purpose I/O input, a change according to the port C interrupt register (PCINT) causes an interrupt request signal to be sent to the CPM interrupt controller. Each port C line can be programmed to assert an interrupt request upon a high-to-low change or any change. Each port C line asserts a unique interrupt request to the CPM interrupt pending register and has a different internal interrupt priority level within the CPM interrupt controller. Refer to **Section 16.20 CPM Interrupt Controller** for more details. Each request can be masked independently in the CPM interrupt mask register. To configure a port C pin as a general-purpose input pin that generates an interrupt, these steps should be followed:

1. Write the corresponding PCPAR bit with a zero.
2. Write the corresponding PCDIR bit with a zero.
3. Write the corresponding PCSO bit with a zero.
4. Set the PCINT bit to determine which edges cause interrupts.
5. Write the corresponding CIMR bit with a 1 to allow interrupts to be generated to the CPU core.
6. Read the pin value using the PCDAT register.

The port C lines associated with the CDx and $\overline{\text{CTSx}}$ pins have a mode of operation where the pin can be internally connected to the SCC but can also generate interrupts. Port C still detects changes on the $\overline{\text{CTS}}$ and CD pins and asserts the corresponding interrupt request, but the SCC simultaneously uses the $\overline{\text{CTS}}$ and/or CD pin to automatically control operation. This allows the user to fully implement protocols V.24, X.21, and X.21 bis (with the assistance of other general-purpose I/O lines). To configure a port C pin as a $\overline{\text{CTS}}$ or CD pin that connects to the SCC and generates interrupts, these steps should be followed:

1. Write the corresponding PCPAR bit with a zero.
2. Write the corresponding PCDIR bit with a zero.
3. Write the corresponding PCSO bit with a 1.
4. Set the PCINT bit to determine which edges cause interrupts.
5. Write the corresponding CIMR bit with a 1 to allow interrupts to be generated to the CPU core.
6. The pin value can be read at any time using PCDAT.

**NOTE**

After connecting the $\overline{\text{CTS}}$ or CD pins to the SCC, the user must also choose the “normal operation” mode in DIAG bits of the GSMR to enable and disable SCC transmission and reception with these pins.

The DREQ0 and DREQ1 lines in port C can assert an external request to the RISC controller instead of asserting an interrupt to the CPU core. Each line can be programmed to assert an interrupt request upon a high-to-low change or any change as configured in PCINT.

**NOTE**

Do not program the DREQ0 and DREQ1 pins to assert external requests to the IDMA, unless the IDMA is used. Otherwise, errant and erratic operation occurs.

### 16.19.10 Port C Registers

The user interfaces with port C via five registers. The port C interrupt control register (PCINT) indicates how changes on the pin cause interrupts when they are generated with that pin. The port C special options register (PCSO) indicates whether certain port C pins can connect to on-chip peripherals and generate an interrupt at the same time. The remaining port C registers exist on the other ports as well. However, since port C does not have an open-drain capability, there is no open-drain register.

**16.19.10.1 PORT C DATA REGISTER.** When read, the port C data (PCDAT) register always reflects the current status of each line.

PCDAT REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | D4 | D5 | D6 | D7 | D8 | D9 | D10 | D11 | D12 | D13 | D14 | D15 |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 966 | | | | | | | | | | | | | | | |

**16.19.10.2 PORT C DATA DIRECTION REGISTER.** The port C data direction register (PCDIR) is a 16-bit register that is cleared at system reset.

PCDIR REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | 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 | 960 | | | | | | | | | | | | | | | |

**16.19.10.3 PORT C PIN ASSIGNMENT REGISTER.** The port C pin assignment register (PCPAR) is a 16-bit register that is cleared at system reset.

PCPAR REGISTER

| BITS | 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 | 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 | 962 | | | | | | | | | | | | | | | |

**16.19.10.4 PORT C SPECIAL OPTIONS.** The port C special options (PCSO) register is a 16-bit read/write register. Each bit defined in the PCSO corresponds to a port C line (PC8–PC11 and PC14–PC15). The PCSO is cleared at reset.

PCSO REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | — | — | — | — | CD2 | CTS2 | CD1 | CTS1 | — | | DREQ1 | DREQ0 |
| 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 | 964 | | | | | | | | | | | | | | | |

Bits 0–7—Reserved

These bits should be written with zeros.

Bits 0–3—Reserved

These bits should be written with zeros.

CDx—Carrier Detect

0 = PCx is a general-purpose interrupt I/O pin. The SCC internal CDx signal is always asserted. If PCDIR configures this pin as an input, the pin can generate an interrupt to the CPU core, as controlled by the PCINT bits.

1 = PCx is connected to the corresponding SCC signal input in addition to being a general-purpose interrupt pin.

CTSx—Clear To Send

0 = PCx is a general-purpose interrupt I/O pin. The SCC internal $\overline{\text{CTSx}}$ signal is always asserted. If PCDIR configures this pin as an input, the pin can generate an interrupt to the CPU core, as controlled by the PCINT bits.

1 = PCx is connected to the corresponding SCC signal input in addition to being a general-purpose interrupt pin.

Bits 12–13—Reserved

These bits should be written with zeros.

DREQx— DMA Request to the RISC

0 = PCx is a general-purpose interrupt I/O pin, with the direction controlled in PCDIR. If PCDIR configures this pin as an input, the pin can generate an interrupt to the CPU core, as controlled by the PCINT bits.

1 = PCx becomes an external request to the IDMA instead of being a general-purpose interrupt pin. The corresponding PCINT bits control when a request is generated.

**NOTE**

DREQx should only be set if the IDMA is being used.

**16.19.10.5 PORT C INTERRUPT CONTROL REGISTER.** The port C interrupt control (PCINT) register is a 16-bit read/write register. Each defined bit in the PCINT corresponds to a port C line to determine whether the corresponding port C line asserts an interrupt request upon a high-to-low change or any change on the pin. The PCINT is cleared at reset.

PCINT REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | RESERVED | | | | EDM 4 | EDM 5 | EDM 6 | EDM 7 | EDM 8 | EDM 9 | EDM 10 | EDM 11 | EDM 12 | EDM 13 | EDM 14 | EDM 15 |
| 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 | 968 | | | | | | | | | | | | | | | |

Bits 0–3—Reserved.

These bits should be written with zeros.

EDMx—Edge Detect Mode for Line x

The corresponding port C line (PCx) asserts an interrupt request according to the following:

0 = Any change on PCx generates an interrupt request.
1 = High-to low change on PCx generates an interrupt request.

## 16.19.11 Port D Pin Functions

Refer to Table 16-42 for all the port D pin options' default descriptions. Each of the 13 port D pins is independently configured as a general purpose I/O pin if the corresponding port D pin assignment register (PDPAR) is cleared. Each pin is configured as a dedicated on-chip peripheral pin if the corresponding PDPAR bit is set.

When the port pin is configured as a general-purpose I/O pin, the signal direction for that pin is determined by the corresponding control bit in the port D data direction register (PDDIR). The port I/O pin is configured as an input if the corresponding PDDIR is cleared; it is configured as an output if the corresponding PDDIR is set. All PDPAR bits and PDDIR pins are cleared on total system reset, configuring all port D pins as general-purpose input pins.

**Table 16-42. Port D Pin Assignment**

| SIGNAL | PIN FUNCTION | |
|---|---|---|
| | PDPAR = 0 | PDPAR = 1 |
| PD15 | PORT D15 | LD8 |
| PD14 | PORT D14 | LD7 |
| PD13 | PORT D13 | LD6 |
| PD12 | PORT D12 | LD5 |
| PD11 | PORT D11 | LD4 |
| PD10 | PORT D10 | LD3 |
| PD9 | PORT D9 | LD2 |
| PD8 | PORT D8 | LD1 |
| PD7 | PORT D7 | LD0 |
| PD6 | PORT D6 | LCD_AC/OE |
| PD5 | PORT D5 | FRAME/VSYNC |
| PD4 | PORT D4 | LOAD/HSYNC |
| PD3 | PORT D3 | SHIFT/CLK |

## 16.19.12 Port D Registers

Port D has three memory-mapped, read/write, 16-bit control registers.

**16.19.12.1 PORT D DATA REGISTER.** A read of the port D data (PDDAT) register returns the data at the pin, independent of whether the pin is defined as an input or an output. This allows detection of output conflicts at the pin by comparing the written data with the data on the pin. A write to the PDDIR is latched, and if that bit in the PDDIR is configured as an output, the value latched for that bit will be driven onto its respective pin. PDDAT can be read or written at any time. PDDAT is not initialized and is undefined at reset.

PDDAT REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 | D11 | D12 | D13 | D14 | D15 |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 976 | | | | | | | | | | | | | | | |

### 16.19.12.2 PORT D DATA DIRECTION REGISTER.

The port D data direction register (PDDIR) is cleared at system reset.

PDDIR REGISTER

| BITS | 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 | 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 | 970 | | | | | | | | | | | | | | | |

For each DRx bit, the definition is as follows:

0 = The corresponding pin is an input.
1 = The corresponding pin is an output.

### 16.19.12.3 PORT D PIN ASSIGNMENT REGISTER.

The port D pin assignment register (PAPAR) is cleared at system reset.

PDPAR REGISTER

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | 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 | 972 | | | | | | | | | | | | | | | |

For each DDx bit, the definition is as follows:

0 = General-purpose I/O. The peripheral functions of the pin are not used.
1 = Dedicated peripheral function. The pin is used by the internal module.

## 16.20 CPM INTERRUPT CONTROLLER

The CPM interrupt controller (CPIC) is the focal point for all interrupts associated with the CPM and it accepts and prioritizes all the internal and external interrupt requests from all functional blocks associated with the CPM. It is also responsible for generating a vector during the CPU interrupt acknowledge cycle.

The following is a list of the CPM interrupt controller’s important features:

- Twenty-seven interrupt sources (15 internal and 12 external)
- Sources can be assigned to a programmable interrupt level (0–7)
- Programmable priority between SCCs
- Two priority schemes for the SCCs
- Programmable highest priority request
- Fully nested interrupt environment
- Unique vector number for each interrupt source

### 16.20.1 Overview

The CPIC receives interrupts from internal sources, such as the two SCCs, two SMCs, SPI, $I^2C$, PIP, general-purpose timers, and port C parallel I/O pins. The CPIC allows masking of each interrupt source. When multiple events within a CPM sub-block cause the interrupt, each event is also maskable in that CPM sub-block.

All CPM sub-block interrupt sources are prioritized and bits are set in the CPM interrupt pending register (CIPR). All interrupt sources within the CIPR are assigned one programmable priority level (0–7) before the request for an interrupt is sent to the U-Bus. An overview of the MPC821interrupt structure is illustrated in Figure 16-132. The lower half of the figure shows the CPIC.

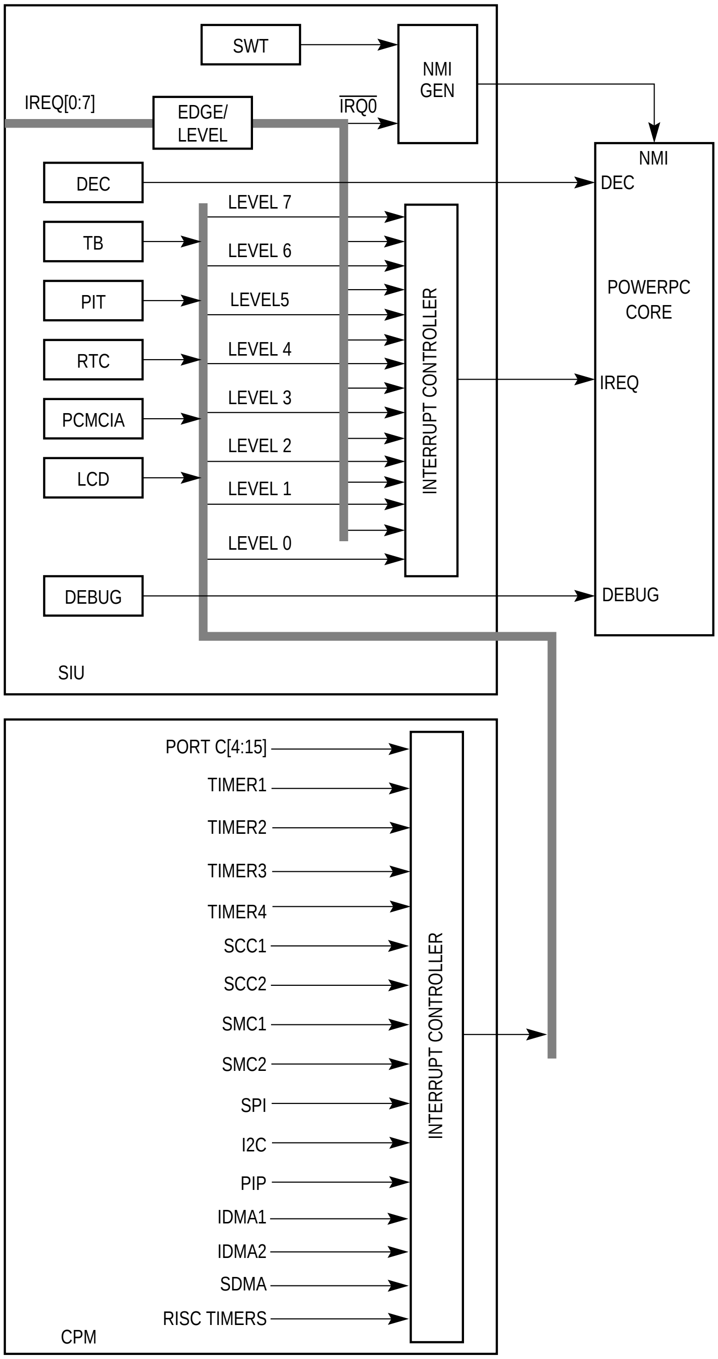


**Figure 16-132. MPC821 Interrupt Structure**

Within the CPM interrupt level, the sources are assigned a priority structure. On the MPC821, some flexibility is given to the user concerning the relative priority of the interrupt sources. This flexibility is in two areas—the ability to modify the relative priority of the SCCs and to choose any interrupt source to be the highest of all the sources.

Once an unmasked interrupt source is pending in the CIPR, the CPIC sends an interrupt request to the U-Bus. This request is at level 0, 1, 2, 3, 4, 5, 6, or 7. The CPIC then waits for the interrupt to be recognized. After the interrupt request is honored by the CPU, the CPU acknowledges the interrupt by setting the IACK bit in the CPM interrupt vector register. When the IACK bit is set, the CIVR is updated with a 5-bit vector corresponding to the sub-block with the highest current priority and the IACK is cleared after one clock cycle.

### 16.20.2 CPM Interrupt Source Priorities

The CPIC has 27 interrupt sources that assert just one programmable interrupt request level to the CP core. The priority between all interrupt sources is shown in Table 16-43. There is some flexibility in the relative ordering of the interrupts in the table, but, in general, the relative priorities are fixed in the descending order shown. An interrupt from the parallel I/O line PC15 has the highest priority and an interrupt from the parallel I/O line PC4 has the lowest. A single interrupt priority number is associated with each table entry.

Notice the lack of SDMA interrupt sources. The SDMA-related interrupts are reported through each individual SCC, SMC, SPI, or $I^2C$ channel. The only true SDMA interrupt source is the SDMA channel bus error entry that is reported when a bus error occurs during an SDMA access. There are two methods to add flexibility to the table of CPM interrupt priorities—the SCC relative priority option and the highest priority option.

**16.20.2.1 SCC RELATIVE PRIORITY.** The relative priority between the two SCCs is programmable and can be dynamically changed. In Table 16-43 there is no entry for SCC1 or SCC2, but rather there are entries for SCCa and SCCb because each of the SCCs can be mapped to any of these locations. This is programmed in the CICR and can be dynamically changed. The user can use this on-the-fly capability to implement a rotating priority.

In addition, the grouping of the locations of the SCCa and SCCb entries has two options—group and spread. In the group scheme, the SCCs are all grouped together at the top of the priority table, ahead of most of the other CPM interrupt sources. This scheme is ideal for applications where all SCCs function at a very high data rate and interrupt latency is very important. In the spread scheme, the SCC priorities are spread over the table so that other sources can have lower interrupt latencies than the SCCs. This scheme is also programmed in the CICR, but it cannot be dynamically modified.

**16.20.2.2 HIGHEST PRIORITY INTERRUPT.** In addition to the SCC relative priority option, the user can choose one interrupt source to be of highest priority. This highest priority interrupt is still within the same interrupt level as the rest of the CPIC interrupts, but is serviced prior to any other interrupt in the table. If the highest priority feature is not used, select PC15 to be the highest priority interrupt and no modifications to the standard interrupt priority order will be made. This highest priority source is dynamically programmable in the CICR and it allows the user to change a normally low priority source into a high priority source for a certain period of time.

### Table 16-43. Prioritization of CPM Interrupt Sources

| NUMBER | PRIORITY LEVEL | INTERRUPT SOURCE DESCRIPTION | MULTIPLE EVENTS |
|---|---|---|---|
| 1F | Highest | Parallel I/O–PC15 | No |
| 1E | | SCCa (Grouped and Spread) | Yes |
| 1D | | SCCb (Grouped) | Yes |
| 1C | | Reserved | Yes |
| 1B | | Reserved | Yes |
| 1A | | Parallel I/O–PC14 | No |
| 19 | | Timer 1 | Yes |
| 18 | | Parallel I/O–PC13 | No |
| 17 | | Parallel I/O–PC12 | No |
| 16 | | SDMA Channel Bus Error | Yes |
| 15 | | IDMA1 | Yes |
| 14 | | IDMA2 | Yes |
| 13 | | SCCb (Spread) | Yes |
| 12 | | Timer 2 | Yes |
| 11 | | RISC Timer Table | Yes |
| 10 | | I2C | Yes |
| F | | Parallel I/O–PC11 | No |
| E | | Parallel I/O–PC10 | No |
| D | | Reserved | Yes |
| C | | Timer 3 | Yes |
| B | | Parallel I/O–PC9 | No |
| A | | Parallel I/O–PC8 | No |
| 9 | | Parallel I/O–PC7 | No |

Table 16-43. Prioritization of CPM Interrupt Sources (Continued)

| NUMBER | PRIORITY LEVEL | INTERRUPT SOURCE DESCRIPTION | MULTIPLE EVENTS |
|---|---|---|---|
| 8 | | Reserved | Yes |
| 7 | | Timer 4 | Yes |
| 6 | | Parallel I/O–PC6 | No |
| 5 | | SPI | Yes |
| 4 | | SMC1 | Yes |
| 3 | | SMC2/PIP | Yes |
| 2 | | Parallel I/O–PC5 | No |
| 1 | | Parallel I/O–PC4 | No |
| 0 | Lowest | Reserved | — |

**16.20.2.3 NESTED INTERRUPTS.** The CPIC supports a fully nested interrupt environment that allows a higher priority interrupt (from another CPM source) to suspend a lower priority interrupt service routine. This nesting is achieved by the CISR. The CPIC prioritizes all interrupt sources based on their assigned priority level. The highest priority interrupt request is presented to the CPU core for servicing. The CPU acknowledges the interrupt by setting the IACK bit in the CIVR.

Following the setting of the IACK bit, the vector number corresponding to this interrupt is made available to the CPU in the CIVR and the interrupt request is cleared. If there are remaining interrupt requests, they are then prioritized and another interrupt request can be presented to the CPU core. At interrupt, the interrupt mask bit in the CPU machine status register is cleared to disable further interrupt requests until the software is ready to handle them.

The CISR can be used to allow a higher priority interrupt within the same interrupt level to be presented to the CPU core before a lower priority interrupt service is completed. Each bit in the CISR corresponds to a CPM interrupt source. When the CPU acknowledge the interrupt by setting the IACK bit, the in-service bit in the CISR register is set by the CPIC for that interrupt source.

The setting of the bit prevents any subsequent CPM interrupt requests at this priority level or lower (within the CPIC interrupt table), until the servicing of the current interrupt has completed and the in-service bit is cleared by the user. Pending interrupts for these sources are still set in the CPIC during this time. Thus, in the interrupt service routine for the CPM interrupts, the user can enable the core interrupt mask to allow higher priority interrupts within this level to generate an interrupt request. This capability provides nesting of interrupt requests for CPM interrupt level sources.

### 16.20.3 Masking Interrupt Sources in the CPM

By programming the CIMR, the user may mask the CPM interrupts to prevent an interrupt request to the CPU core. Each bit in the CIMR corresponds to one of the CPM interrupt sources. To enable an interrupt, write a 1 to the corresponding CIMR bit. When a masked CPM interrupt source has a pending interrupt request, the corresponding bit in the CIPR is still set, even though the interrupt is not generated to the CPU core. By masking all interrupt sources in the CIMR, the user can implement a polling interrupt servicing scheme for the CPM interrupts.

When a CPM interrupt source has multiple interrupting events, the user can individually mask these events by programming a mask register within that block. Table 16-44 indicates the interrupt sources that have multiple interrupting events. Figure 16-133 illustrates an example of how the masking occurs, using an SCC as an example.

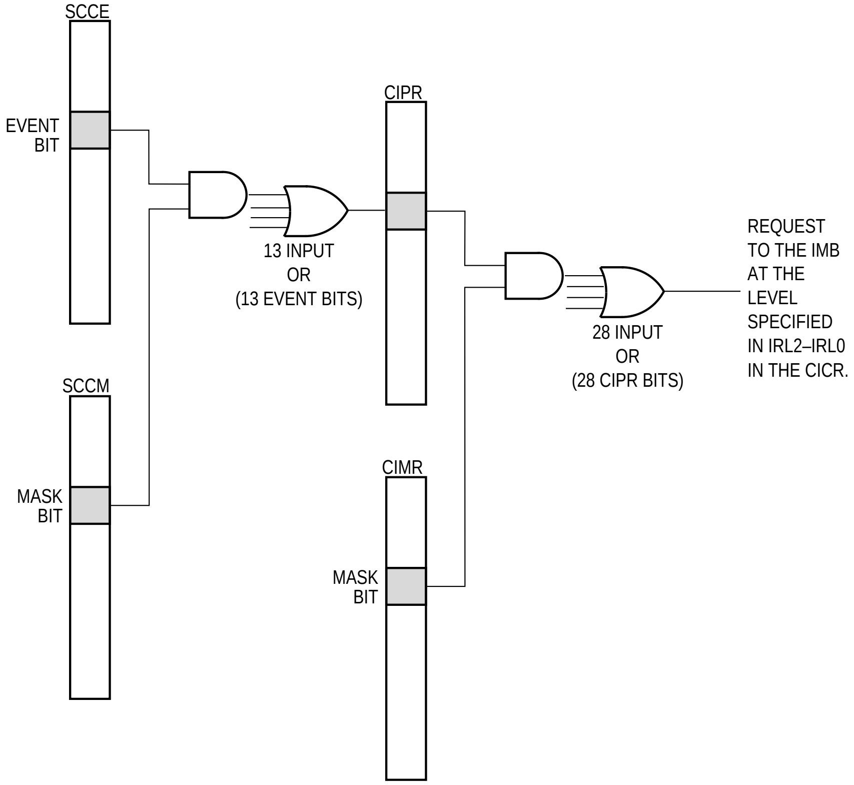

SCCE
EVENT BIT
SCCM
MASK BIT
13 INPUT OR (13 EVENT BITS)
CIPR
CIMR
MASK BIT
28 INPUT OR (28 CIPR BITS)
REQUEST TO THE IMB AT THE LEVEL SPECIFIED IN IRL2–IRL0 IN THE CICR.

Figure 16-133. Interrupt Request Masking

### 16.20.4 Interrupt Vector Generation and Calculation

Pending unmasked CPM interrupts are presented to the CPU core in order of priority. The CPU core responds to an interrupt request by setting the IACK bit in the CIVR. The interrupt vector that allows the core to locate the interrupt service routine is made available to the CPU by reading the CIVR. For CPM interrupts, the CPIC passes an interrupt vector corresponding to the highest priority, unmasked, pending interrupt. The CPIC encoding of the five low-order bits of the interrupt vector is listed in Table 16-44.

**Table 16-44. Encoding the Interrupt Vector**

| INTERRUPT NUMBER | INTERRUPT SOURCE DESCRIPTION | INTERRUPT VECTOR |
|---|---|---|
| 1F | Parallel I/O—PC15 | 11111 |
| 1E | SCC1 | 11110 |
| 1D | SCC2 | 11101 |
| 1C | Reserved | 11100 |
| 1B | Reserved | 11011 |
| 1A | Parallel I/O—PC14 | 11010 |
| 19 | Timer 1 | 11001 |
| 18 | Parallel I/O—PC13 | 11000 |
| 17 | Parallel I/O—PC12 | 10111 |
| 16 | SDMA Channel Bus Error | 10110 |
| 15 | IDMA1 | 10101 |
| 14 | IDMA2 | 10100 |
| 13 | Reserved | 10011 |
| 12 | Timer 2 | 10010 |
| 11 | RISC Timer Table | 10001 |
| 10 | I2C | 10000 |
| F | Parallel I/O—PC11 | 01111 |
| E | Parallel I/O—PC10 | 01110 |
| D | Reserved | 01101 |
| C | Timer 3 | 01100 |
| B | Parallel I/O—PC9 | 01011 |
| A | Parallel I/O—PC8 | 01010 |
| 9 | Parallel I/O—PC7 | 01001 |
| 8 | Reserved | 01000 |

### Table 16-44. Encoding the Interrupt Vector (Continued)

| INTERRUPT NUMBER | INTERRUPT SOURCE DESCRIPTION | INTERRUPT VECTOR |
|---|---|---|
| 7 | Timer 4 | 00111 |
| 6 | Parallel I/O—PC6 | 00110 |
| 5 | SPI | 00101 |
| 4 | SMC1 | 00100 |
| 3 | SMC2 / PIP | 00011 |
| 2 | Parallel I/O—PC5 | 00010 |
| 1 | Parallel I/O—PC4 | 00001 |
| 0 | Error | 00000 |

Notice that the interrupt vector table is the same as the CPM interrupt priority table except for two differences. First, the SCC vectors are fixed; they are not affected by the SCC group mode, spread mode, or the relative priority order of the SCCs. Second, an error vector exists as the last entry in this table. The error vector is issued by the CPM if an interrupt is requested by the CPM but was masked by the user prior to being serviced by CPU core and if no other pending interrupts for the CPM are present. The user should provide an error interrupt service routine, even if it is simply an RFI instruction.

## 16.20.5 Programming Model

There are four CPIC registers—CICR, CIPR, CIMR, and CISR.

**16.20.5.1 CPM INTERRUPT CONFIGURATION REGISTER.** The 24-bit read/write CPM interrupt configuration register (CICR) defines the request level for the CPM interrupts, the priority between the SCCs, and the highest priority interrupt. The CICR, which can be dynamically changed by the user, is cleared at reset.

CICR

| **BITS** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **FIELD** | RESERVED | | | | | | | | RESERVED | | RESERVED | | SCbP | | SCaP | |
| **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** | 940 | | | | | | | | | | | | | | | |
| **BITS** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **FIELD** | IRL0 | IRL1 | IRL2 | HP0 | HP1 | HP2 | HP3 | HP4 | IEN | — | | | | | | SPS |
| **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** | 942 | | | | | | | | | | | | | | | |

SCbP—SCCb Priority Order

These two bits define which SCC asserts it's request in the SCCb priority position. The user should not program the same SCC to more than one priority position (a or b). These bits can be changed dynamically.

00 = SCC1 asserts it's request in the SCCb position.
01 = SCC2 asserts it's request in the SCCb position.

10 = Reserved.11 =Reserved.SCaP—SCCa Priority Order

These two bits define which SCC asserts it's request in the SCCa priority position. The user should not program the same SCC to more than one priority position (a or b). These bits can be changed dynamically.

00 = SCC1 asserts it's request in the SCCa position.
01 = SCC2 asserts it's request in the SCCa position.

10 = Reserved.11 =Reserved.IRL[0:2]—Interrupt Request Level

The IRL field contains the priority request level of the interrupt from the CPM that is sent to the CPU core. Level 0 indicates the highest priority interrupt and level 7 indicates the lowest. The IRL field is initialized to zero during reset. In most systems, value $4 is a good value to choose for the IRL field.

HP[0:4]—Highest Priority

These bits specify the 5-bit interrupt number of the single CPIC interrupt source that is advanced to the highest priority in the table. These bits can be dynamically modified. To keep the original priority order intact, simply program these bits to 11111.

IEN—Interrupt Enable

The IEN bit acts as a master enable for the CPM interrupts.

0 = CPM interrupts are disabled
1 = CPM interrupts are enabled

Bits 25–30—Reserved

SPS—Spread Priority Scheme

This bit, which selects the relative SCC priority scheme, cannot be changed dynamically.

0 = Grouped. The SCCs are grouped by priority at the top of the table.
1 = Spread. The SCCs are spread by priority in the table.

**16.20.5.2 CPM INTERRUPT PENDING REGISTER.** Each bit in the 32-bit read/write CPM interrupt pending register (CIPR) corresponds to a CPM interrupt source. When a CPM interrupt is received, the CPIC sets the corresponding bit in the CIPR.

CIPR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | PC15 | SCC1 | SCC2 | — | — | PC14 | TIMER1 | PC13 | PC12 | SDMA | IDMA1 | IDMA2 | — | TIMER2 | R–TT | I2C |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 944 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

| BITS | 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 | 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 | 946 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

In a vectored interrupt scheme, the CIPR clears the CIPR bit when the CPU acknowledges the interrupt by setting the IACK bit in the CIVR. The vector number corresponding to the CPM interrupt source is then available for the CPU in the CIVR. The CIPR bit is not cleared if an event register exists for that interrupt source. Event registers exist for interrupt sources that have multiple source events. For example, the SCCs have multiple events that cause SCC interrupts.

In a polled interrupt scheme, the user must periodically read the CIPR. When a pending interrupt is handled, the user clears the corresponding bit in the CIPR. However, if an event register exists, the unmasked event register bits should be cleared instead, causing the CIPR bit to be cleared.) To clear a bit in the CIPR, the user writes a 1 to that bit. Since the user can only clear bits in this register, bits written as zeros are not affected. The CIPR is cleared at reset.

**NOTE**

The SCC CIPR bit positions are not changed according to the relative priority between SCCs (as determined by the SCxP and SPS bits in the CICR). No bit in the CIPR is set if the error vector is issued.

**16.20.5.3 CPM INTERRUPT MASK REGISTER.** Each bit in the 32-bit read/write CPM interrupt mask register (CIMR) corresponds to a CPM interrupt source. The user masks an interrupt by clearing the corresponding bit in the CIMR and enables an interrupt by setting the corresponding bit in the CIMR. When a masked CPM interrupt occurs, the corresponding bit in the CIPR is still set, regardless of the CIMR bit, but no interrupt request is passed to the CPU core.

If a CPM interrupt source is requesting interrupt service when the user clears it's CIMR bit, the request stops. If the user sets the CIMR bit later, a previously pending interrupt request is processed by the CPU core, according to it's assigned priority. The CIMR can be read by the user at any time and is cleared at reset.

CIMR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | PC15 | SCC1 | SCC2 | — | — | PC14 | TIME R1 | PC13 | PC12 | SDM A | IDMA 1 | IDMA 2 | — | TIME R2 | R–TT | I2C |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 948 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | PC11 | PC10 | — | TIME R3 | PC9 | PC8 | PC7 | — | TIME R4 | PC6 | SPI | SMC1 | SMC2 / PIP | PC5 | PC4 | — |
| 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 | 94A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## NOTE

The SCC CIMR bit positions are not affected by the relative priority between SCCs (as determined by the SCxP and SPS bits in the CICR). To clear bits that were set by multiple interrupt events, the user must clear all the unmasked events in the corresponding event register. If a bit in the CIMR is masked at the same time that the corresponding CIPR bit causes an interrupt request to the CPU, then the interrupt is not processed, but the error vector is issued if the interrupt acknowledge cycle occurs with no other CPM interrupts pending. Thus, the user should always include an error vector routine, even if it just contains the RFI instruction. The error vector cannot be masked.

**16.20.5.4 CPM INTERRUPT IN-SERVICE REGISTER.** Each bit in the 32-bit read/write CPM interrupt in-service register (CISR) corresponds to a CPM interrupt source. In a vectored interrupt environment, the CPIC sets the CISR bit when the CPU acknowledges the interrupt by setting the IACK bit in the CPM interrupt vector register. The user's interrupt service routine must clear this bit after servicing is complete. If an event register exists for this peripheral, it's bits would normally be cleared as well. To clear a bit in the CISR, the user writes a 1 to that bit. Since the user can only clear bits in this register, bits written as zeros will not be affected. The CISR is cleared at reset.

CISR

| **BITS** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **FIELD** | PC15 | SCC1 | SCC2 | — | — | PC14 | TIMER1 | PC13 | PC12 | SDMA | IDMA1 | IDMA2 | — | TIMER2 | R–TT | I2C |
| **RESET** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **R/W** | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| **ADDR** | 94C |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **BITS** | **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** | 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** | 94E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

This register can be read by the user to determine which interrupt requests are currently in progress (the interrupt handler started execution) for each CPM interrupt source. More than one bit in the CISR can be a 1 if higher priority CPM interrupts are allowed to interrupt lower priority level interrupts within the same CPM interrupt level. For example, the TIMER2 interrupt routine could interrupt the handling of the TIMER3 routine using a special nesting technique described earlier. During this time, the user would see both the TIMER3 and the TIMER2 bits simultaneously set in the CISR.

### NOTE

The SCC CISR bit positions are not affected by the relative priority between SCCs (as determined by the SCxP and SPS bits in the CICR). If the error vector is taken, no bit in the CISR is set. All undefined bits in the CISR return zeros when read. The user can control the extent to which CPM interrupts can interrupt other CPM interrupts by selectively clearing the CISR. A new interrupt is processed if it has a higher priority than the higher priority interrupt having it's CISR bit set. Thus, if an interrupt routine sets the interrupt mask bit in the CPU core and also clears it's CISR bit at the beginning of the interrupt routine, a lower priority interrupt can interrupt the higher one, as long as the lower priority interrupt is of higher priority than any other CISR bits that are currently set.

**16.20.5.5 CPM INTERRUPT VECTOR REGISTER.** The CPM interrupt vector register (CIVR) is a 16-bit register. Bits 0-4 of the register contain the interrupt vector number. To update the register with the current interrupt vector number, the CPU should set the IACK bit (bit 15). The bit is cleared after one clock cycle. The register may be read anytime. Write operation to bits 5-15 has no effect and they are always read as zeros.

CIVR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | VECTOR NUMBER | | | | | 0 | | | 0 | | | | | | | IACK |
| 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 | 930 | | | | | | | | | | | | | | | |

### 16.20.6 Interrupt Handler Examples

The following examples illustrate proper interrupt handling of CPM interrupts. However, nesting interrupts within the CPM interrupt level is not shown in the following examples.

**16.20.6.1 EXAMPLE 1—PC6 INTERRUPT HANDLER.** In this example, the CPIC hardware clears the PC6 bit in the CIPR during the interrupt acknowledge cycle. This is an example of a handler for an interrupt source without multiple events.

1. Set the IACK bit in the CIVR.
2. Read the vector to access interrupt handler.
3. Handle the event associated with a change in the state of the port C6 pin.
4. Clear the PC6 bit in the CISR.
5. Execute the RFI instruction.

**16.20.6.2 EXAMPLE 2—SCC1 INTERRUPT HANDLER.** In this example, the CIPR bit SCC1 remains set as long as one or more unmasked event bits remain in the SCCE1 register. This is an example of a handler for an interrupt source with multiple events. Notice that the bit in CIPR does not need to be cleared by the handler, but the bit in CISR does.

1. Set the IACK bit in the CIVR.
2. Read the vector to access interrupt handler.
3. Immediately read the SCC1 event register into a temporary location.
4. Decide which events in the SCCE1 will be handled in this handler and clear those bits as soon as possible. (SCCE bits are cleared by writing ones).
5. Handle the events in the SCC1 Rx or Tx BD tables.
6. Clear the SCC1 bit in the CISR.
7. Execute the RFI instruction. If any unmasked bits in SCCE1 remain at this time (either not cleared by the software or set by the MPC821 during the execution of this handler), this interrupt source is made pending again immediately following the RFI instruction.

# SECTION 17
# PCMCIA INTERFACE

## 17.1 OVERVIEW

The PCMCIA host adapter module provides all necessary control logic for a PCMCIA socket interface and only requires external analog power switching logic and buffering to be provided. Additional external analog buffers allow the PCMCIA host adapter module to support up to two PCMCIA sockets. The PCMCIA interface contains the following features:

- A host adapter interface fully compliant with the PCMCIA standard, release 2.1+ (PC Card -16).
- Support for up to two PCMCIA sockets, requiring only external buffering and analog switching logic.
- Support for eight memory or I/O windows that can be assigned to either of the two sockets.

## 17.2 SYSTEM CONFIGURATION

This system configuration, electrical isolation between the sockets and system bus must be accomplished using external buffers and bus transceivers. These buffers also provide the voltage conversion required from the MPC821 3.3 to 5 V cards. These proponents should be powered by the $V_{CC}$ card. Since the MPC821 is 5 V-friendly (it accepts 5 V inputs while generating 3.3 V outputs) no conversion is required for inputs. The PCMCIA host adapter provides the control signals necessary for demultiplexing the signals shared among the sockets. Figure 17-1 illustrates a system configuration consisting of two PCMCIA sockets.

## 17.3 PCMCIA MODULE SIGNAL DEFINITIONS

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 pins of the system bus. Figure 17-1 illustrates the external signals for the PCMCIA host adapter module.

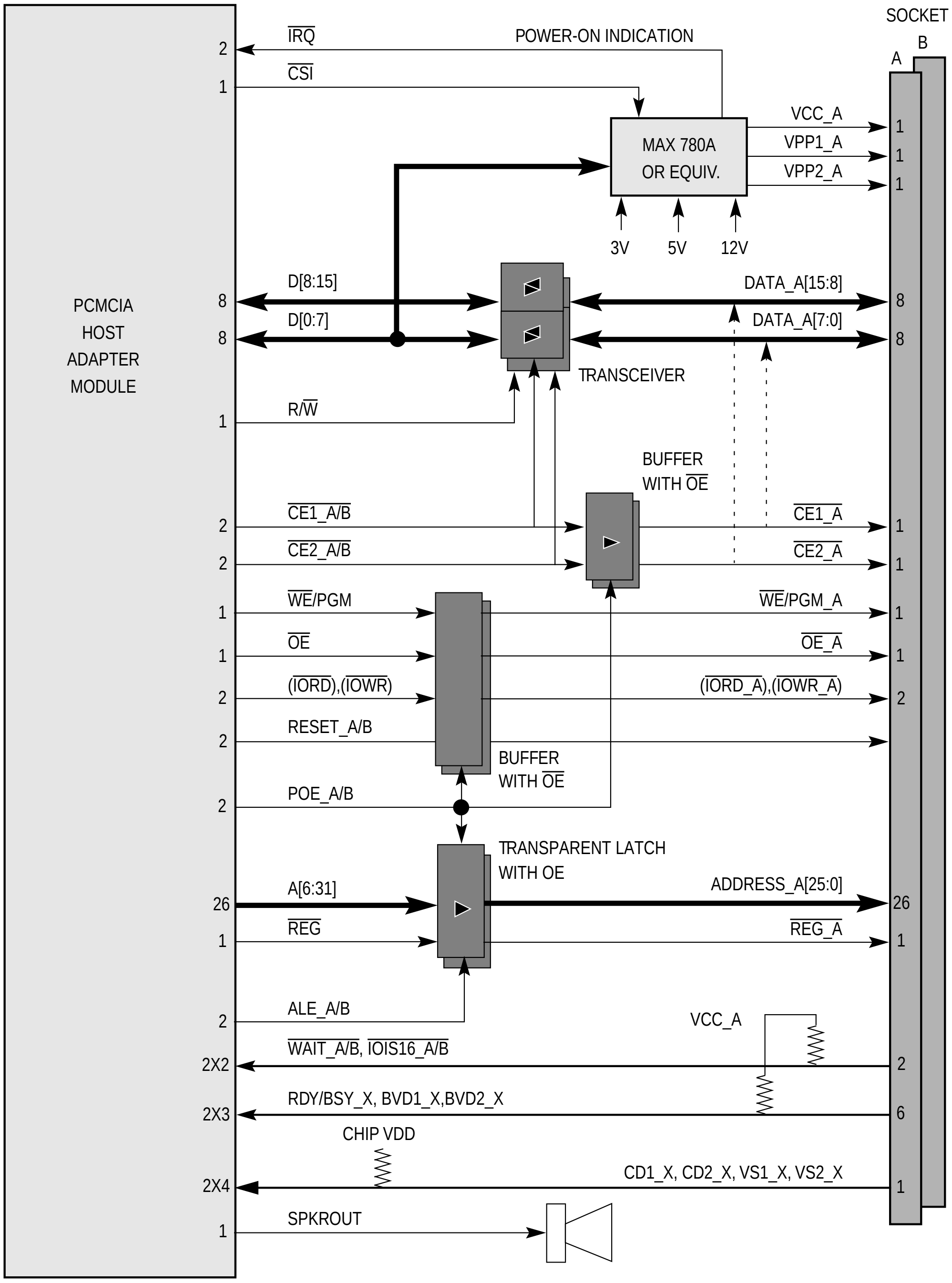


**Figure 17-1. System With Two PCMCIA Sockets**

### 17.3.1 PCMCIA Cycle Control Signals

**A6–A31**

Address Bus—Output. Signals A6 through A31 should be buffered to generate the socket signals A25 through A0. These are address bus output lines that allow direct addressing of up to 64 megabytes of memory on each PCMCIA card. A6 is the most-significant bit and A31 is the least-significant bit.

**$\overline{\text{REG}}$**

Attribute Memory Select—Output. When this signal 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). On accesses with $\overline{\text{REG}}$ asserted, accesses to common memory or DMA devices are blocked. When no PCMCIA access is performed, this signal is Transfer Size 0 (TSIZ0).

**$\overline{\text{CE1\_x}}$, $\overline{\text{CE2\_x}}$**

Card Enable—Output. When a PCMCIA access is performed, the $\overline{\text{CE1}}$ and $\overline{\text{CE2}}$ signals are card enable output signals. The $\overline{\text{CE1}}$ signal enables even bytes and $\overline{\text{CE2}}$ enables odd bytes.

**Table 17-1. Card Enable As Driven By The MPC821**

| PORT SIZE | ACCESS SIZE | MPC821: A31 (SLOT: A0) | $\overline{\text{CE2}}$ | $\overline{\text{CE1}}$ |
|---|---|---|---|---|
| 8 Bits | 16-Bit (Only Even) | 0 | 1 | 0 |
| | 8-Bit Odd | 1 | 1 | 0 |
| | 8-Bit Even | 0 | 1 | 0 |
| 16 Bits | 16-Bit (Only Even) | 0 | 0 | 0 |
| | 8-Bit Odd | 1 | 0 | 1 |
| | 8-Bit Even | 0 | 1 | 0 |
| | No Access | X | 1 | 1 |

**D0–D15**

Data Bus—Bidirectional. Signals D0 through D15 constitute the bidirectional data bus. The most-significant bit is D0 and the least-significant bit is D15.

**$\overline{\text{WAIT\_x}}$**

Extend Bus Cycle—Input. This signal is asserted by the PC card to delay completion of the pending memory or I/O cycle.

### R/W

External Transceiver Direction—Output. This signal is part of the MPC821 bus. It is asserted (driven high) during any read cycles of the MPC821 and negated (driven low) during write cycles. It is used in the PCMCIA interface to control the direction of the data bus transceivers.

### IORD_x

I/O Read—Output. During PCMCIA accesses, this signal is asserted together with REG_x and it is used to read data from the PC card I/O space. IORD_x is valid only when the REG_x and at least one of the CE1_x and CE2_x signals is also asserted.

### IOWR_x

I/O Write—Output. During PCMCIA accesses, this signal is asserted with REG_x. It is used to latch data into the PC card I/O space. IOWR_x is only valid when the REG_x and at least one of the CE1_x and CE2_x signals is also asserted.

### OE_x

Output Enable—Output. During PCMCIA accesses, the OE_x signal is used to drive memory read data from a PC card in a PCMCIA socket.

### WE_x

Write Enable/Program—Output. During PCMCIA accesses, the WE_x signal is used to latch memory write data to the PC card in a PCMCIA socket. This signal can also be used as the programming strobe for PC cards employing programmable memory technologies.

### ALE_x

Address Latch Enable—Output. These signals are strobes to control the external buffers of the address and REG signals for the appropriate PC card being accessed. ALE_A asserts (driven high) when socket A is being accessed and ALE_B asserts (driven high) when socket B is being accessed.

### IOIS16

I/O Port Is 16 Bits—Input. When the card and it’s socket are programmed for I/O interface operation, this signal is used as IOIS16 and must be asserted by the card when the address on the bus corresponds to an address on the PC card and the I/O port which is being addressed is capable of 16-bit accesses. If the I/O region in which the address resides is programmed as 8 bits wide, then the IOIS16 signal is ignored.

### 17.3.2 PCMCIA Input Port Signals

The following signals are used by a PCMCIA slot to indicate card status. The MPC821 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 and these signals can be used by a system 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_x(7) which has extended edge detection capability and IP_x(2) which serves as $\overline{\text{IOIS16\_x}}$ cycle control signals for PCMCIA cycles.

**$\overline{\text{VS1\_x}}$, $\overline{\text{VS2\_x}}$**

Voltage Sense—Input. These signals are used as VS1 and VS2 and are generated by PC cards and they notify the socket of the card $V_{CC}$ requirement. These signals are connected to IP_x(0:1) pins.

**WP**

Write Protect—Input. When the card and its socket are programmed for memory interface operation, this signal is used as WP and reflects the status of the write-protect switch on the PC card. It must be asserted by the PC card when the PC card switch is enabled and negated when the switch is disabled. For a PC card that is writeable without a switch, this signal must be connected to ground. If the PC card is permanently write-protected this signal must be connected to $V_{CC}$. These signals are connected to IP_x(2) pins.

**$\overline{\text{CD1\_x}}$, $\overline{\text{CD2\_x}}$**

Card Detect—Input. These signals provide for 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 to IP_x4 and IP_x3 pins, respectively.

**BVD1_x, BVD2_x**

Battery Voltage Detect—Input. When the card and it’s socket are programmed for memory interface operation, these signals are used as BVD1_x and BVD2_x and are generated by PC cards having an onboard battery. These signals report the condition of the battery. 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 pins, respectively.

**STSCHG**

Status Change—Input. When the card and its socket are programmed for I/O interface operation, the BVD1_x signal is used as status change (STSCHG) and is generated by I/O PC cards. The STSCHG signal 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 set to zero. The STSCHG signal must be asserted when both bits are set to one.

### SPKR

Speaker—Input. When the card and its socket are programmed for I/O interface operation, the BVD2_x signal is used as digital audio (SPKR) and it is generated by I/O PC cards. The SPKR signal must be used to provide the socket's single amplitude (digital) audio waveform to the system. The SPKR signal of all sockets programmed for I/O interface operation are XOR'd together and routed through the speaker out signal (SPKROUT).

### RDY/BSY_x

Ready/Busy—Input. When the card and it's 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. This signal must be held negated when no interrupt is requested. These signals are connected to IP_x(7) pins.

## 17.3.3 PCMCIA Output Port Signals

The following signals are used by a PCMCIA slot to control the RESET input and output enable of the buffers to the card. The MPC821 provides a way for the software to control the output signal state. This function is not necessarily specific to the PCMCIA interface these signals can be used by a system as a general-purpose output port. These signals appear on pins OP(0:3).

### RESET_x

Card Reset—Output. This signal is 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 the OP(0) pin and RESET_B is connected to the OP(2) pin.

### POE_X

PCMCIA Buffers Output Enable—Output. This line is an output port line reflecting the value of the CAOE and CBOE bits in the PCMCIA interface power control register. It should be used to three-state the address pins and strobe lines addressing each one of the slots. POE_A is connected to the OP(1) pin and POE_B is connected to the OP(3) pin.

### 17.3.4 Other PCMCIA Signals

The following special function signals are used by the PCMCIA socket. Their function is not necessarily specific to PCMCIA.

**$\overline{\text{IRQ}}$**

Power Is On—Input. Two of the $\overline{\text{IRQ}}$ signals provided for general-purpose interrupt requests can be used by the card power supply circuitry to notify the MPC821 processor when the power supply to the card has reached the full required voltage.

**SPKROUT**

Speaker out—Output. This signal provides a digital audio waveform to be driven to the system's speaker. It is generated as a logic exclusive or of the SPKRA and SPKRB input signals.

## 17.4 OPERATION DESCRIPTION

### 17.4.1 Memory-Only Cards

Table 17-2 lists the worst-case conditions of host programming memory cards.

**Table 17-2. Host Programming For Memory Cards**

| MEMORY ACCESS TIME | 600 NS | | | 200 NS | | | 150 NS | | | 100 NS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | STP | LNG | HLD | STP | LNG | HLD | STP | LNG | HLD | STP | LNG | HLD |
| CLK CYCLE | 100 | 300 | 150 | 30 | 120 | 90 | 20 | 80 | 75 | 15 | 60 | 50 |
| 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 |

NOTES:
1. Because the minimum hold time is one clock, the real access time is, access time + one clock.
2. Hold time and setup time (HLD and STP) are the read or write worst-case.
3. The worst-case hold time is data disable from $\overline{\text{OE}}$.
4. The worst-case setup time is address to strobe.
5. Length (LNG) is the minimum strobe time.
6. This table assumes no $\overline{\text{WAIT}}$ usage. If using $\overline{\text{WAIT}}$ signal, the minimum strobe time is at least 35 ns + 1 system clock.

### 17.4.2 I/O Cards

Table 17-3 lists the worst-case conditions of host programming I/O cards.

**Table 17-3. Host Programming For I/O Cards**

| | 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 |

NOTE: Setup time worst-case is for a write. In these cases, setup=data_set_up_befor_iord +1 clock.

### 17.4.3 Interrupts

Each input from the PCMCIA card to the host (BVD, CD, RDY, VS) is sampled on the PIPR and any change on these bits is reported there. The contents of the PSCR is logically AND'ed with the PER to generate a "PCMCIA I/F interrupt". The interrupt level for the exception generated is programmable by the user and the PCMCIA I/F can generate an additional interrupt for the RDY/$\overline{\text{IRQ}}$ signal that can trigger on level (low or high) or edge (fall or rise) of the input signal.

### 17.4.4 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.

### 17.4.5 Reset and Three-State Control

The user can reset the PCMCIA cards or disable the output of the external latches by writing to a specific bit in the PGCR.

### 17.4.6 DMA

The MPC821 DMA module with the CPM microcode provides two independent DMA (IDMA) channels. Refer to the **Section 16.11 IDMA Emulation** for details. 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 the PRS field in the appropriate POR and when configured appropriately, the PCMCIA controller supplies the required signaling to the socket. Notice that DMA to and from the PCMCIA interface is performed through dual-access DMA transfers.

DMA requests can be supplied through either the SPKR, $\overline{\text{IOIS16}}$, or INPACK signals. To support the DMA function, the slot signal INPACK should be connected to DREQ0 for slot A or DREQ1 for slot B. The actual source used for a DMA request is programmed through the CADREQ/CBDREQ field in the PGCRA/PGCRB for slot A/B. If the internal DMA request is enabled, then port C should be programmed to not be DREQ0/DREQ1. When the internal DMA request is disabled, then the DMA request is assumed to be DREQ0/DREQ1. In this case port C should be programmed such that the pin of PC15/14 is assigned as DREQ0/DREQ1.

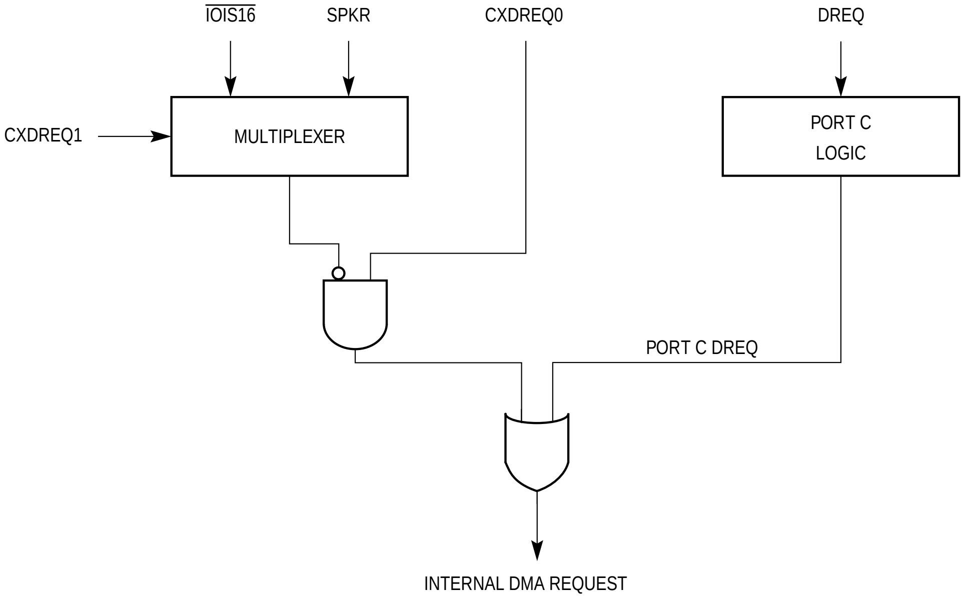


**Figure 17-2. Internal DMA Request Logic**

## 17.5 PROGRAMMING MODEL

The following section describes the PCMCIA interface programming model. Generally, all registers are memory-mapped within the internal control register area. The following registers are used to control the PCMCIA interface.

| 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 | PCMCIA Base Register (Per Window) |
| POR | PCMCIA Option Register (Per Window) |

### Table 17-4. PCMCIA Interface Input Pins Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 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/SPKR in for Card A | |
| 6 | CABVD1 | Battery Voltage/STSCHG in for Card A | |
| 7 | CARDY | RDY/$\overline{\text{IRQ}}$ of Card A Pin | |
| 8-15 | Reserved | | |
| 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/SPKR in for Card B | |
| 22 | CBBVD1 | Battery Voltage/STSCHG in for Card B | |
| 23 | CBRDY | RDY/$\overline{\text{IRQ}}$ of Card B Pin | |
| 24-31 | Reserved | | |

**Table 17-5. PCMCIA Interface Status Changed Register**

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 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/SPKR in for Card A Changed | |
| 6 | CABVD1_C | Battery Voltage/STSCHG in for Card A Changed | |
| 7 | Reserved | | |
| 8 | CARDY_L | RDY/$\overline{\text{IRQ}}$ of Card A Pin is Low | |
| 9 | CARDY_H | RDY/$\overline{\text{IRQ}}$ of Card A Pin is High | |
| 10 | CARDY_R | RDY/$\overline{\text{IRQ}}$ of Card A Pin Rising Edge Detected | |
| 11 | CARDY_F | RDY/$\overline{\text{IRQ}}$ of Card A Pin Falling Edge Detected | |
| 12-15 | Reserved | | |
| 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/SPKR in for Card B Changed | |
| 22 | CBBVD1_C | Battery Voltage/STSCHG in for Card B Changed | |
| 23 | Reserved | | |
| 24 | CBRDY_L | RDY/$\overline{\text{IRQ}}$ of Card B Pin is Low | |
| 25 | CBRDY_H | RDY/$\overline{\text{IRQ}}$ of Card B Pin is High | |
| 26 | CBRDY_R | RDY/$\overline{\text{IRQ}}$ of Card B Pin Rising Edge Detected | |
| 27 | CBRDY_F | RDY/$\overline{\text{IRQ}}$ of Card B Pin Falling Edge Detected | |
| 28-31 | Reserved | | |

NOTE: Reset value: Not initialized.

### Table 17-6. PCMCIA Interface Enable Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0 | CA_EVS1 | Enable for Voltage Sense 1 for Card A Changed | 0 = Disable interrupt on any change in the relevant pin.<br>1 = Enable interrupt on changes of the relevant pin. |
| 1 | CA_EVS2 | Enable for Voltage Sense 2 for Card A Changed | |
| 2 | CA_EWP | Enable for Write Protect for Card A Changed | |
| 3 | CA_ECD2 | Enable for Card Detect 2 for Card A Changed | |
| 4 | CA_ECD1 | Enable for Card Detect 1 for Card A Changed | |
| 5 | CA_EBVD2 | Enable for Battery Voltage/SPKR in for Card A Changed. | |
| 6 | CA_EBVD1 | Enable for Battery Voltage/STSCHG in for Card A Changed. | |
| 7 | Reserved | | |
| 8 | CA_ERDY_L | Enable for RDY/$\overline{\text{IRQ}}$ of Card A Pin is Low | |
| 9 | CA_ERDY_H | Enable for RDY/$\overline{\text{IRQ}}$ Card A Pin is High | |
| 10 | CA_ERDY_R | Enable for RDY/$\overline{\text{IRQ}}$ Card A Pin Rising Edge Detected | |
| 11 | CA_ERDY_F | Enable for RDY/$\overline{\text{IRQ}}$ Card A Pin Falling Edge Detected | |
| 12-15 | Reserved | | |
| 16 | CB_EVS1 | Enable for Voltage Sense 1 for Card B Changed | 0 = Disable interrupt on any change in the relevant pin.<br>1 = Enable interrupt on changes of the relevant pin. |
| 17 | CB_EVS2 | Enable for Voltage Sense 2 for Card B Changed | |
| 18 | CB_EWP | Enable for Write Protect for Card B Changed | |
| 19 | CB_ECD2 | Enable for Card Detect 2 for Card B Changed | |
| 20 | CB_ECD1 | Enable for Card Detect 1 for Card B Changed | |
| 21 | CB_EBVD2 | Enable for Battery Voltage/SPKR in for Card B Changed | |
| 22 | CB_EBVD1 | Enable for Battery Voltage/STSCHG in for Card B Changed | |
| 23 | Reserved | | |
| 24 | CB_ERDY_L | Enable for RDY/$\overline{\text{IRQ}}$ of Card B Pin is Low | |
| 25 | CB_ERDY_H | Enable for RDY/$\overline{\text{IRQ}}$ Card B Pin is High | |

### Table 17-6. PCMCIA Interface Enable Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 26 | CB_ERDY_R | Enable for RDY/$\overline{\text{IRQ}}$ Card B Pin Rising Edge Detected | |
| 27 | CB_ERDY_F | Enable for RDY/$\overline{\text{IRQ}}$ Card B Pin Falling Edge Detected | |
| 28-31 | Reserved | | |

NOTE: Reset value: 0x00000000

### Table 17-7. PCMCIA Interface General Control Register A

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-7 | CAIREQLVL(0:7) | Define Interrupt Level for IREQ for Card A | Note: Only one bit of this field should be set at any given time. |
| 8-15 | CASCHLVL(0:7) | Define Interrupt Level for STSCHG for Card A | Note: Only one bit of this field should be set at any given time. |
| 16-17 | CADREQ | Define pin to be used as internal DMA request to the on-chip DMA controller (Channel 0). | 0x = Disable internal DMA request from slot A.<br>10 = Enable $\overline{\text{IOIS16\_A}}$ as internal DMA request for slot A.<br>1 1 = Enable SPKR_A as internal DMA request for slot A. |
| 18-23 | Reserved | | |
| 24 | CAOE | Card A Output Enable | This bit is reflected on the OP(1) pin used to three-state the external buffers when the card power is activated. |
| 25 | CARESET | Card A Reset | This bit is reflected on the OP(0) pin used to reset the card. |
| 26-31 | Reserved | | |

### Table 17-8. PCMCIA Interface General Control Register B

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-7 | CBIRQLVL(0:7) | Define Interrupt Level for IREQ for Card B | Note: Only one bit of this field should be set at any given time. |
| 8-15 | CBSCHLVL(0:7) | Define Interrupt Level for STSCHG for Card B | Note: Only one bit of this field should be set at any given time. |
| 16-17 | CBDREQ | Define pin to be used as internal DMA request to the on chip DMA controller (channel 1). | 0x = Disable internal DMA request from slot B.<br>10 = Enable $\overline{\text{IOIS16\_B}}$ as internal DMA request for slot B.<br>1 1 = Enable SPKR_B as internal DMA request for slot B. |
| 18-23 | Reserved | | |
| 24 | CBOE | Card B Output Enable | This bit is reflected on the OP(3) pin used to three-state the external buffers when the card power is activated. |
| 25 | CBRESET | Card A Reset | This bit is reflected on the OP(2) pin used to reset the card. |
| 26-31 | Reserved | | |

### Table 17-9. PCMCIA Base Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-31 | PBA(0:31) | **PCMCIA BASE ADDRESS**.<br>The base address register is compared to the address on the address bus to determine if a PCMCIA window is being accessed by an internal bus master. These bits are used in conjunction with the BSIZE field in the POR. | |

### Table 17-10. PCMCIA Option Register

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-4 | BSIZE | **PCMCIA BANK SIZE**.<br>This field determines the address mask field of each option register and it provides masking for any of the corresponding bits in the associated PCMCIA base register.<br>The Bank Size is calculated as BankSize = 2, where BSIZE represents the gray code shown to the right. | BSIZE BANK SIZE<br>00000 = 1 Byte<br>00001 = 2 Bytes<br>00011 = 4 Bytes<br>00010 = 8 Bytes<br>00110 = 16 Bytes<br>00111 = 32 Bytes<br>00101 = 64 Bytes<br>00100 = 128 Bytes<br>01100 = 256 Bytes<br>01101 = 512 Bytes<br>01111 = 1 KByte<br>01110 = 2 KBytes<br>01010 = 4 KBytes<br>01011 = 8 KBytes<br>01001 = 16 KBytes<br>01000 = 32 KBytes<br>11000 = 64 KBytes<br>11001 = 128 KBytes<br>11011 = 256 KBytes<br>11010 = 512 KBytes<br>11110 = 1 MByte<br>11111 = 2 MBytes<br>11101 = 4 MBytes<br>11100 = 8 MBytes<br>10100 = 16 MBytes<br>10101 = 32 MBytes<br>10111 = 64 MBytes |
| 5-11 | Reserved | | |

## Table 17-10. PCMCIA Option Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 12-15 | PSHT(0:3) | **PCMCIA STROBE HOLD TIME.** This attribute is used to determine when IOWR_x or WE_x are negated during a PCMCIA write access or when IORD_x or OE_x are negated during a PCMCIA read access handled by the PCMCIA interface. This is helpful to meet address/data hold time requirements for slow memories and peripherals. | 0000 = Strobe negation to address change 0 clock<br>0001 = Strobe negation to address change 1 clock<br>0010 = Strobe negation to address change 2 clock<br>0011 = Strobe negation to address change 3 clock<br>0100 = Strobe negation to address change 4 clock<br>0101 = Strobe negation to address change 5 clock<br>0110 = Strobe negation to address change 6 clock<br>0111 = Strobe negation to address change 7 clock<br>1000 = Strobe negation to address change 8 clock<br>1001 = Strobe negation to address change 9 clock<br>1010 = Strobe negation to address change 10 clock<br>1011 = Strobe negation to address change 11 clock<br>1100 = Strobe negation to address change 12 clock<br>1101 = Strobe negation to address change 13 clock<br>1110 = Strobe negation to address change 14 clock<br>1111 = Strobe negation to address change 15 clock |
| 16-19 | PSST(0:3) | **PCMCIA STROBE SET_UP TIME.** This attribute is used to determine when IOWR_x or WE_x are asserted during a PCMCIA write access or when IORD_x or OE_x are asserted during a PCMCIA read access handled by the PCMCIA interface. This is helpful to meet address/setup time requirements for slow memories and peripherals. | 0000 = Reserved<br>0001 = Address to strobe assertion 1 clock cycles<br>0010 = Address to strobe assertion 2 clock cycles<br>0011 = Address to strobe assertion 3 clock cycles<br>0100 = Address to strobe assertion 4 clock cycles<br>0101 = Address to strobe assertion 5 clock cycles<br>0110 = Address to strobe assertion 6 clock cycles<br>0111 = Address to strobe assertion 7 clock cycles<br>1000 = Address to strobe assertion 8 clock cycles<br>1001 = Address to strobe assertion 9 clock cycles<br>1010 = Address to strobe assertion 10 clock cycles<br>1011 = Address to strobe assertion 11 clock cycles<br>1100 = Address to strobe assertion 12 clock cycles<br>1101 = Address to strobe assertion 13 clock cycles<br>1110 = Address to strobe assertion 14 clock cycles<br>1111 = Address to strobe assertion 15 clock cycles |

## Table 17-10. PCMCIA Option Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 20-24 | PSL(0:4) | **PCMCIA STROBE LENGTH.**<br>This attribute 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 the length of the cycle.<br>The cycle may be lengthened by asserting the $\overline{\text{WAIT}}$ signal. | 00001 = Strobe asserted 1 clock cycles<br>00010 = Strobe asserted 2 clock cycles<br>00011 = Strobe asserted 3 clock cycles<br>00100 = Strobe asserted 4 clock cycles<br>00101 = Strobe asserted 5 clock cycles<br>00110 = Strobe asserted 6 clock cycles<br>00111 = Strobe asserted 7 clock cycles<br>01000 = Strobe asserted 8 clock cycles<br>01001 = Strobe asserted 9 clock cycles<br>01010 = Strobe asserted 10 clock cycles<br>01011 = Strobe asserted 11 clock cycles<br>01100 = Strobe asserted 12 clock cycles<br>01101 = Strobe asserted 13 clock cycles<br>01110 = Strobe asserted 14 clock cycles<br>01111 = Strobe asserted 15 clock cycles<br>10000 = Strobe asserted 16 clock cycles<br>10001 = Strobe asserted 1 7 clock cycles<br>10010 = Strobe asserted 18 clock cycles<br>10011 = Strobe asserted 19 clock cycles<br>10100 = Strobe asserted 20 clock cycles<br>10101 = Strobe asserted 21 clock cycles<br>10110 = Strobe asserted 22 clock cycles<br>10111 = Strobe asserted 23 clock cycles<br>11000 = Strobe asserted 24 clock cycles<br>11001 = Strobe asserted 25 clock cycles<br>11010 = Strobe asserted 26 clock cycles<br>11011 = Strobe asserted 27 clock cycles<br>11100 = Strobe asserted 28 clock cycles<br>11101 = Strobe asserted 29 clock cycles<br>11110 = Strobe asserted 30 clock cycles<br>11111 = Strobe asserted 31 clock cycles<br>00000 = Strobe asserted 32 clock cycles |
| 25 | PPS | **PCMCIA PORT SIZE.**<br>This field specifies the port size of this PCMCIA window. | 0 = 8 bits port size<br>1 = 16 bits port size |

### Table 17-10. PCMCIA Option Register (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 26-28 | PRS | **PCMCIA REGION SELECT**. | 000 = Common Memory Space<br>001 = Reserved<br>010 = Attribute Memory Space<br>011 = I/O Space<br>100 = DMA (Normal DMA Transfer)<br>101 = DMA Last Transaction<br>110 = Reserved<br>111 = Reserved<br>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.<br>Note: 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.<br>1 = This window defined for slot B. |
| 30 | WP | This window is write-protected. Any write attempt to this window results in a bus error (machine_check interrupt). | |
| 31 | PV | **PCMCIA VALID BIT**.<br>This bit indicates that the contents of the PCMCIA base register and option register pair are valid. | 0 = This bank is invalid.<br>1 = This bank is valid. |

## 17.6 PCMCIA CONTROLLER TIMING EXAMPLES

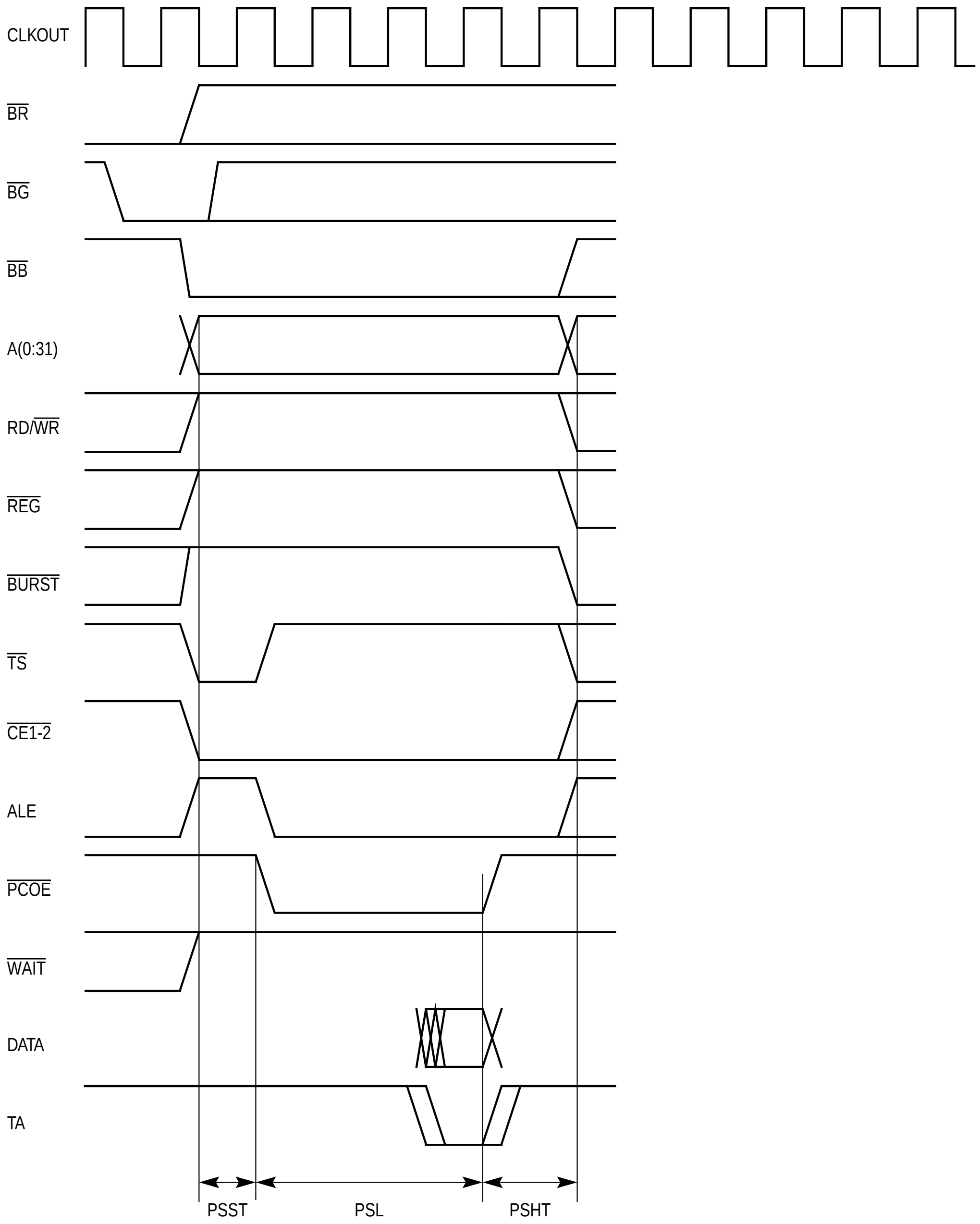


**Figure 17-3. PCMCIA Single Beat Read Cycle PRS = 0 PSST = 1 PSL = 3 PSHT = 1**

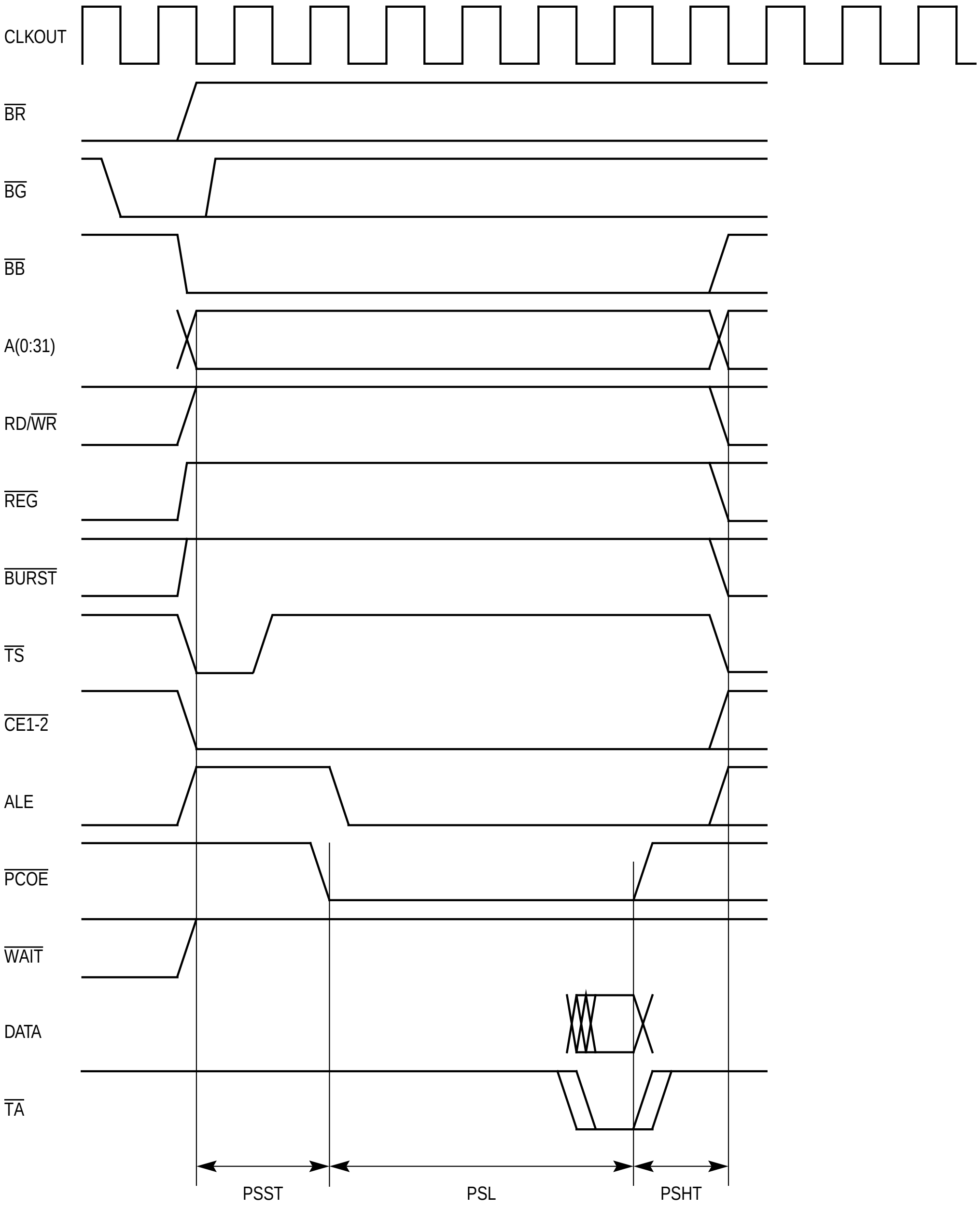


**Figure 17-4. PCMCIA Single Beat Read Cycle PRS = 0 PSST = 2 PSL = 4 PSHT = 1**

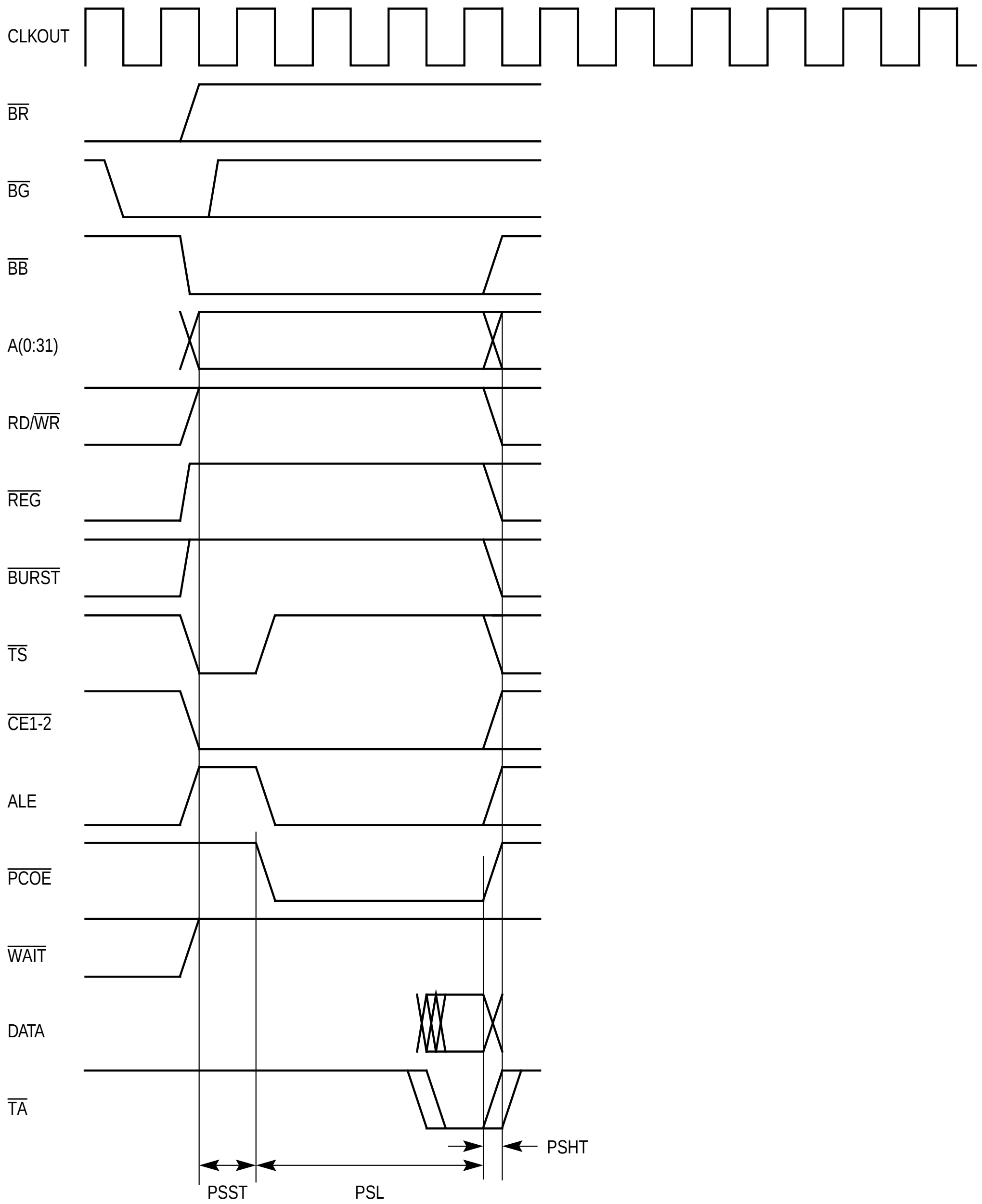


**Figure 17-5. PCMCIA Single Beat Read Cycle PRS = 0 PSST = 1 PSL = 3 PSHT = 0**

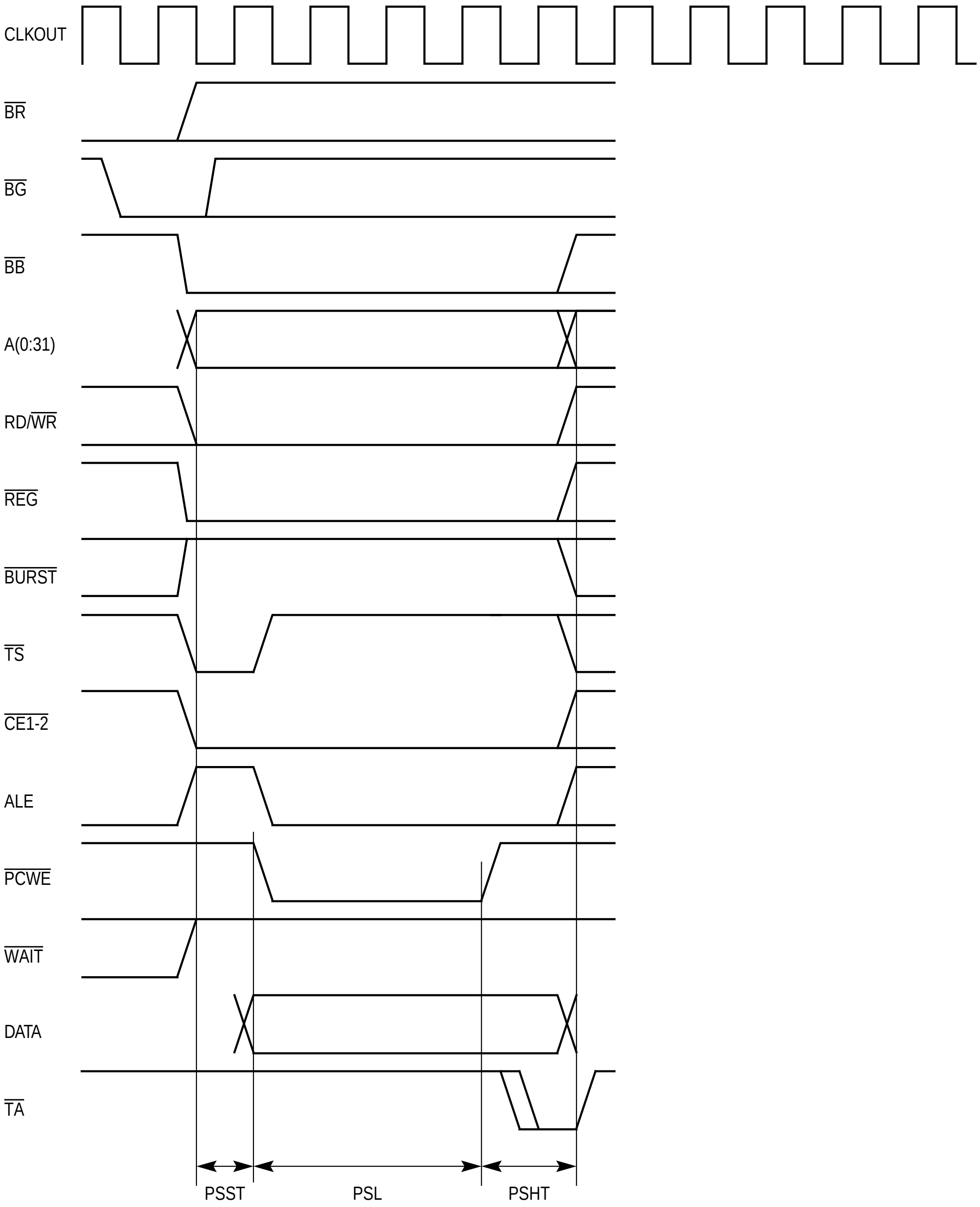


**Figure 17-6. PCMCIA Single Beat Write Cycle PRS = 2 PSST = 1 PSL = 3 PSHT = 1**

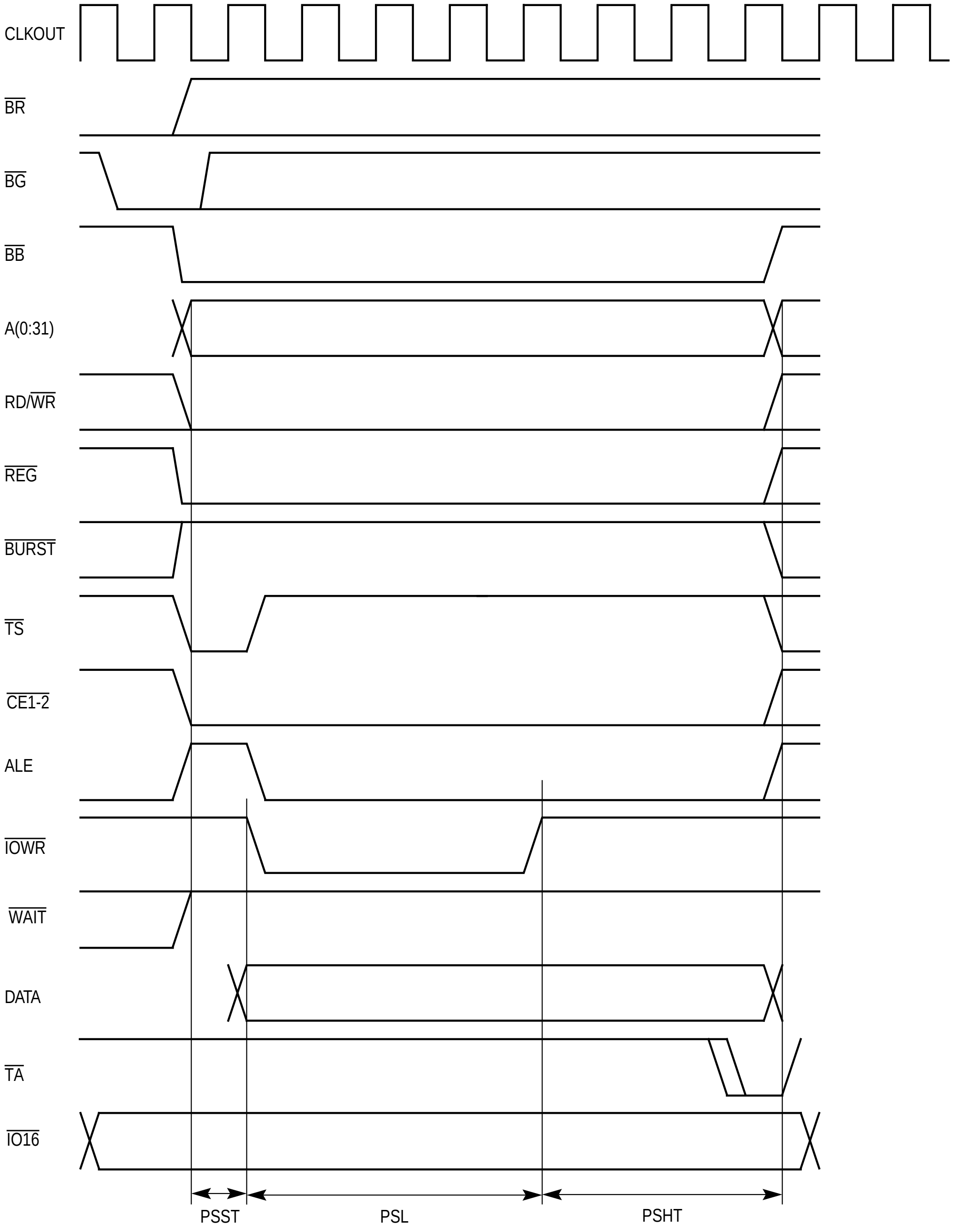


Figure 17-7. PCMCIA Single Beat Write Cycle PRS = 3 PSST = 1 PSL = 4 PSHT = 3

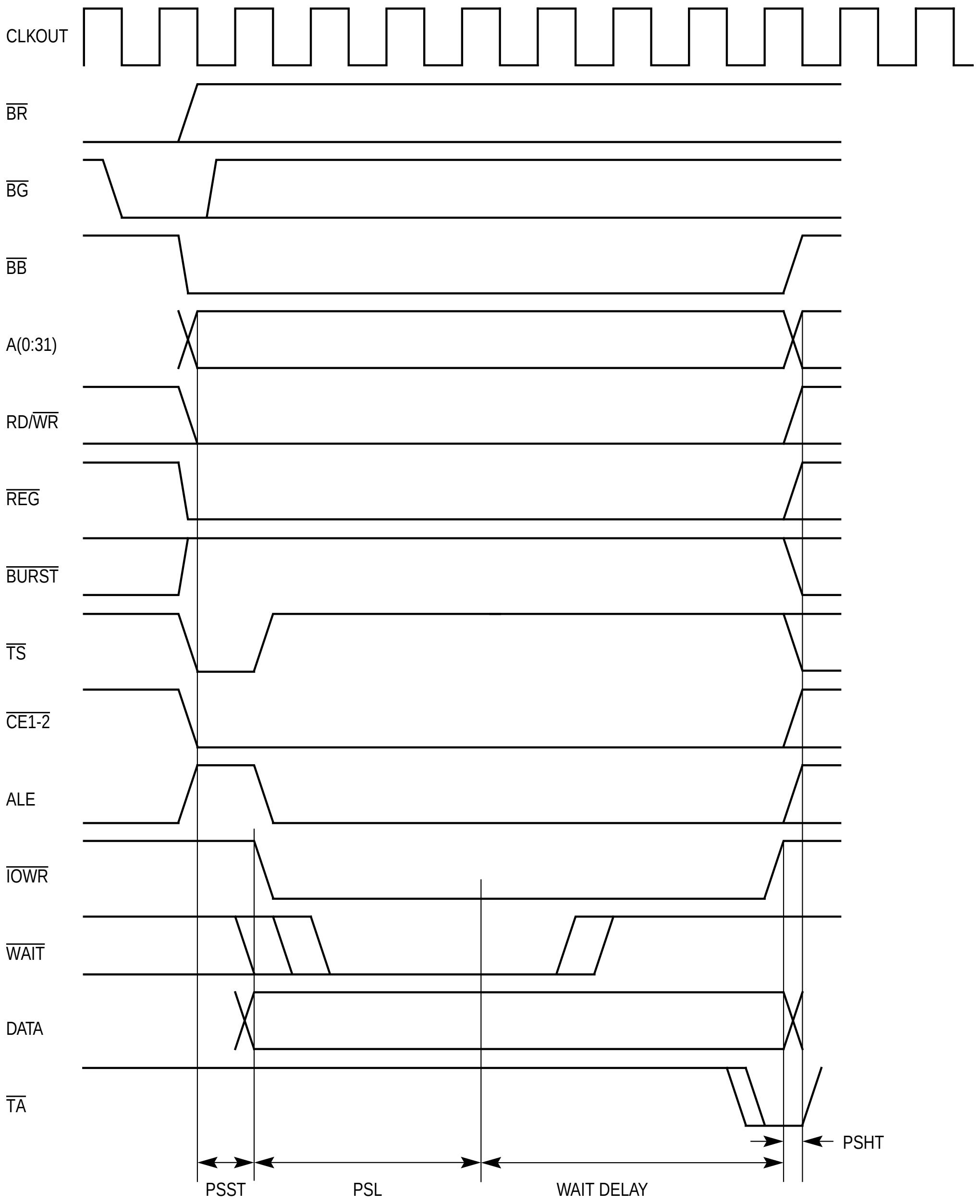


**Figure 17-8. PCMCIA Single Beat Write With Wait PRS = 3 PSST = 1 PSL = 3 PSHT = 0**

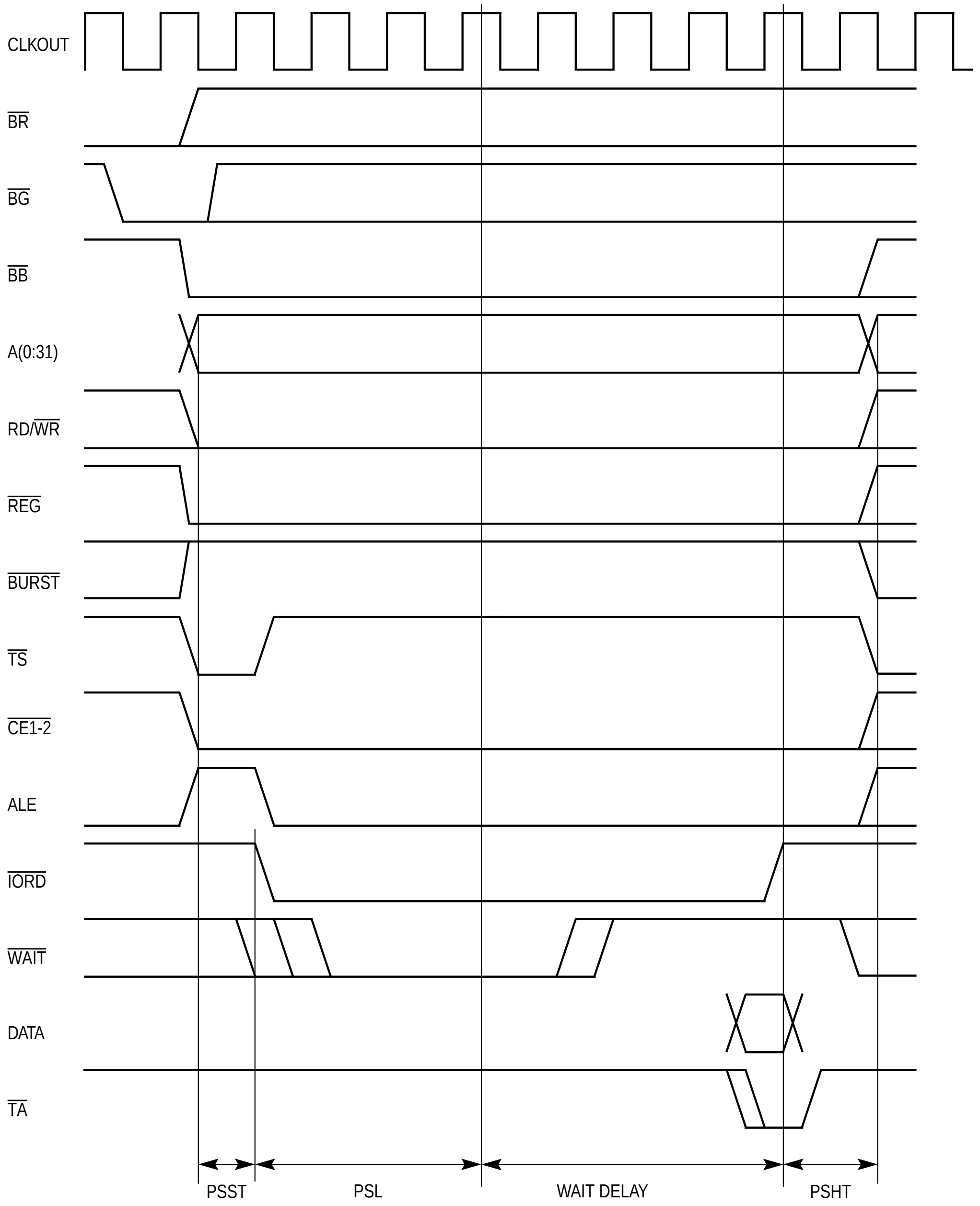


**Figure 17-9. PCMCIA Single Beat Read With Wait PRS = 3 PSST = 1 PSL = 3 PSHT =1**

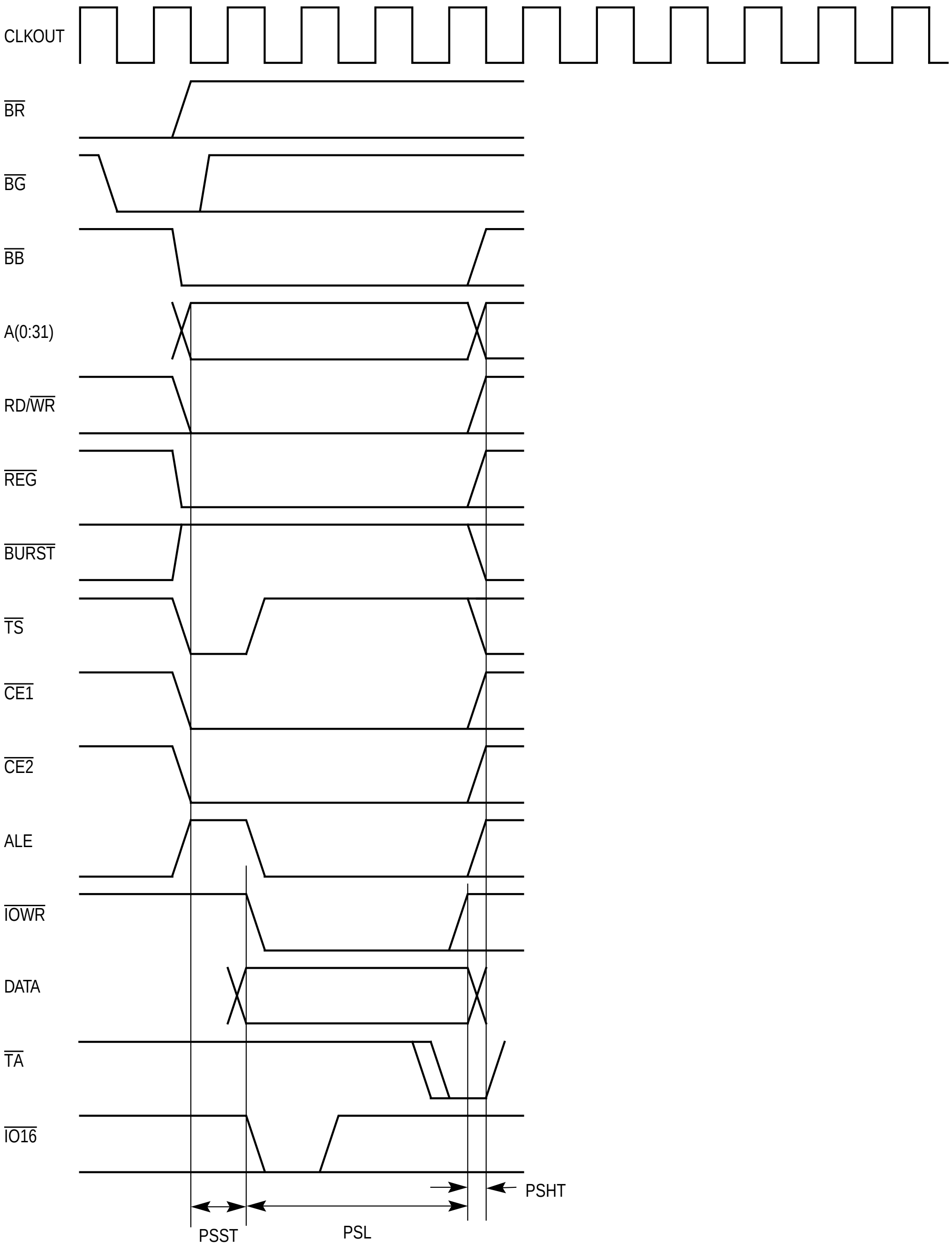


**Figure 17-10. PCMCIA I/O Read PPS = 1 PRS = 3 PSST = 1 PSL = 2 PSHT = 0**

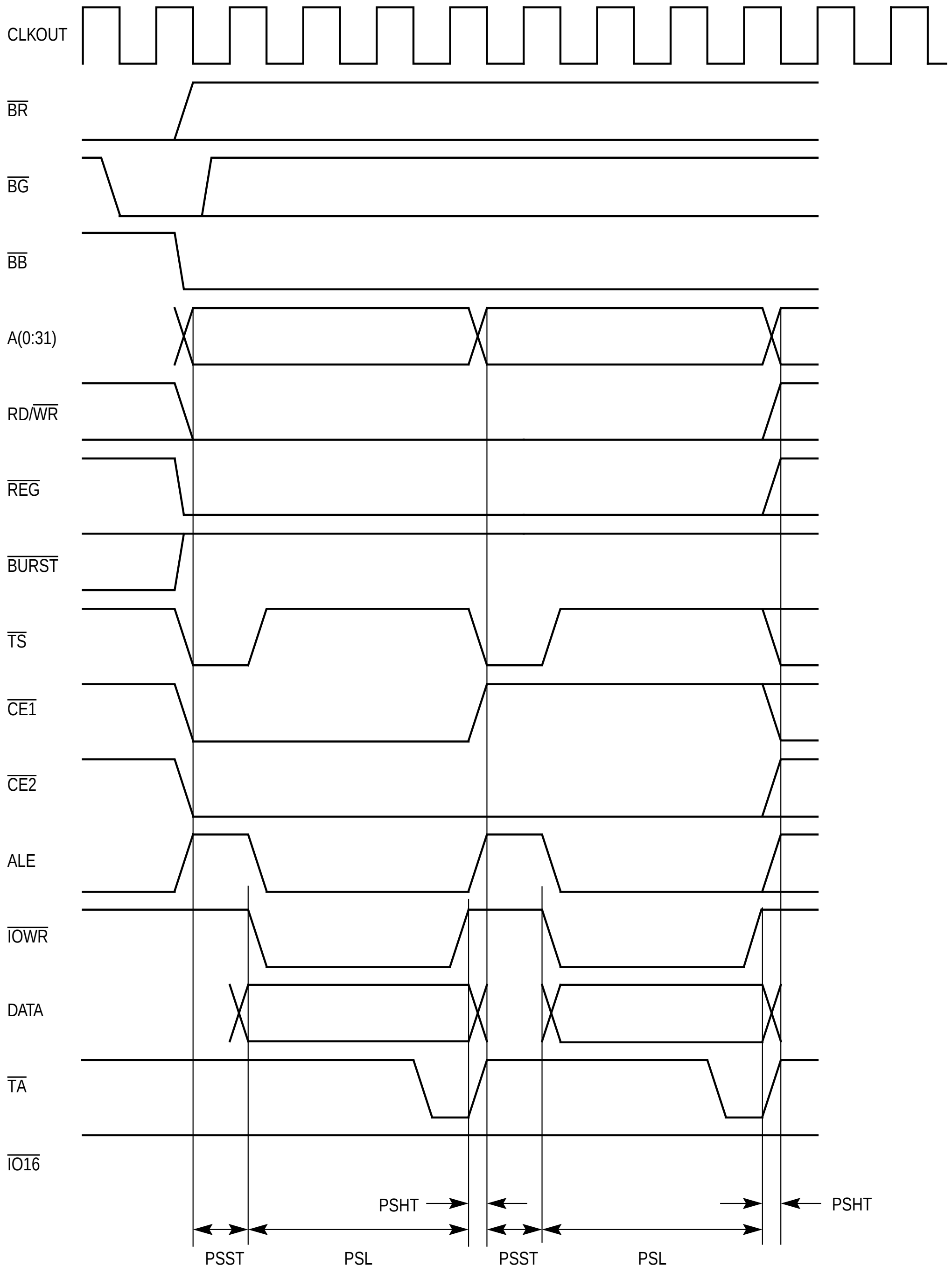


**Figure 17-11. PCMCIA I/O Read PPS = 1 PRS = 3 PSST = 1 PSL = 2 PSHT = 0**

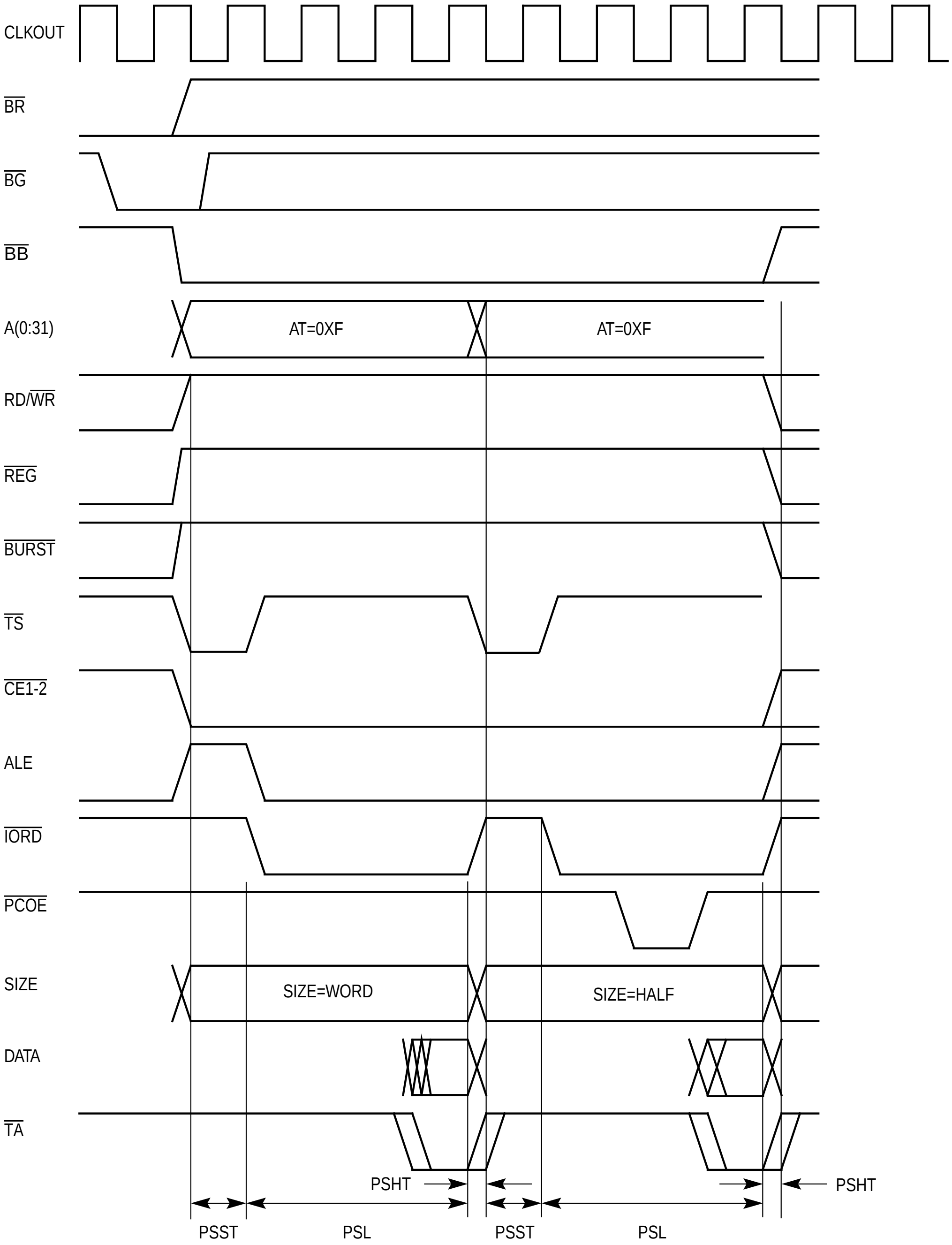


**Figure 17-12. PCMCIA DMA Read Cycle PRS = 4 PSST = 1 PSL = 3 PSHT = 0**

# SECTION 18
# LCD CONTROLLER

## 18.1 BACKGROUND

### 18.1.1 LCD Technology

A liquid crystal display (LCD) is a passive display technology that uses ambient light in the environment to display images with very little power. They consist of two pieces of glass with electrodes printed on the inside and polarizers are used on the outside front and rear surfaces. When the LCD is off, no voltage is applied to the electrodes and light passes through the LCD. When it is on, voltage is applied to the electrodes and the liquid crystal molecules align themselves in the direction of the electric field. This causes the light to be blocked and out of phase with the polarizers, creating a dark area on the LCD. The darkness of LCD pixels is a function of the RMS voltage applied to them. This means that polarity of the applied voltage is not important and that no DC bias is allowed. A typical black and white LCD module is illustrated in Figure 18-1 below.

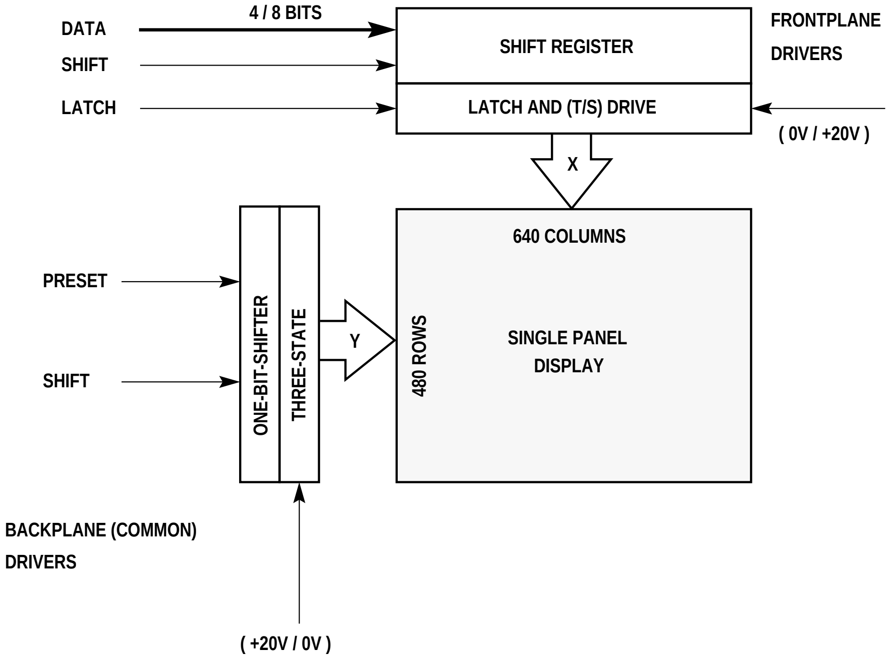


Figure 18-1. LCD Panel

The Y drivers apply high voltage to one row at a time. At the same time, the X drivers connect the dark pixels to the opposing voltage and bright pixels to the same voltage as the row. The value of each pixel is shifted by the shift register and latched when the whole row is ready. While one row is being refreshed, the next row is being shifted in and each row is refreshed at a rate inverse to the number of rows. If the number of rows is large, then the refresh rate might become too small. When this occurs, the panel is divided into 2 sections, usually upper and lower. Then the X shift register, which is twice as long, works at twice the speed to refresh two columns at the same time, one column in the upper section and one in the lower section. The X shift register can be loaded 1, 2, 4 or 8 bits per shift.

The time it takes to refresh a row is called the duty-cycle. In a simple refresh policy, one row is refreshed at a time. However, this creates large voltage spikes on the LCD electrodes. Another technique, called active addressing, refreshes groups of rows and smoothes the applied voltage that each pixel receives over approximately half a frame cycle. This significantly improves contrast and other characteristics of a simple LCD display. The drawback of active addressing is that it is fairly complex and needs a dedicated chip to perform. This chip holds a large display buffer (typically equivalent to a quarter of a VGA display per circuit). The active addressing algorithm is done on entire columns and it usually fits in the LCD display path and appears to the system as a thin film technology (TFT) interface.

It is important to avoid long-term DC bias in the voltages applied to the LCD screen because it causes defects in the polarization as well as raindrop-like patterns (raindrop effect) to appear on the display. To overcome this problem the controller should change the field polarity every few frame times.

### 18.1.2 Basic LCD Interface

Simple LCD panels with up to several hundred individual pixels or pictograms (large pixels with shape and meaning) have no electronics on them at all. There are 30-40 front panel inputs and several (1-4, or more) back panel inputs and when combined, these inputs charge the pixels at their intersection. Motorola microcontroller devices like the MC68HC0L5 provide such an LCD interface. An example of a complete LCD subsystem is illustrated in Figure 18-2.

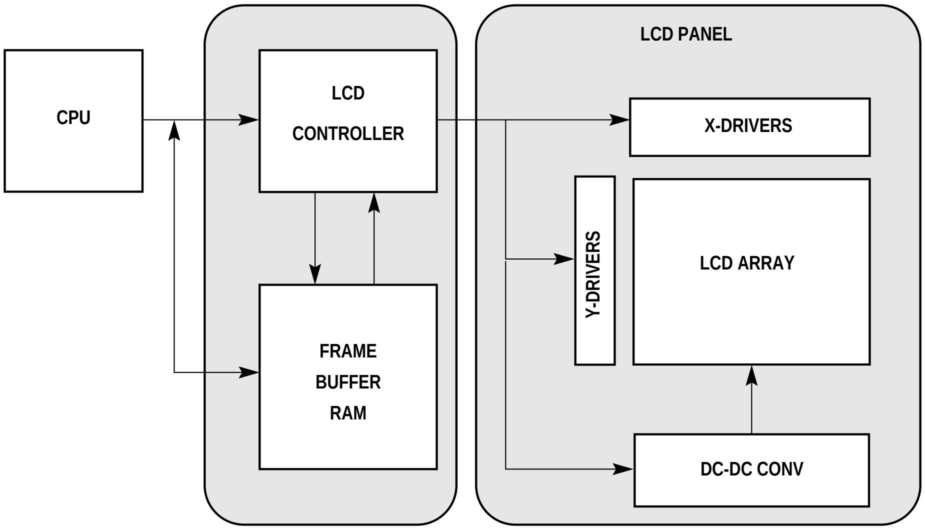


Figure 18-2. LCD Subsystem

**18.1.2.1 STANDARD LCD INTERFACE (PASSIVE).** The LCD panel contains a master/slave shift register. The slave is used for displaying and the master is filled from the outside. The LCD controller fills the shift register, provides framing and master-to-slave write signals, and reverses the display polarity from frame to frame.

The standard LCD interface consists of several parallel data bits (1-8) that are shifted by shift clock into an X shift register. After the shift register is full, a latch signal transfers the pixels from the shift register to driver latches and moves the Y pointer down one line. At this point, the shift operation continues to the next line and after all the lines are scanned, the frame signal moves the Y pointer to the beginning of the frame. The LCD controller also accesses the frame buffer. Addressing can be split or nonsplit. The interpretation of the read-in data can differ dependent upon the display modes.

**18.1.2.2 TFT INTERFACE (ACTIVE).** The thin film transistor (TFT) interface for high-performance LCD panels looks more like a digital RGB or black and white video signal and has several data bits in parallel. The shift clock is also present. Latch and frame signals are called horizontal sync and vertical sync and have special timing. There is also a special signal called output enable that blanks/enables data, but does not affect the clock. For color displays, 9-bit (3 bits per basic color) is supported. An 8-bit (4-bit) pixel data fetched from the frame buffer is passed through a 256 x 12 memory that selects one out of 256 (16) colors.

**18.1.2.3 SMART PANEL INTERFACE.** In smart panel interface, the whole memory buffer is on the panel, which is memory-mapped and connected to the system bus. The CPU supports the panel, so no controller is needed. With this interface, there are two more common controls for the LCD panels. The sleep mode (or panel_on signal) control is used for power saving and is activated by a command from the CPU to the LPC. Shut-off control is inverted LPC-enable. Also, for conventional LCD panels, control is needed to kill any DC biases which are built up during normal operation. LCD_AC is the required signal that inverts the polarity of voltages presented to the LCD. Usually, this signal toggles every several frames (1-20).

## 18.2 LCD SYSTEM OVERVIEW

The MPC821 LCD system is illustrated in Figure 18-3 below.

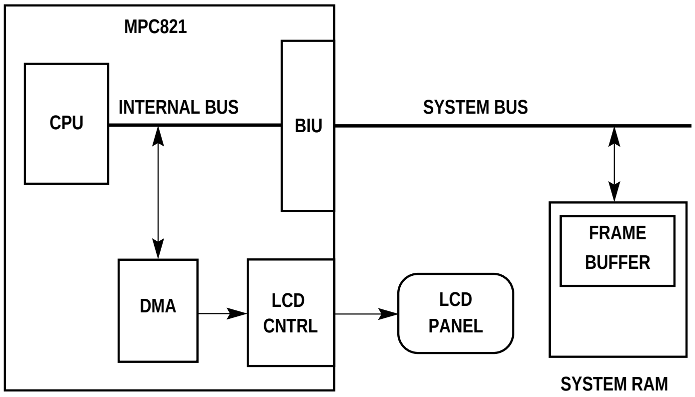


**Figure 18-3. MPC821 LCD System**

The LCD panel controller (LPC) is initialized by the CPU first. It gets the frame buffer address, the operational modes and various configuration bits. After being enabled, the LPC requests the DMA to fetch the frame buffer data. The frame buffer is always organized as an orthogonal matrix; rows and columns. The data fetched from the buffer undergoes interpretation for grayness/color and frame format for a specific LCD interface. The data is packed according to the gray model chosen. If a split display panel is used, two buffers are needed. Therefore, two buffer pointers must be initialized. The LPC uses continuous DMA to feed the display. The most power-intensive configuration supported is full VGA with 4 bits per pixel.

In this example, the display characteristics would be:

- 640 × 480 = 307,200 pixels/screen
- (307,200 pixels/screen)* 4 bits/pixel = 1.228 megabits/screen = 150 kilobytes/screen
- 70 Hz * 150 kilobytes = 10.5 megabytes/second
- 70 Hz is 14.3 milliseconds per frame: 14.3 milliseconds/(307,200 pixels clocked 4 bits at a time) = 186 nanoseconds (approximately 5 MHz) serial clock to the LCD drivers

The display serial clock is actually slightly faster than 5 MHz because of the panel overhead. In most VGA displays, however, an 8-bit LCD serial data (split screen) is used and the display frequency is 3.76 MHz. An assumption about the system configuration is that the parallel to serial clock ratio should be between 4:1 and 5:1 to guarantee bus access during system activity.

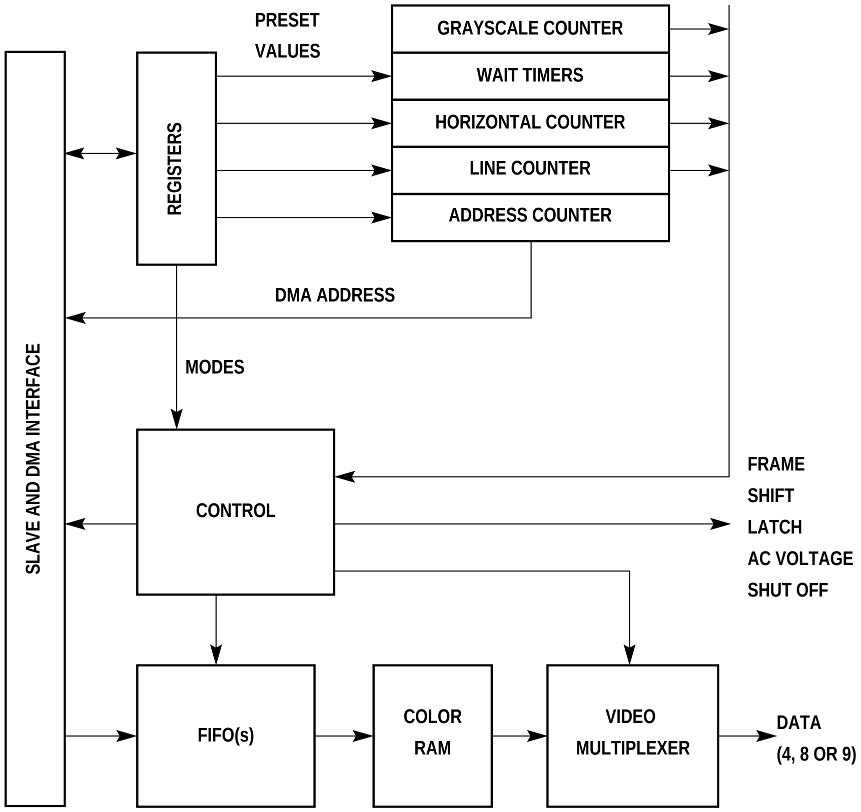


**Figure 18-4. LCD Controller General Block Diagram**

## 18.3 FEATURES

The following is a list of the LCD controller’s important features

- Passive and active panels are supported
- 1, 2, or 4 bits per pixel gray mode generates up to 16 gray levels using an advanced FRC algorithm
- 4 or 8 bits per pixel color mode generates 16 or 256 simultaneous colors out of 4,096 (512 for TFT)
- Interfaces to the LCD panel through 4-, 8-, or 9-bit wide parallel output bus
- Supports nonsplit or vertically-split screens
- 2+2 or 4+4 split display data modes (2+2 means 2 bits for low screen and 2 bits for high screen in parallel)
- Built-in 256 entry color RAM:
  - — Maps each 4- or 8-bit color code to one of 4,096 (512 for TFT) colors. Each color is defined by a 4-bit code for red, green and blue.
  - — Maps each 2- or 4 bit gray level code to one of 16 gray levels.
- Programmable wait time between lines and frames
- Panel voltage control–programmable LVDAC through duty-cycle, for contrast adjustments. This is implemented by using an existing on-chip timer.
- Programmable polarity for all LCD interface signals
- TFT/RGB output drives advanced buffer LCD driver chips
- Uses burst read DMA cycles for maximum bus performance

## 18.4 LPC ARCHITECTURE

A detailed block diagram of the LPC architecture is illustrated in Figure 18-5.

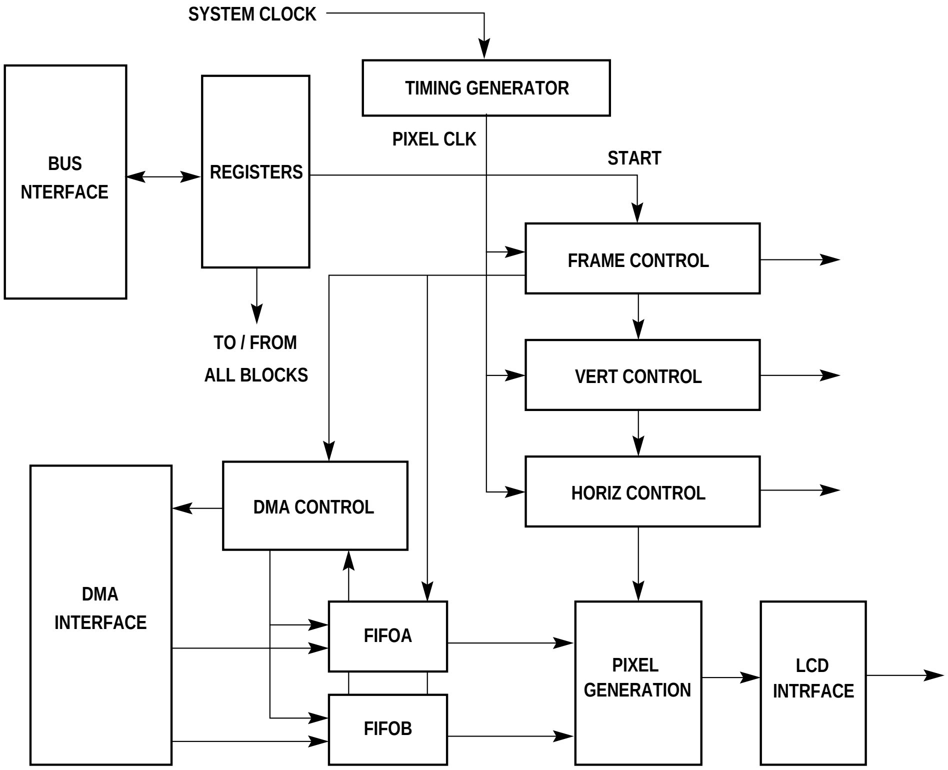


**Figure 18-5. Detailed Block Diagram**

After reset, the LPC is idle because the LPC-ON bit in the configuration register is cleared at reset. The host should program all relevant slave registers and then write a 1 to the LPC-ON bit. When enabled, the FIFOs generate the FIFO_Hungry signals and the DMA control starts filling the FIFOs by initiating burst read cycles. When the FIFOs are sufficiently full, the LPC state machine starts working. Data is formatted and shifted out by the horizontal control. The horizontal machine works by presetting the line pixel counter. Each pixel time the counter is decremented. After reaching count zero, additional waits can be made using the wait-after-line register value. After one horizontal line is shifted out, the vertical control updates the outputs and after all the lines are shifted out, the cycle repeats. The lines counter works the same as the pixel counter (operating on lines) and is decremented each line time.

### 18.4.1 FIFO

There are two FIFOs in the LPC that are concatenated for nonsplit displays and used separately for split displays. The capacity of each FIFO is 12 words of 32 bits.When the FIFO is capable of accepting a burst (minimum of 4 word vacancy) it asserts the FIFO_Hungry signal. This triggers the DMA control. Both FIFOs are then written with bursts of 32-bit words, so there is no chance of overflowing. After the FIFOs are sufficiently filled, the LPC asserts the FIFO_Ready signal for the frame control so that it can start processing a frame.

If, during the frame, a FIFO underruns, there is no mechanism for recovery and the display can suffer. The operating system and it’s design should ensure that underruns do not occur during normal operation. The READ mechanism for the FIFOs is more complex since the FIFO can be read in increments of 2, 4, 8 or 16 bits and there is a special mechanism to keep track of valid data.

### 18.4.2 LPC Pixel Generation

The LPC pixel generation block transfers data from the FIFOs through gray/color modifications to the output interface. This block fills the correct amount of data to the output latch, does the reads from the correct FIFOs with the correct size. The gray function and COLOR_RAM are parts of this block. In the event of a split display where data is output in parallel to both subpanels, the half-time pixel generator reads the first half-time from FIFO A and the second half-time from FIFO B.

**18.4.2.1 GRAYSCALE GENERATION.** For black and white display, each pixel is represented by a single bit in the frame buffer. A zero corresponds to the pixel off and a one corresponds to the pixel on. The LPC supports generation of up to 16 gray levels. In this case, each pixel is represented by 2 bits (4 shades) or 4 bits (16 shades). The grayscale is generated by controlling the number of frames (out of 16 consecutive frames) for which the pixel is on. This method is referred to as the frame rate control (FRC). To reduce flickering, pixels with the same gray level are turned on and off at different times, but they keep the same duty-cycle.

When using 2 bits per pixel, the color RAM is used to map the 2-bit code to one of 16 gray levels, but with 4 bits per pixel, 4-bit code is mapped to one of 16 gray levels. Refer to Figure 18-6 for details.

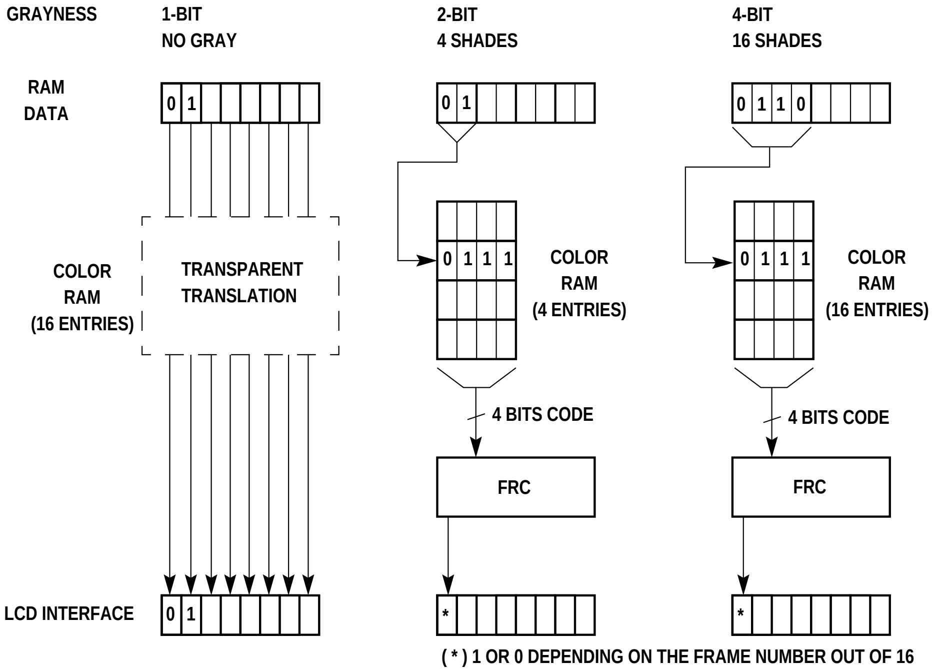


**Figure 18-6. Grayscale Generation**

**18.4.2.2 COLOR GENERATION.** Each color pixel is represented as a 4- or 8-bit code in the frame buffer. Using the color RAM, the pixel code is mapped to 12-bit RGB code, allowing selection of one of 4,096 colors. For standard (passive) color display, the FRC algorithm processes each color to generate the required amount of intensity. For TFT (active) displays, 9 out of the 12 bits (3 bits per color) are output directly onto the LCD data bus. Refer to Figure 18-7 for more information.

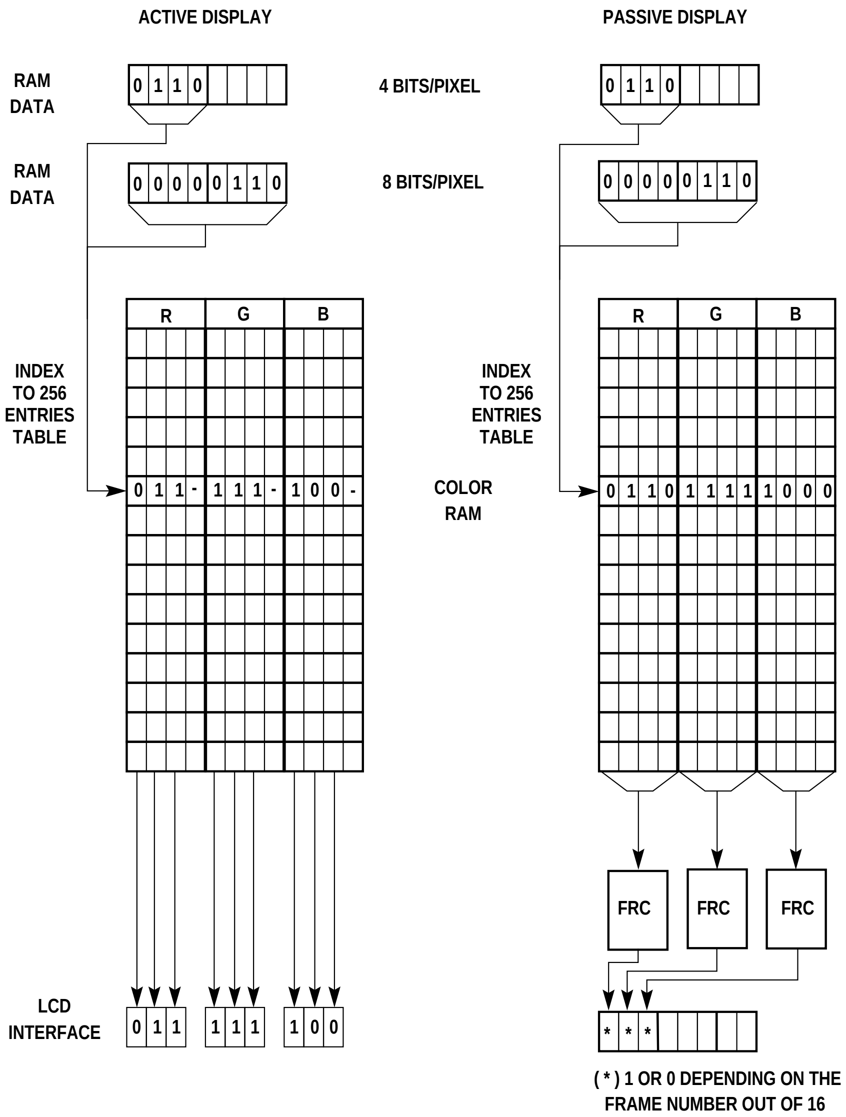


**Figure 18-7. Color Generation**

### 18.4.3 Horizontal Control

The horizontal control block counts the correct pixel count for one line and any additional wait time after the line end. It also enables the pixel generation block and by sending a signal from the vertical control to each line. It then signals the vertical control function.

### 18.4.4 Vertical Control

The vertical control block counts the lines and activates the horizontal control. After all lines are scanned, it waits the time specified in the wait_after_frame bit field of the horizontal control register. On completion, it signals the frame control and turns inactive.

### 18.4.5 Frame Control

The frame control block initializes all counters and fires the DMA and vertical control blocks. It can also generate frame interrupts.

### 18.4.6 DMA Control

The DMA control block, when enabled, checks the FIFO_Hungry signals against FIFO enabled and decides which FIFO to fill. For each FIFO that is hungry, the address latch is loaded with the buffer_pointer, the buffer pointer is incremented by 16 bytes, then DMA is initiated. DMA control waits until four writes are completed and then it is ready to service another request. The frame buffer address should be aligned to16 bytes, so the four least-significant address bits are always zero. When the LCD controller is started at reset or after underrun condition, it starts video data processing once the FIFOs are filled by five DMA read bursts (20 longwords).

### 18.4.7 Slave Interface

The slave interface addresses 32-bit internal configuration registers of the LPC. The LPC asserts one frame interrupt, which is asserted when a frame is finished. This interrupt is enabled through a bit in the configuration register.

### 18.4.8 LCD Interface

The LCD interface module drives the data and handshake signals to the LCD display. The polarity of all interface signals is user programmable. The LCD interface can be configured to support all major LCD panels configurations—split/nonsplit, color/black and white, and passive/TFT.

**18.4.8.1 SINGLE/SPLIT LCD PANELS.** Some LCD panels split the display area into two independent vertical regions that are scanned together. Thus, two lines are shifted and displayed simultaneously. In this mode, half of the data bus is used to drive the upper half of the screen and the other half is used to drive the lower half of the screen. Refer to Figure 18-8 for an illustration of the single and split LCD panels.

**18.4.8.2 STANDARD INTERFACE.** A standard LCD interface uses the following signals whose polarity is programmable.

- Data—The width of this bus is configured from 1 to 9 bits in several combinations.
- Frame—This signal starts the frame by putting the Y pointer to the first row.

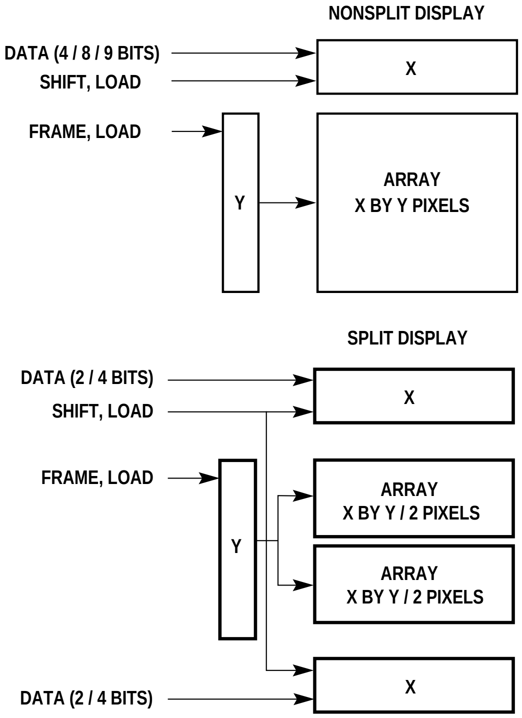


**Figure 18-8. Single and Split LCD Panels**

### SHIFT

On the positive edge of the shift signal, data is latched into the X shift register.

- Load—This signal moves the contents of the shift register into the drive latches.
- LCD_AC—This signal toggles every few frames. It is used by LCD panels to nullify any DC voltage residues. The toggle rate is programmable.
- Panel_On—This output is a direct value of the panel_on configuration bit. Internally, it starts operation and instructs the panel to become active. In the inactive state, it saves power by disabling the panel and controller.
- Analog_Out—This is a low frequency digital-to-analog output that is used to control the contrast voltage provided to the LCD panel. The analog output is generated by integration on a PWM waveform the duty-cycle of the wave controls the voltage. The PWM wave is generated by one of the CPM timers. Refer to **Section 16.7.3 PWM Mode** for more details.

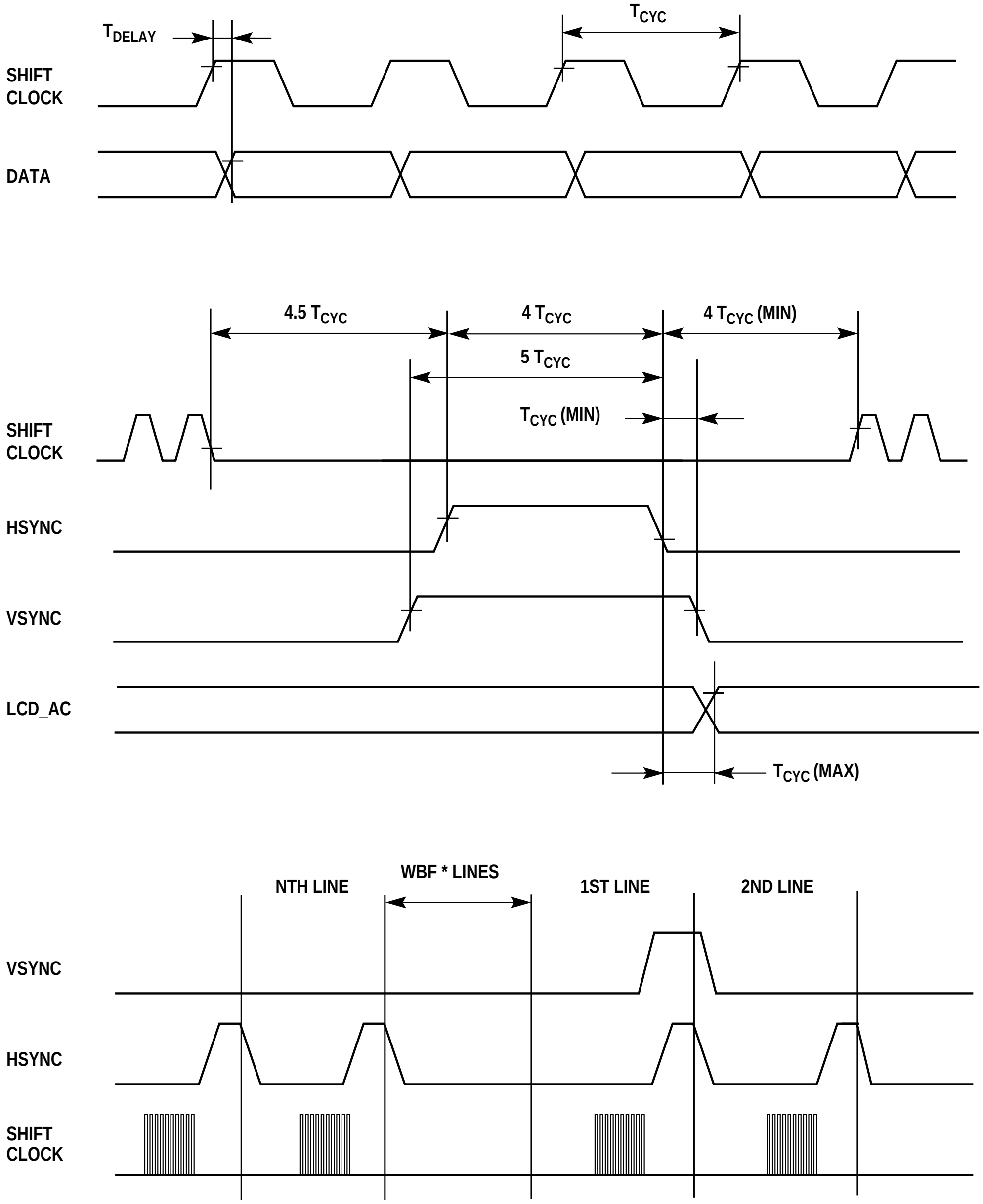


**Figure 18-9. Passive Display Interface Timing Diagram**

**18.4.8.3  DIGITAL CRT INTERFACE.** A digital CRT (TFT) interface uses the following signals whose polarity is programmable.

- $\overline{\text{OE}}$—The output data valid signal is similar to the blanking output in a CRT and (when valid) enables the data to be shifted into the display. When disabled, the data is invalid and the trace is off.
- Clock—Data clock. When OE is valid, data is latched on the negative edge of this clock.
- Hsync—Horizontal sync causes the panel to start a new line. This is functionally similar to the LPC load signal.
- Vsync—Vertical sync causes the panel to start a new frame. This is functionally similar to the LPC frame signal.
- Data—4-, 8-, or 9-bit data. For black and white displays, 4- or 8-bit data is the same for standard interfaces and 9-bit data is used for color displays. Refer to **Section 18.5 Programming Model** for definitions of terms. Listed below are some formulas that you can use to calculate the hsync and vsync cycles from.

```
vsync_cycle = hsync_cycle * (L (+WBF)
hsync_cycle = shift_clk_cycle * (P/LBW +12 (WBL))
shift_clk_cycle = Lcdclk_cycle * F
Lcdclk_cycle = CLK_cycle * LCD_div_factor (programmed)
```

Where:

L = Number of lines in panel (/2 if split display, +VPW (in LCVCR) if tft).

WBF = Number of wait between frames.

P = Number of pixels per line in panel (*2 if split display).

LBW = LCD bus width (4 or 8 for passive, 1 for tft).

WBL = number of wait between lines.

CLK_cycle = clkout_cycle in “normal high” mode (the mode after reset).

LCD_div_factor = LCD div factor is programmed in SCCR (dflcd * dfalcd).

F= inner rate factor that depends on the configuration:

3: For 4 bits per pixel color passive display with 8-bit LCD data bus width no split.

2: For 4 bits per pixel color passive display with 8-bit LCD data bus width split or with 4-bit LCD data bus width no split.

1: All other configurations.

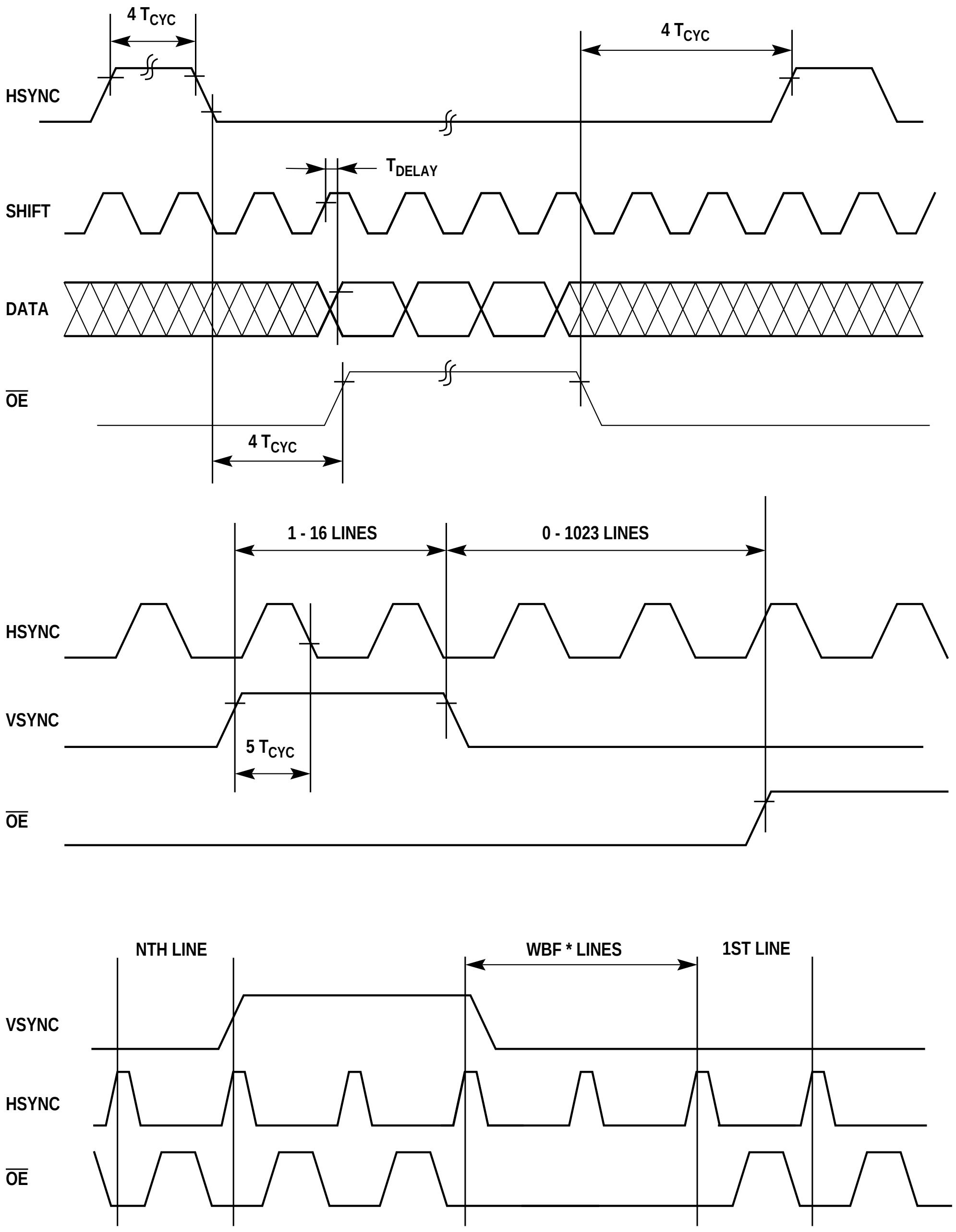


**Figure 18-10. Digital CRT Display Interface Timing Diagram**

### 18.4.9 LPC Clock

The LPC module is clocked by the LCDCLK that is generated by the system interface unit by dividing the system clock. It is used for converting the frame data to pixel format. The division factor is dependent on the type of display used. LCDCLK should be programmed according to the following table (F = LCD panel clock frequency).

| DATA DECODING | LCD DATA BUS | | | | |
|---|---|---|---|---|---|
| | 8 BITS/<br>NO SPLIT | 8 BITS/<br>SPLIT | 4 BITS/<br>NO SPLIT | 4 BITS/<br>SPLIT | 9 BITS/<br>NO SPLIT |
| 1-Bit Mono | F | F | F | F | — |
| 2-Bit Gray | 2 * F | 2 * F | F | F | — |
| 4-Bit Gray | 2 * F | 2 * F | F | F | — |
| 4-Bit Color Passive | 3 * F | 2 * F | 2 * F | — | — |
| 4-Bit Color TFT | — | — | — | — | F |

## 18.5 PROGRAMMING MODEL

### 18.5.1 LCD Panel Configuration Register

The LCD panel configuration register (LCCR) is a 32-bit, memory mapped, read/write register that holds the mode and configuration bits for the LCD panel. It is cleared on hard reset.

LCCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | BNUM | | | | | | | | | | | | | | – | – |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| ADDR | 840 | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | IEN | IRQL | | | CLKP | OEP | HSP | VSP | DP | BPIX | | LBW | SPLT | CLOR | TFT | PON |
| RESET | | | | | | | | | | | | | | | | |
| 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 | 840 | | | | | | | | | | | | | | | |

BNUM—Number of Bursts

This bit field contains the number of burst cycles required for one refresh cycle.

IEN—Interrupt Enable

0 = The end of frame interrupt is disabled.
1 = The end of frame interrupt is enabled.

IRQL—Interrupt Request Level

The IRQL field contains the priority request level of the LCD controller's interrupt being sent to the CPU.

CLKP—Clock Polarity

0 = The panel clock polarity is active high.
1 = The panel clock polarity is active low.

OEP—Output Enable Polarity

0 = The panel output enable signal polarity is active high.
1 = The panel output enable signal polarity is active low.

HSP—Horizontal Sync Polarity

0 = The panel horizontal sync signal polarity is active high.
1 = The panel horizontal sync signal polarity is active low.

VSP—Vertical Sync Polarity

0 = The panel vertical sync signal polarity is active high.
1 = The panel vertical sync signal polarity is active low.

DP—Data Polarity

0 = The panel data polarity is active high.
1 = The panel data polarity is active low.

BPIX—Bits Per Pixel

This bit field indicates how many bits in memory represent one pixel.

00 = One bit per pixel.
01 = Two bits per pixel.
10 = Four bits per pixel.
11 = Eight bits per pixel.

LBW—LCD Bus Width

This bit field indicates how many data bits are output on the same shift clock. It is only valid for monochrome displays. For color TFT display 9 bits are output.

0 = Four bits per clock.
1 = Eight bits per clock.

SPLT—Split Display Mode

0 = The display is not split (one row is displayed at a time).
1 = The display is in split mode (two rows are displayed at a time).

CLOR—Color Display

0 = The LCD panel is a monochrome display.
1 = The LCD panel is a color display.

TFT—TFT Display

0 = The LCD panel is a passive display.
1 = The LCD panel is an active TFT display.

PON—Panel On

0 = The LCD controller operation is disabled.
1 = The LCD controller operation is enabled.

### Example

A formula to calculate the bus performance while running a 256 color LCD panel is as follows:

SCLK = System clock frequency

FRM = Frame refresh rate

BIT = Bits per pixel

COL = Pixels per line

ROW = Lines per frame

MB = Number of system clocks per memory burst

BUNUM = Number of bursts per frame

BUNUM = (COL x ROW x BIT) / 128

Bus Band Width = (BUNUM x FRM x MB) / SCLK

For example, if the system clock is 33 MHz, one burst takes five clocks (2, 1, 1, 1 SRAM TYPE), LCD is full VGA, and the refresh rate is 80 MHz.

BUNUM = 640 x 480 x 8 / 128 = 19200

Bus Band Width = 19200 x 80 x 5 / 33000000 = 23%.

### 18.5.2 LCD Horizontal Control Register

The LCD horizontal control register (LCHCR) is a 32-bit, memory-mapped, read/write register that holds the panel horizontal size and other configuration bits. It is cleared on hard reset.

LCHCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | — | — | — | BO | AT[1:3] | | | HPC | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 844 | | | | | | | | | | | | | | | |
| **BITS** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| FIELD | HPC | | | | | | WBL | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 844 | | | | | | | | | | | | | | | |

BO—Byte Order

0 = PPC little endian byte order.
1 = Big/little endian byte order.

AT—Address Type[1:3]

The AT field contains values that the user wants to appear on the address type pins AT1–AT3 when the associated SDMA channel accesses memory. AT0 is always driven to 1.

HPC—Horizontal Pixel Count

This 11-bit field specifies the number of pixels per line. The value for this field should be programmed according to the following chart (H = the number of pixels per line).

| DATA DECODING | LCD DATA BUS | | | | |
|---|---|---|---|---|---|
| | 8 BITS/<br>NO SPLIT | 8 BITS/<br>SPLIT | 4 BITS/<br>NO SPLIT | 4 BITS/<br>SPLIT | 9 BITS/<br>NO SPLIT |
| 1-Bit Mono, 2- or 4-Bit Gray | 1/8 * H | 1/4 * H | 1/4 * H | 1/2 * H | — |
| 4-Bit Color Passive | 3/8 * H | 3/4 * H | 3/4 * H | — | — |
| 4-Bit Color TFT | — | — | — | — | H |

WBL—Wait Between Lines

Wait period between lines. It is measured in panel clock cycles.

### 18.5.3 LCD Vertical Configuration Register

The LCD vertical configuration register (LCVCR) is a 32-bit, memory-mapped, read/write register that holds the panel's vertical size and other configuration bits. It is cleared on hard reset.

LCVCR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | VPW | | | | — | — | — | LCD_AC | | | | VPC | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 848 | | | | | | | | | | | | | | | |
| BITS | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| FIELD | VPC | | | | | — | WBF | | | | | | | | | |
| RESET | | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ADDR | 848 | | | | | | | | | | | | | | | |

VPW —Vertical Sync Pulse Width (TFT display only)

This bit field controls the width of the vertical sync pulse with line number resolution. Programming it to *n* causes Vsync to be active for *n*-lines period. It is used for TFT display only and should be programmed to zero for all other displays.

LCD_AC—LCD AC Timing

This bit field specifies how many frames are displayed before the AC signal is toggled.

VPC—Vertical Pixel Count

This bit field specifies the number of lines on the LCD panel. The value in this field should be programmed according to the following chart (V = number of lines).

| DATA DECODING | LCD DATA BUS | | | | |
|---|---|---|---|---|---|
| | 8 BITS/<br>NO SPLIT | 8 BITS/<br>SPLIT | 4 BITS/<br>NO SPLIT | 4 BITS/<br>SPLIT | 9 BITS/<br>NO SPLIT |
| 1-Bit Mono, 2- or 4-Bit Gray | V | V/2 | V | V/2 | — |
| 4-Bit Color Passive | V | V/2 | V | — | — |
| 4-Bit Color TFT | — | — | — | — | V |

WBF—Wait Between Frames

This is the wait period between frames. It is measured in lines.

### 18.5.4 LCD Status Register

The LCD status register (LCSR) is an 8-bit register that is used to report certain events to the CPU. On recognition of an event, the LCD controller sets it's corresponding bit in the LCSR, regardless of the corresponding mask bit. The LCSR is a memory-mapped register and can be read at any time. A bit is cleared by writing a 1 (writing a 0 does not affect a bit's value) and more than one bit can be cleared at a time. This register is cleared at reset.

LCSR

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | — | BERR | UN | EOF |
| RESET | | | | | | | | |
| R/W | | | | | | R/W | R/W | R/W |
| ADDR | 858 | | | | | | | |

BERR—Bus Error

This status bit is set if a read cycle associated with the LCD controller is abnormally terminated.

UN—Underrun

An underrun condition is detected.

EOF—End Of Frame

This status bit is set on completion of a frame and a maskable interrupt is generated to the CPU.

### 18.5.5 Frame Buffer A Start Address Register

The frame buffer A start address (LCFAA) register is a 32-bit register that holds the start address of the frame buffer to be loaded into FIFO A. In this register, the DMA control presets the FIFO_A_ADDRESS register. Since all bursts must be 16-byte aligned, this register does not use the 4 least-significant bits of the address.

### 18.5.6 Frame Buffer B Start Address Register

The frame buffer B start address (LCFBA) register is a 32-bit register that holds the start address of the frame buffer to be loaded into FIFO B. It is only used in split mode. In this register, the DMA control presets the FIFO_B_ADDRESS register. Since all bursts are 16-byte aligned, this register does not use the 4 least-significant bits of the address.

### 18.5.7 Color RAM

The color RAM is used for mapping a 2-bit gray code to one of eight gray levels and for mapping 4- or 8-bit color code to 9 bits RGB value. The color RAM contains 256 entries and each entry is 16 bits wide. In grayscale mode, only the first four odd entries (1, 3, 5. and 7) are used. The color RAM is located in the dual-port RAM and is not initialized at reset. Refer to **Section 18.4.2.2 Color Generation** for more information.

### NOTE

Prior to color RAM programming, the LCCR must be programmed to the specific data coding (BPIX field).

**18.5.7.1 ONE-BIT/PIXEL MONOCHROME MODE.** When using this mode (based on LCCR), the color RAM should be configured as shown in the bit field.

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | — | — | — | — | — | — | — | — | GLC | | | |
| RESET | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U |
| 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 | (DPR) E00—FFF | | | | | | | | | | | | | | | |

In this mode, the color RAM is virtually not used. However, the user should program the first 16 entries of the color RAM in such a way that it becomes transparent. The GLC bit field has to be programmed to the value of the entry number. For entry 0, GLC=0000 and for entry 1, GLC=0001.

GLC—Gray Level Code (black and white display)
This bit field is a 4-bit code that represents the grayscale level.

**18.5.7.2 TWO-BIT/PIXEL GRAYSCALE MODE.**

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | — | — | — | — | GLCB | | | | GLCA | | | |
| RESET | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U |
| 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 | (DPR) E00—FFF | | | | | | | | | | | | | | | |

GLCA/GLCB—Gray Level Code (black and white display)
This bit field is a 4-bit code that represents the grayscale level for a given pixel code. GLCA and GLCB should be programmed to the same value.

- Entry 1 of the color RAM should be programmed with the gray level code corresponding to pixel code 00.
- Entry 3 of the color RAM should be programmed with the gray level code corresponding to pixel code 01.

- Entry 5 of the color RAM should be programmed with the gray level code corresponding to pixel code 10.
- Entry 7 of the color RAM should be programmed with the gray level code corresponding to pixel code 11.

### 18.5.7.3 FOUR BIT/PIXEL GRAYSCALE MODE.

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | — | — | — | — | GLCB | | | | GLCA | | | |
| RESET | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U |
| 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 | (DPR) E00—FFF | | | | | | | | | | | | | | | |

GLCA/GLCB—Gray Level Code (black and white display)

This bit field is a 4-bit code that represents the grayscale level for a given pixel code. GLCA and GLCB should be programmed to the same value. The first 16 odd entries of the color RAM (1, 3, 5) should be programmed.

### 18.5.7.4 PASSIVE, FOUR- AND EIGHT-BIT COLOR MODE.

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | R | | | | G | | | | B | | | |
| RESET | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U |
| R/W | R/W | R/W | R/W | R/W | R/W | W | W | W | W | W | W | W | W | W | W | W |
| ADDR | (DPR) E00—FFF | | | | | | | | | | | | | | | |

R—Red Level (color display)

This bit field is a 4-bit code that represents the red color level.

G—Green Level (color display)

This bit field is a 4-bit code that represents the green color level.

B—Blue Level (color display)

This bit field is a 4-bit code that represents the blue color level.

### 18.5.7.5 ACTIVE (TFT), FOUR- AND EIGHT-BIT COLOR MODE.

| BITS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIELD | — | — | — | — | R | | | — | G | | | — | B | | | — |
| RESET | | | | | | | | | | | | | | | | |
| R/W | | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | | |

R—Red Level (color display)
This bit field is a 3-bit code that represents the red color level.

G—Green Level (color display)
This bit field is a 3-bit code that represents the green color level.

B—Blue Level (color display)
This bit field is a 3-bit code that represents the blue color level.

## 18.5.8 LCD Panel Hookups

**Table 18-1. LCD Panel Connection to MPC821**

| PIN<br>PANEL TYPE | SHIFT/ CLOCK (PD[3]) | HSYNC/ LOAD (PD[4]) | VSYNC/ FRAME (PD[5]) | LCD_AC OE (PD[6]) | (MSB) LD0 (PD[7]) | LD1 (PD[8]) | LD2 (PD[9]) | LD3 (PD[10]) | LD4 (PD[11]) | LD5 (PD[12]) | LD6 (PD[13]) | LD7 (PD[14]) | (LSB) LD8 (PD[15]) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SHARP–LQ9D021 (color TFT) | CK-P[10] | HSYNC-P[6] | VSYNC-P[4] | ENAB-P[28] | R2-P[7] | R1-P[5] | R0-P[3] | G2-P[19] | G1-P[17] | G0-P[13] | B2-P[29] | B1-P[27] | B0-P[25] |
| SHARP–LM4801F (PASSIVE B&W) | CP2-P[3] | CP1-P[2] | S-P[1] | | | | | | | DU0-P[8] | DU1-P[9] | DU2-P[10] | DU3-P[11] |
| SHARP–LM64P839 (PASSIVE B&W) SPLIT FRAME | CP2-P[3] | CP1-P[2] | S-P[1] | | | DU0-P[8] | DU1-P[9] | DU2-P[10] | DU3-P[11] | DL0-P[15] | DL1-P[13] | DL2-P[14] | DL3-P[15] |
| HITACHI–LMG7211URFR (PASSIVE B&W) | CL2-P[3] | CL1-P[2] | FRAME-P[1] | | | | | | | D0-P[8] | D1-P[9] | D2-P[10] | D3-P[11] |
| SHARP LM32K07 (PASSIVE B&W) | CP2-P[10] | CP1-P[11] | S-P[12] | | | | | | | D0-P[6] | D1-P[5] | D2-P[4] | D3-P[3] |

# SECTION 19
# DEVELOPMENT SUPPORT

Emulators require a level of control and observation that are in sharp contrast to the trend of modern microcomputers and microprocessors in which many bus cycles are directed to internal resources and are not externally visible. The same is true for bus analyzers. To enhance support for development tools, some of the development support functions are implemented in the silicon. Program flow tracking, internal watchpoint and breakpoint generation, and emulation systems control over the activity of the CPU (debug mode) are just some of the features that allow the user to efficiently debug systems based on the MPC821.

## 19.1 PROGRAM FLOW TRACKING

The MPC821 provides many options for tracking program flows that impact performance in varying degrees. The information provided while tracking code flow can be compressed and captured externally and then parsed by a post-processing program using the microarchitecture defined here. The program instruction flow is visible on the external bus when the MPC821is programmed to operate in serialized mode and to show all fetch cycles on the external bus. When working in this mode, although tracking of the program instruction flow is simpler, the performance of the MPC821 is much lower than when working in regular mode. See Table 19-18 for programming the CPU to operate in this mode.

The MPC821 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), increases performance but makes 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 *a*rchitecturally executed.

To reconstruct program trace, the program code combined with additional MPC821 information is required. Reporting program trace during retirement significantly complicates the visibility and increases the die size for the two reasons. More than one instruction can retire in a clock cycle and it is harder to report on indirect branches during retirement. Because of this, program trace is reported during fetch and helps reconstruct the instructions that actually retire after fetch canceled instructions are reported. Instructions are fetched sequentially until branches (direct or indirect), exceptions or interrupts appear in the program flow or until a stall in execution forces the machine to avoid fetching the next address. These instructions may be architecturally executed or they may be canceled in some stage of the machine pipeline.

Therefore, the additional information 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 all indirect flow changes targets. Indirect flow changes include all branches using the link and count registers as the target address, all interrupts/ exceptions, as well as rfi and mtmsr because it may cause context switch.
- 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. The issue of data compression that could reduce the amount of memory needed by the debug system is also mentioned.

## 19.1.1 Functional Description

**19.1.1.1 THE INTERNAL HARDWARE.** To make the events that occur in the machine visible, a few dedicated pins are used. Also, a special bus cycle attribute called program trace cycle is defined. The program trace cycle attribute is attached to all fetch cycles resulting from indirect flow changes. When program trace recording is required, the user can ensure these cycles are visible on the external bus.

The VSYNC indication, when asserted, forces all fetch cycles marked with the program trace cycle attribute to be visible on the external bus, even if their data is found in one of the internal devices. To enable the external hardware to properly synchronize with the internal activity of the CPU, the assertion and negation of VSYNC 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. For more information on the activity of the external hardware during program trace, refer to **Section 19.1.1.5 The External Hardware**.

**NOTE**

To keep the pin count of the chip as low as possible, VSYNC is not implemented as one of the chip’s external pins. Instead, it is asserted and negated using the serial interface implemented in the development port. For more information on this interface, refer to **Section 19.3.2 Development Port**. Forcing the CPU to show all fetch cycles marked with the program trace cycle attribute can be accomplished by either asserting VSYNC (as mentioned above) or by programming the fetch show cycle bits in the instruction support control register (ICTRL). For more information refer to **Section 19.1.2 Instruction Fetch Show Cycle Control**.

When VSYNC indication is asserted, all fetch cycles marked with the program trace cycle attribute become visible on the external bus. These cycles generate regular bus cycles when the instructions reside in one of the external devices or generate address-only cycles when the instructions are in one of the internal devices (I-cache and internal ROM). When VSYNC is asserted, some performance degradation occurs because of the additional external bus cycles. Since this performance degradation is expected to be very small, it is possible to program the machine to show all indirect flow changes and, therefore, always perform these additional external bus cycles and maintain the same behavior when VSYNC is asserted and negated. For more information see Table 19-18.

The status pins are divided into two groups:

- VF [0 . . 2]

  Instruction queue status: denotes the type of the last fetched instruction or how many instructions were flushed from the instruction queue. These status pins are used for both functions because queue flushes only happen in clocks where there is no fetch type information to be reported. See Table 19-1 for the definition of possible instruction types.

  — Possible instruction queue flushes:
  000–None
  001–1 instruction was flushed from the instruction queue
  010–2 instructions were flushed from the instruction queue
  011–3 instructions were flushed from the instruction queue
  100–4 instructions were flushed from the instruction queue
  101–5 instructions were flushed from the instruction queue
  110–Reserved
  111–Like VF = ‘111’

- 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:
  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 and should be ignored by the program trace external hardware. For details, refer to **Section 19.3.1 Debug Mode Support**.

Table 19-1. VF Pins Encoding

| VF | INSTRUCTION TYPE | VF NEXT CLOCK WILL HOLD |
|---|---|---|
| 000 | None | More Instruction Type Information |
| 001 | Sequential | |
| 010 | Branch (Direct or Indirect) Not Taken | |
| 011 | VSYNC Was Asserted/negated and Therefore the Next Instruction Will be Marked With the Program Trace Cycle Attribute | |
| 100 | Interrupt/Exception Taken, the Target Will be Marked With the Program Trace Cycle Attribute | Queue Flush Information[2] |
| 101 | Branch Indirect Taken, rfi, mtmsr, isync and in Some Cases mtspr, the Target Will be Marked With the Program Trace Cycle Attribute[1] | |
| 110 | Branch Direct Taken | |
| 111 | Branch (Direct or Indirect) Not Taken | |

NOTES: 1. Refer to **Section 19.1.1.4 Sequential Instructions Marked As Indirect Branch**.

2. Unless the next clock VF = 111, refer to **Section 19.1.1.2 Queue Flush Information Special Case**.

**19.1.1.2 QUEUE FLUSH INFORMATION SPECIAL CASE.** There is one special case where the queue flush information is expected on the VF pins. This is easily monitored since the only case where this can happen is when VF =111 and the maximum number of possible queue flushes is five.

**19.1.1.3 PROGRAM TRACE WHEN IN DEBUG MODE.** When entering debug mode an interrupt/exception taken is reported on the VF pins (VF=’100’) and a cycle marked with the program trace cycle is made externally visible. When the CPU is in debug mode, the VF pins equal ‘000’ and the VFLS pins equal ‘11’. For more information on the MPC821 debug mode, refer to **Section 19.3 Development System Interface**.

If VSYNC is asserted/negated while the CPU is in debug mode, this information is reported when the first VF pins report as the CPU returns to regular mode. If VSYNC was not changed while in debug mode, the first VF pins report will be encoded as VF=’101’ (indirect branch) due to the rfi instruction 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.

**19.1.1.4 SEQUENTIAL INSTRUCTIONS MARKED AS INDIRECT BRANCH.** There are instances where non-branch (sequential) instructions can affect the machine in a similar manner of indirect branch instructions. These instructions include rfi, mtmsr, isync, and mtspr to registers CMPA-F, ICTRL, ICR, and DER.

These instructions are marked by the CPU as indirect branch instructions (VF = ‘101’) and the following instruction address is marked with the program trace cycle attribute, as if it was an indirect branch target. Therefore, when one of these special instructions is detected in the CPU, the address of the following instruction is externally visible. The reconstructing software is now able to correctly evaluate the effect of these instructions.

**19.1.1.5 THE EXTERNAL HARDWARE.** When program trace is needed, the external hardware must sample the status pins (VF and VFLS) of every clock and mark the address of all cycles with the program trace cycle attribute. Program trace is used in various ways, but back trace and window trace are two strong possibilities.

**19.1.1.5.1 Back trace.** This 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, the external hardware should start sampling VF and VFLS pins and the address of all cycles marked with the program trace cycle attribute immediately after reset is negated. Since the instruction show cycles programming default to show all out of reset, all cycles marked with the program trace cycle attribute are visible on the external bus. VSYNC should be asserted sometime after reset and negated when the actual event occurs. If show all is not the preferred for the instruction show cycles prior to the actual event occurring, VSYNC must be asserted before the changing instruction show cycles programming from show all. Notice that if the timing of the event in question is unknown, it is possible to use cyclic buffers. After VSYNC is negated the trace buffer contains the program flow trace of the program executed before the event in question occurred.

**19.1.1.5.2 Window trace .** This is useful when a record of the program trace between two events is required. If window trace is needed, the VSYNC pin should be asserted between these two events. After the VSYNC pin is negated the trace buffer contains information describing the program trace of the program executed between the two events.

**19.1.1.5.3 Synchronizing the Trace Window to the CPU Internal 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 CPU, 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, refer to **Section 19.3 Development System Interface**.

The following is a possible set of steps that enable the user to synchronize the trace window to the internal CPU events:

- Enter debug mode, either immediately out of reset of using the debug mode request.
- Program the hardware to break on the event that marks the start of the trace window using the control registers defined in **Section 19.2 Watchpoints And Breakpoints Support**.
- Enable debug mode entry for the programmed breakpoint in register DER (refer to Table 19-24).
- Return to the regular code run (refer to **Section 19.3.1.7 Exiting Debug Mode**).

- The hardware generates a breakpoint when the event in question is detected and the machine enters debug mode (refer to **Section 19.3.1.3 Entering Debug Mode**).
- Program the hardware to break on the event that marks the end of the trace window.
- Assert VSYNC.
- Return to the regular code run. The first report on the VF pins is VSYNC, where VF =’011’.
- The external hardware starts sampling the program trace information after the VF pins indicate VSYNC.
- The hardware generates a breakpoint when the event in question is detected and the machine enters debug mode.
- Negate VSYNC.
- Return to the regular code run (issue an rfi). The first encoding on the VF pins is VSYNC, where VF =’011’.
- The external hardware stops sampling the program trace information after recognizing VSYNC on the VF pins.

**19.1.1.5.4 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 the following table to calculate the trace window start address.

**Table 19-2. Detecting the Trace Buffer Start Point**

| VF1 | VF2 | STARTING POINT | DESCRIPTION |
|---|---|---|---|
| 011<br>VSYNC | 001<br>Sequential | T1 | VSYNC Asserted. Followed by a Sequential Instruction.<br>The Start Address is T1. |
| 011<br>VSYNC | 110<br>Branch Direct Taken | T1 - 4 +<br>Offset(T1 - 4) | VSYNC Asserted. Followed by a Taken Direct Branch.<br>The Start Address is the Target of the Direct Branch. |
| 011<br>VSYNC | 101<br>Branch Indirect Taken | T2 | VSYNC Asserted. Followed by a Taken Indirect Branch.<br>The Start Address is the Target of the Indirect Branch. |

**19.1.1.5.5 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 the assertion/negation of VSYNC. When VF = ‘011’, it is a VSYNC assertion/negation report only if the prior value of VF was ‘000’, ‘001’, or ‘010’.

**19.1.1.5.6 Detecting the Trace Window End Address.** The information on the status pins that describes the last fetched instruction and last queue/history buffer flush, changes every clock. Cycles marked as program trace cycle are generated on the external bus only when the system interface unit (SIU) arbitrates over the external bus. Therefore, there is a delay between the report that a cycle marked as program trace cycle is performed and the actual time that this cycle can be detected on the external bus.

When VSYNC is negated by the user (through the serial interface of the development port), the CPU 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. Therefore, at the last stage of the reconstruction software, the last two instructions should be ignored.

**19.1.1.6 COMPRESS.** To store all the information generated on the pins during program trace (5 bits per clock + 30 bits per show cycle) a large memory buffer is required. However, since this information includes events that were canceled, compression is possible and can be very beneficial in this situation. External hardware can be added to eliminate all canceled instructions and reports only on taken/not taken branches, indirect flow change, and the number of sequential instructions after the last flow change.

## 19.1.2 Instruction Fetch Show Cycle Control

Instruction fetch show cycles are controlled by the bits in the ICTRL and the state of VSYNC. The following table defines the level of fetch show cycles generated by the CPU. For information on the fetch show cycles control bits see Table 19-18.

**Table 19-3. Fetch Show Cycles Control**

| VSYNC | ISCTL (29:31)<br>INSTRUCTION<br>FETCH SHOW CYCLE<br>CONTROL BITS | SHOW CYCLES GENERATED |
|---|---|---|
| X | 000 | All Fetch Cycles |
| X | X01 | All Change of Flow<br>(Direct and Indirect) |
| X | X10 | All Indirect Change of Flow |
| 0 | X11 | No Show Cycles Are Performed |
| 1 | X11 | All Indirect Change of Flow |

NOTE: When ICTRL(29:31) is set to 010 or 110, the $\overline{\text{STS}}$ functionality of the OP2/MODCK1/$\overline{\text{STS}}$ pin must be enabled by writing 10 or 11 to the DBGC field of the SIUMCR. The address on the external bus should only be sampled when $\overline{\text{STS}}$ is asserted.

A cycle marked with the program trace cycle attribute is generated for any change in the VSYNC state (assertion or negation).

## 19.2 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 CPU supports watchpoints that are generated inside the core as well as breakpoints that are 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 any of the peripherals of the system, including those found on or outside the MPC821 or those found by an external development system. Peripherals found on the external bus use the serial interface of the development port to assert the external breakpoint.

In the CPU, as in other RISC processors, saving/restoring machine state on the stack during exception handling is done in the software. When the software is in the middle of saving/restoring the machine state, the $MSR_{RI}$ bit is cleared. Exceptions that occur are handled by the CPU when the $MSR_{RI}$ bit is clear and they result in a nonrestartable machine state. For more information refer to **Section 6.3.5.1 Restartability After An Interrupt**.

In general, breakpoints are recognized in the CPU only when the $MSR_{RI}$ bit is set, which guarantees machine restartability after a breakpoint. In this working mode, breakpoints are said to be masked. There are times when it is preferable to enable breakpoints even when the $MSR_{RI}$ bit is clear, with the possible risk of causing 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 the external pins, regardless of the value of the $MSR_{RI}$ bit. The counters, although counting watchpoints, are part of the internal breakpoints logic and are not decremented when the CPU is operating in the masked mode and the $MSR_{RI}$ bit is clear. Figure 5 19-1 illustrates the watchpoints and breakpoints support of the CPU.

### 19.2.1 Internal Watchpoints and Breakpoints

This section describes the internal breakpoints and watchpoints support of the CPU. For more information on external breakpoints support, refer to **Section 19.3 Development System Interface**. Internal breakpoint and watchpoint support is based on:

- Eight comparators comparing information on instruction and load/store cycles
- Two counters
- Two AND-OR logic structures

The comparators perform compare on the instruction address (I-address), the load/store address (L-address), and the load/store data (L-data). The comparators are able to detect the following conditions:

- Equal to
- Not equal to
- Greater than
- Less than

Greater-than-or-equal-to and less-than-or-equal-to are easily obtained from these four conditions. Refer to **Section 19.2.1.6 Generating Six Compare Types** for more information. Using the AND-OR logic structures “in range” and “out of range” detections (on address and data) are supported. Using the counters, it is possible to program a breakpoint to be recognized after an event was detected a predefined number of times.

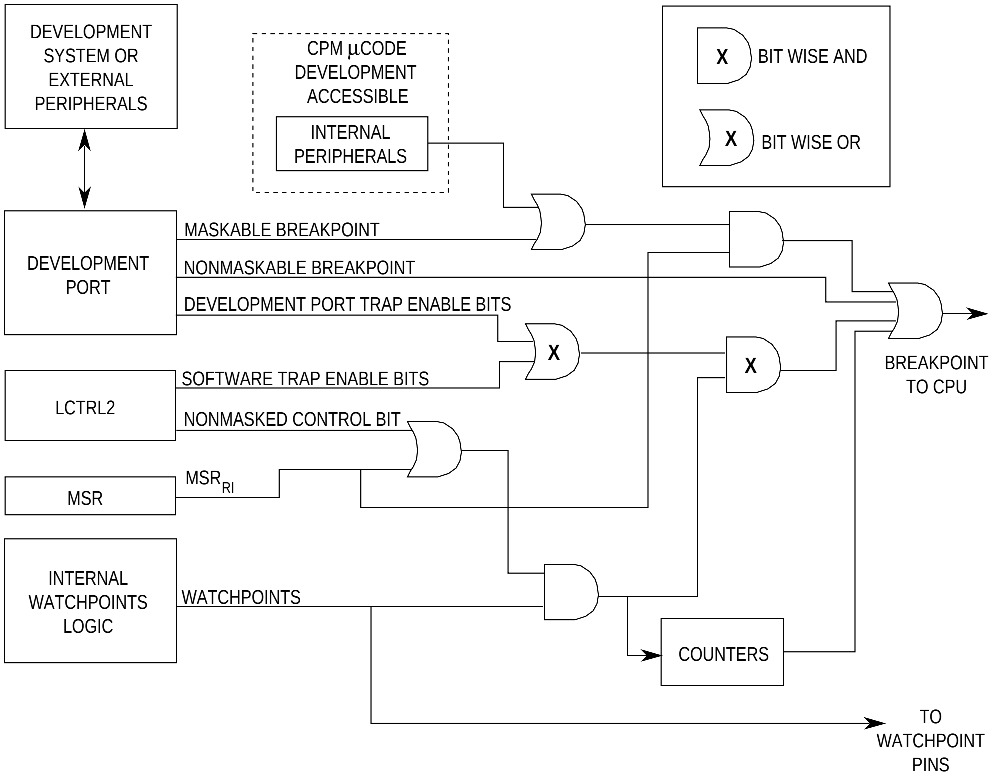


**Figure 19-1. Watchpoints and Breakpoint Support in the CPU**

The L-data comparators operate on load or store fixed-point data. When operating on fixed-point data the L-data comparators perform compare on bytes, half-words and words and treat numbers as either signed or unsigned values. The comparators generate match events then instruction match events enter the instruction AND-OR logic where the instruction watchpoints and breakpoint are generated. The asserted instruction watchpoints can generate the instruction breakpoint. Two different events can decrement one of the counters. When a counter on one of the instruction watchpoints expires, the instruction breakpoint is asserted.

The instruction watchpoints and the load/store match events on address/data enter the load/store AND-OR logic where the load/store watchpoints and breakpoint are generated. The load/store watchpoints (when asserted) can generate the load/store breakpoint or decrement one of the counters. When a counter on one of the load/store watchpoints expires, the load/store breakpoint is asserted.

Watchpoints progress in the machine and are reported on retirement. Internal breakpoints progress in the machine until they reach the top of the history buffer when the machine branches to the breakpoint exception routine. To allow use of the breakpoint features without adding restrictions on the software, the address of the load/store cycle that generated the load/store breakpoint is not stored in the data address register (DAR). In a load/store breakpoint, the address of the load/store cycle that generated the breakpoint is stored in the breakpoint address register (BAR).

**19.2.1.1 FEATURES.** The following list summarizes the important 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 least-significant bit masking, according to the size of the bus cycle for the byte and half-word working modes. For details, refer to **Section 19.2.1.3 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 fixed-point data. Match is detected only on the valid part of the data bus (according to the cycle’s size and the two address least-significant bits).
- No internal breakpoint/watchpoint support for unaligned words and half-words.
- The L-data comparators can be programmed to treat fixed-point numbers as signed or unsigned values.
- Combined comparator pairs to detect in and out of range conditions, including either signed or unsigned values on the L-data.
- A programmable AND-OR logic structure between the four instruction comparators results in five outputs, four instruction watchpoints, and one instruction breakpoint.

- A programmable AND-OR logic structure between the four instruction watchpoints and the four load/store comparators results in three outputs, two load/store watchpoints, and one load/store breakpoint.
- Five watchpoint pins, three for the instruction and two for the load/store.
- Two dedicated 16-bit down counters. Each can be programmed to count either an instruction watchpoint or a load/store watchpoint. Only architecturally executed events are counted, (count up is performed in case of recovery).
- On-the-fly trap enable programming of the different internal breakpoints using the serial interface of the development port (refer to **Section 19.3.2 Development Port**). Software control is also available.
- Watchpoints do not change the timing of the machine.
- Internal breakpoints and watchpoints are detected on the instruction during instruction fetch.
- Internal breakpoints and watchpoints are detected on the load/store during load/store bus cycles.
- Instruction and load/store breakpoints and watchpoints are handled on retirement and then reported.
- Breakpoints and watchpoints on recovered instructions (as a result of exceptions, interrupts or miss prediction) are not reported and do not change the timing of the machine.
- Instructions with instruction breakpoints are not executed. The machine branches to the breakpoint exception routine before it executes the instruction.
- Instructions with load/store breakpoints are executed. The machine branches to the breakpoint exception routine after it executes the instruction. The address of the access is placed in the BAR register.
- Load/store multiple and string instructions with load/store breakpoints first finish execution and then the machine branches to the breakpoint exception routine.
- Load/store data compare is 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.
- Both “go to x” and “continue” working modes are supported for instruction breakpoints.

**19.2.1.2 RESTRICTIONS.** There are times when the same watchpoint can be detected more than once during the execution of a single instruction. For example, load/store watchpoint detected on more than one transfer when executing load/store multiple/string or load/store watchpoint detected on more than one byte when working in byte mode. In these cases only one watchpoint of the same type is reported for a single instruction. Similarly, only one watchpoint of the same type can be counted in the counters for a single instruction. Since watchpoint events are reported upon the retirement of the instruction that caused the event and more than one instruction can retire from the machine in a single clock, ensuing events may be reported in the same clock. Moreover, if the same event is detected on more than one instruction (tight loops or range detection) can only be reported once. The internal counters count correctly in these cases.

**19.2.1.3 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, the user should write the required match value to the correct half-word of the data comparator and the mask in the L-data comparator. In the situation where the user prefers to break on bytes, the byte mask for each L-comparator and the bytes to be matched must be written in the data comparator.

Since bytes and half-words can be accessed using a larger data width instruction, it is impossible for the user to predict the exact value of the L-address lines when the requested byte/half-word is accessed. If the matched byte is byte 2 of the word and accessed using a load word instruction, the L-address value will be of the word (byte 0). Therefore, the CPU masks the two least significant bits of the L-address comparators whenever a word access is performed and the least significant bit whenever a half-word access is performed. Address range is only supported when aligned according to the access size.

#### 19.2.1.3.1 Examples.

- **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: The event will be correctly detected if the compiler chooses a load/store instruction with data size of half-word. If the compiler chooses load/store instructions with data size greater than half-word (word, multiple), there might be some false detections.

- These can only be ignored by the software that handles the breakpoints. The following figure illustrates this partially supported scenario:

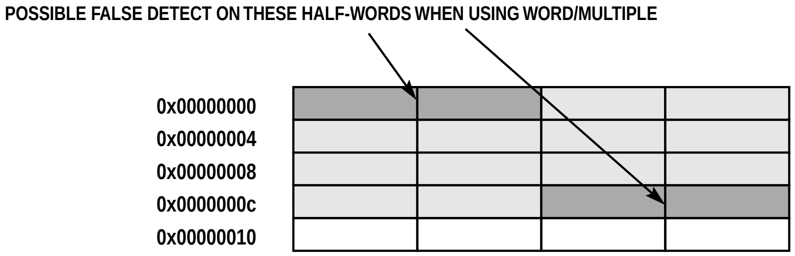

POSSIBLE FALSE DETECT ON THESE HALF-WORDS WHEN USING WORD/MULTIPLE
0x00000000
0x00000004
0x00000008
0x0000000c
0x00000010

Figure 19-2. Partially Supported Watchpoints/Breakpoint Example

**19.2.1.4 CONTEXT DEPENDENT FILTER.** The CPU can only be programmed to recognize internal breakpoints when the $MSR_{RI}$ bit is set (masked mode) or to always recognize internal breakpoints (nonmasked mode). When the CPU is programmed only to recognize internal breakpoints (when $MSR_{RI}$ =1) it is possible to debug all parts of the code, except when registers SRR0 and SRR1, DAR, and DSISR are busy and $MSR_{RI}$ =0 (in the prologues and epilogues of interrupt/exception handlers).

When the CPU is programmed to recognize internal breakpoints, it is possible to debug all parts of the code. However, if an internal breakpoint is recognized when $MSR_{RI}$ =0 (registers SRR0 and SRR1 are busy), the machine enters into a nonrestartable state. For more information refer to **Section 6.3.5.1 Restartability After An Interrupt**. When working in the masked mode all internal breakpoints detected when $MSR_{RI}$ =0 are lost and detected watchpoints are not counted by the debug counters. Detected watchpoints are always reported on the external pins, regardless of the value of the $MSR_{RI}$ bit.

The CPU defaults to the masked mode after reset. The CPU is input in the nonmasked mode by setting the BRKNOMSK bit in the LCTRL2 register.The BRKNOMSK bit controls all internal breakpoints (I-breakpoints and L-breakpoints). See Table 19-20 for details.

**19.2.1.5 IGNORE FIRST MATCH.** To facilitate the debugger utilities of “continue” and “go from x”, the ignore first match option is supported for the instruction breakpoints. When an instruction breakpoint is first enabled, the first instruction will not cause an instruction breakpoint if the ignore first match (IFM) bit in the instruction support control register (ICTRL) is set. This case is used for “continue” utilities. When IFM is clear, every matched instruction can cause an instruction breakpoint. This case is used for “go from x”. TheIFM bit is set by the software and cleared by the hardware following 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.

**19.2.1.6 GENERATING SIX COMPARE TYPES.** Using the four compare types–equal, not equal, greater than, and less than–it is possible to generate two additional compare types:

- Greater than or equal to
- Less than or equal to

Generating the greater than or equal compare type can be accomplished by using the greater than compare type and programming the comparator to the value in question minus 1. Likewise, generating the less than or equal compare type can be accomplished by using the less than compare type and programming the comparator to the value in question plus 1. Notice that this method does not work for the following boundary cases:

- Less than or equal 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 when in signed mode (0111...1).
- Greater than or equal of the maximum negative number when in signed mode (1000...).

These boundary cases do not require special support because they are considered ‘always true’. These cases can be programmed using the ignore option of the load/store watchpoint programming. See Table 19-20 for more information.

## 19.2.2 Functional Description

**19.2.2.1 INSTRUCTION SUPPORT DETAILED DESCRIPTION.** There are four instruction address comparators (A, B, C, and D). Each one 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” detect.

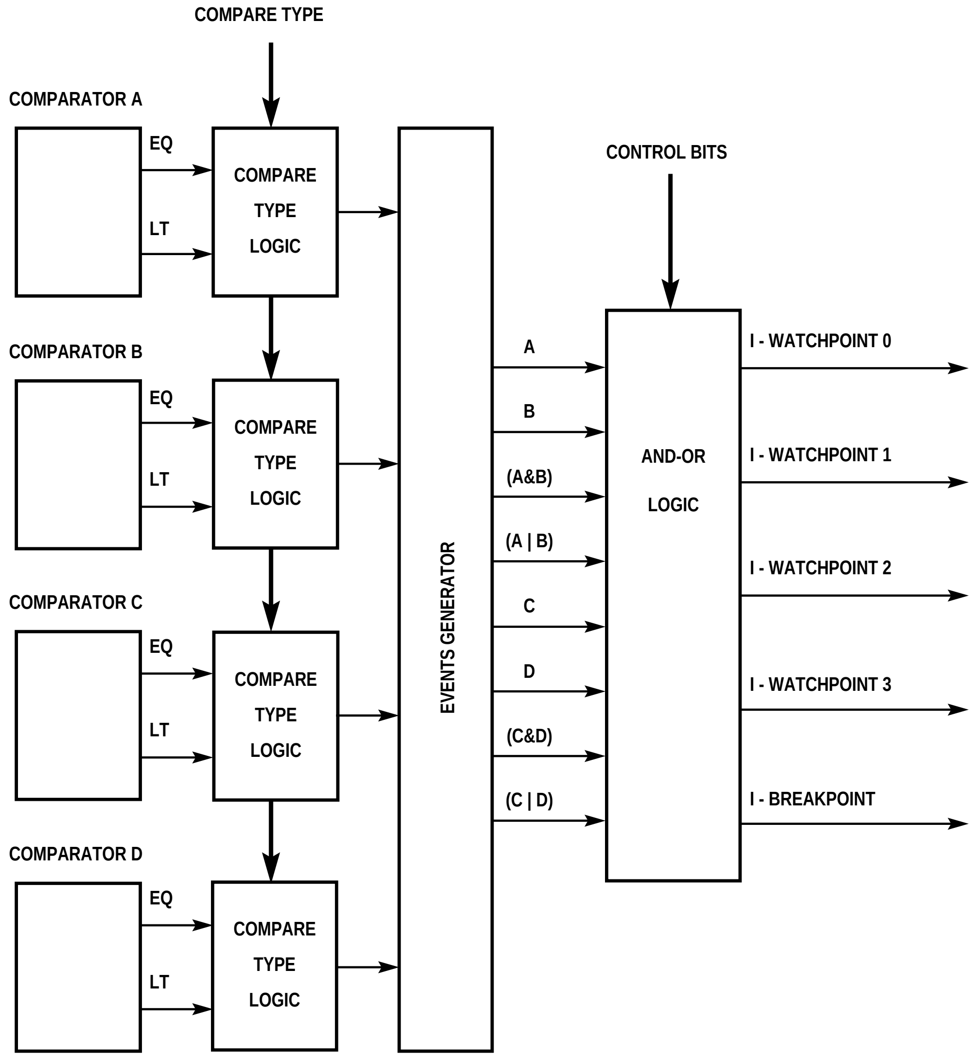


**Figure 19-3. Instruction Support General Structure**

Table 19-4. Instruction Watchpoints Programming Options

| NAME | DESCRIPTION | PROGRAMMING OPTIONS |
|---|---|---|
| IW0 | 1st Instruction Watchpoint | Comparator A<br>Comparators (A & B) |
| IW1 | 2nd Instruction Watchpoint | Comparator B<br>Comparator (A \| B) |
| IW2 | 3rd Instruction Watchpoint | Comparator C<br>Comparators (C & D) |
| IW3 | 4th Instruction Watchpoint | Comparator D<br>Comparator (C \| D) |

**19.2.2.2 LOAD/STORE SUPPORT DETAILED DESCRIPTION.** There are two load/store address comparators (E and F). Each one compares the 32 address bits and the cycle's attributes (read/write). The two least significant bits are masked ignored whenever a word is accessed and the least significant bit is masked whenever a half-word is accessed. Each comparator generates two output signals—equal and less than. These signals generate one of four events from each comparator—equal, not equal, greater than, or less than. For more information, refer to **Section 19.2.1.3 Byte And Half-Word Working Modes**).

There are two load/store data comparators G and H. Each is 32 bits wide and can be programmed to treat numbers as signed or unsigned values. 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). When operating in byte mode all signals are significant. When operating in half-word mode, only four signals from each 32-bit comparator are significant and when operating in word mode, only two signals are significant.

One of the following four match events are 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 will be asserted. Using the match indication signals, four load/store data events are generated as shown in Table 19-5.

Table 19-5. Load/Store Data Events

| EVENT NAME | EVENT FUNCTION (SEE NOTE) |
|---|---|
| G | (Gmatch0 \| Gmatch1 \| Gmatch2 \| Gmatch3) |
| H | (Hmatch0 \| Hmatch1 \| Hmatch2 \| Hmatch3) |
| (G & H) | ((Gmatch0 & Hmatch0) \| (Gmatch1 & Hmatch1) \| (Gmatch2 & Hmatch2) \| (Gmatch3 & Hmatch3)) |
| (G \| H) | ((Gmatch0 \| Hmatch0) \| (Gmatch1 \| Hmatch1) \| (Gmatch2 \| Hmatch2) \| (Gmatch3 \| Hmatch3)) |

NOTE: & denotes a logical AND, but | 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 19-6. Load/Store Watchpoints Programming Options

| NAME | DESCRIPTION | INSTRUCTION EVENTS PROGRAMMING OPTIONS | L-ADDRESS EVENTS PROGRAMMING OPTIONS | L-DATA EVENTS PROGRAMMING OPTIONS |
|---|---|---|---|---|
| LW0 | 1st Load/Store Watchpoint | IW0, IW1, IW2, IW3, Ignore Instruction Events | Comparator E<br>Comparator F<br>Comparators (E & F)<br>Comparators (E \| F)<br>Ignore L-addr Events | Comparator G<br>Comparator H<br>Comparators (G & H)<br>Comparators (G \| H)<br>Ignore L-data Events |
| LW1 | 2nd Load/Store Watchpoint | IW0, IW1, IW2, IW3, Ignore Instruction Events | Comparator E<br>Comparator F<br>Comparators (E & F)<br>Comparators (E \| F)<br>Ignore I-addr Events | Comparator G<br>Comparator H<br>Comparators (G & H)<br>Comparators (G \| H)<br>Ignore L-data Events |

When programming the 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.

**19.2.2.3 THE COUNTERS.** There are two 16-bit down counters. Each counter is able to count one of the instruction watchpoints or one of the load/store watchpoints. Both generate the corresponding breakpoint when they reach zero. When working in the masked mode, the counters do not count detected watchpoints when $MSR_{RI}$ =0. For details, refer to **Section 19.2.1.4 Context Dependent Filter**. Counter values are not predictable if they are counting watchpoints programmed on the instructions that alter the counters. Readings from the counters when active must be synchronized by inserting a sync instruction before the read is performed.

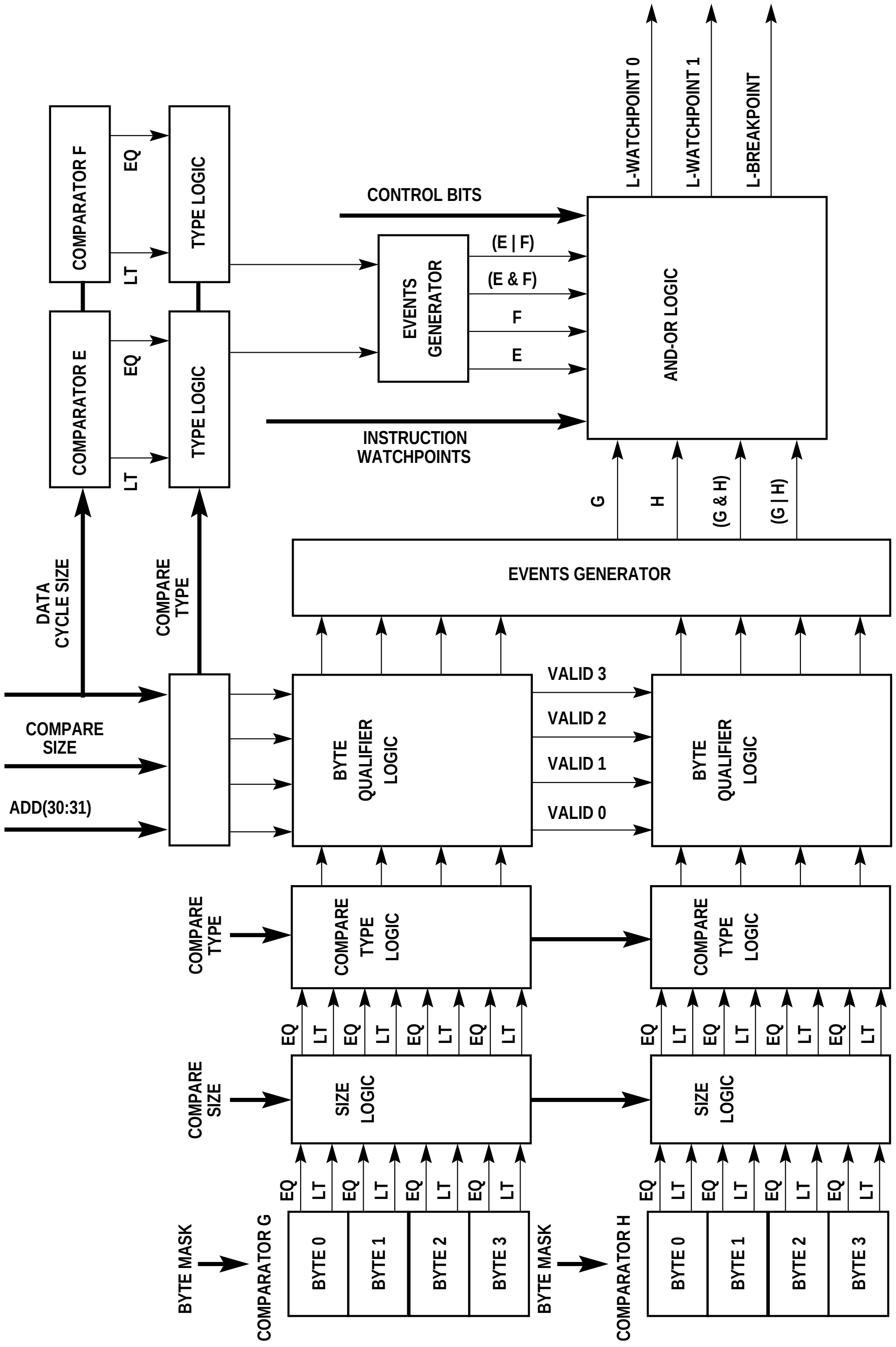


Figure 19-4. Load/Store Support General Structure

NOTE

When programmed to count instruction watchpoints, the last instruction that decrements the counter to zero is treated like any other instruction breakpoint in the sense that it is not executed before the machine branches to the breakpoint exception routine. As a side effect of this behavior, the value of the counter inside the breakpoint exception routine equals 1 and not zero, 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 the sense that it is executed before the machine branches to the breakpoint exception routine. Therefore, the value of the counter inside the breakpoint exception routine equals zero.

**19.2.2.4 TRAP ENABLE PROGRAMMING.** The trap enable bits can be programmed by regular software (only if $MSR_{PR}$ =0) using the mtspr instruction or on-the-fly using the special development port interface. For more information, refer to **Section 19.3.2.4 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 the ICTRL and the LCTRL2 registers using the mtspr instruction. For the exact bits placement, refer to Tables 19-18 and 19-20.

## 19.3 DEVELOPMENT SYSTEM INTERFACE

When debugging an existing system it is sometimes helpful to be able to do so without making any changes. Although, in some cases it is not helpful and may even make it impossible to add load to the lines connected to the existing system. The development system interface of the CPU supports this configuration.

The development system interface of the CPU uses the development port, which is a dedicated serial port and, therefore, does not need any of the regular system interfaces. Controlling the activity of the system from the development port is accomplished when the CPU is in debug mode. The development port is a relatively inexpensive interface that allows the development system to operate in a lower frequency than the CPU’s frequency. It is also possible to debug the CPU using monitor debugger software. For more information, refer to **Section 19.4 Software Monitor Debugger Support**.

Debug mode is a state where the CPU fetches all instructions from the development port. When in debug mode, 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 purposes two possible working modes are defined—debug mode enable and debug mode disable. These working modes are only selected during reset. For details, refer to **Section 19.3.1.2 Debug Mode Enable vs. Debug Mode Disable**.

The user can work in debug mode directly out of reset or the CPU can be programmed to enter into the debug mode as a result of a predefined sequence of events. These events can be any interrupt or exception in the CPU system, including the internal breakpoints, in combination with two levels of development port requests and one peripheral breakpoint request generated internally and 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. When in debug mode, the rfi instruction returns the machine to its regular work mode. The relationship between the debug mode logic to the rest of the CPU chip is illustrated in the following figure.

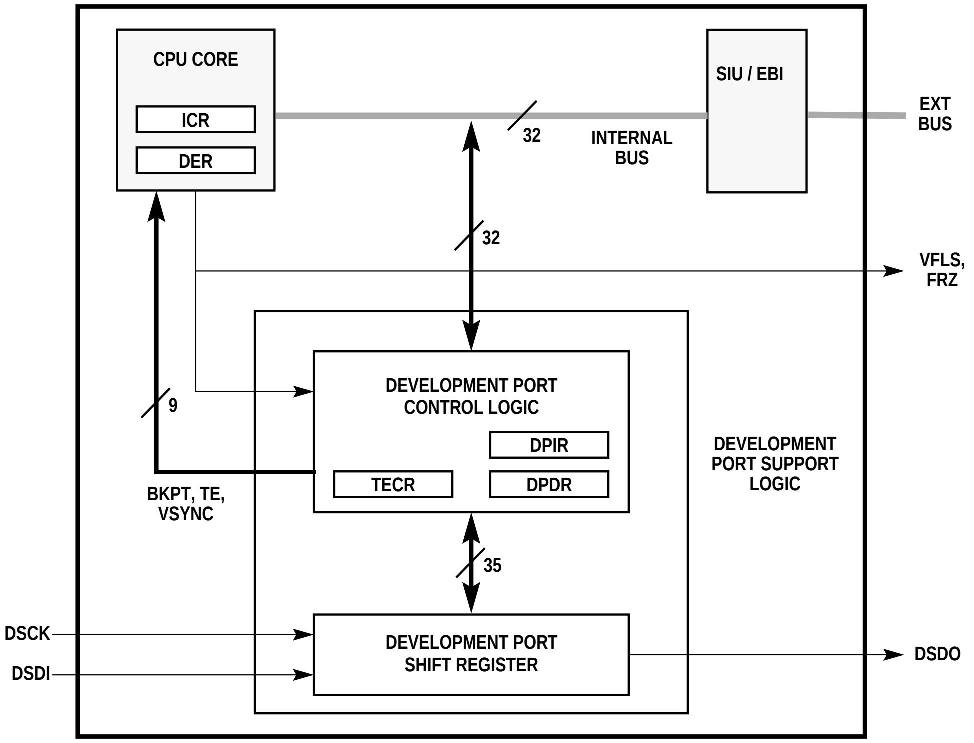


**Figure 19-5. Functional Diagram of the MPC821 Debug Mode Support**

The development port provides a full duplex serial interface for communications between the internal development support logic of the CPU and an external development tool. The development port can operate in two working modes–trap enable mode and debug mode.

The trap enable mode is used to shift the following control signals into the CPU internal development support logic.

- Instruction trap enable bits is used for on-the-fly programming of the instruction breakpoint.
- Load/store trap enable bits is used for on-the-fly programming of the load/store breakpoint.
- 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 CPU. For more details, refer to **Section 19.3.2 Development Port**.

### 19.3.1 Debug Mode Support

**19.3.1.1 GENERAL.** The debug mode of the CPU provides the development system with the following functions:

- Controls and maintains the execution of the processor in all circumstances.
  - — The development port can force the CPU to enter the debug mode even when external interrupts are disabled.
- Debug mode can be entered immediately out of reset, thus allowing the user to debug a system without ROM.
- The user can selectively define (through an enable register) the events that cause the machine to enter into debug mode.
- Contains a cause register that indicates why debug mode is entered.
- After entry into debug mode program execution continues from the location where they entered debug.
- 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 the privileged state ($MSR_{PR}$ =0) in debug mode, allowing execution of any instruction and access to any storage location.
- An OR signal of all interrupt cause register bits enables the development port to detect pending events while already in debug mode. For example, the development port can detect a debug mode access to a nonexisting memory space.

Figure 19-6 illustrates the debug mode logic implemented in the core.

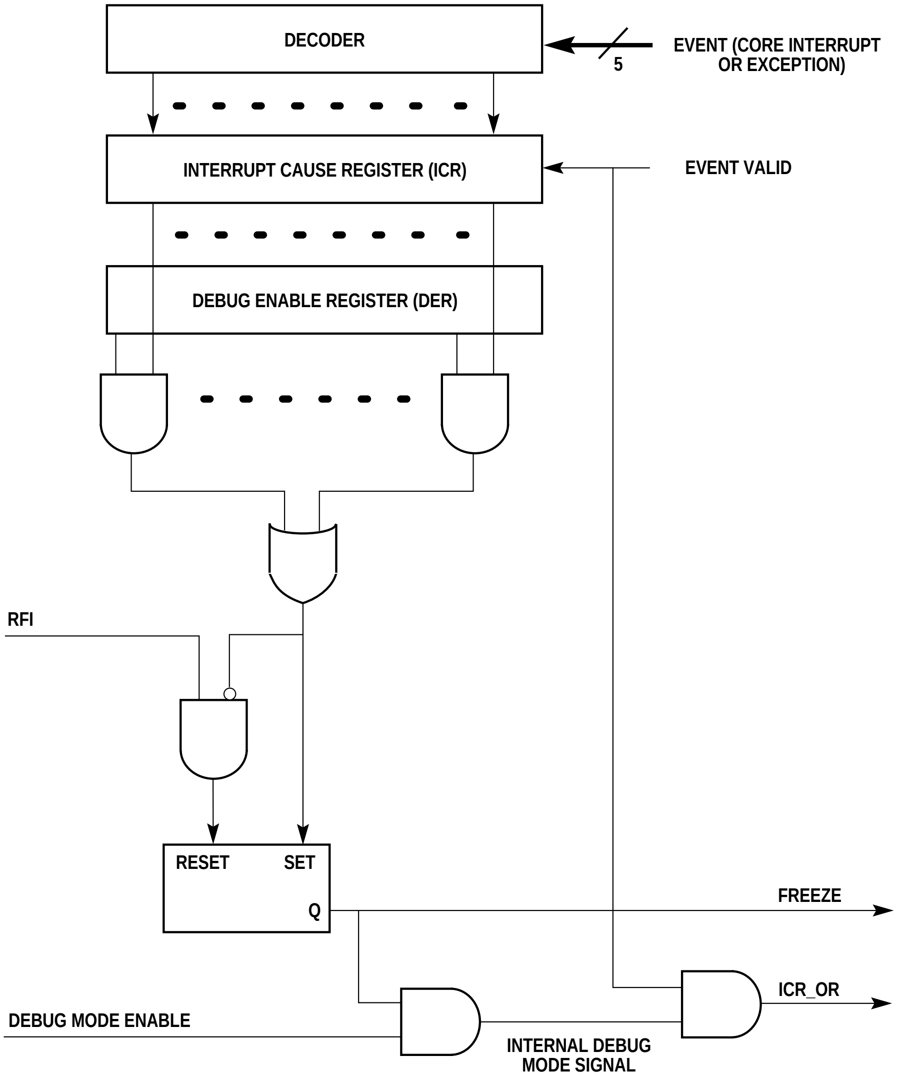

DECODER
5
EVENT (CORE INTERRUPT OR EXCEPTION)
INTERRUPT CAUSE REGISTER (ICR)
EVENT VALID
DEBUG ENABLE REGISTER (DER)
RFI
RESET
SET
Q
FREEZE
ICR_OR
DEBUG MODE ENABLE
INTERNAL DEBUG MODE SIGNAL

**Figure 19-6. Debug Mode Logic Diagram**

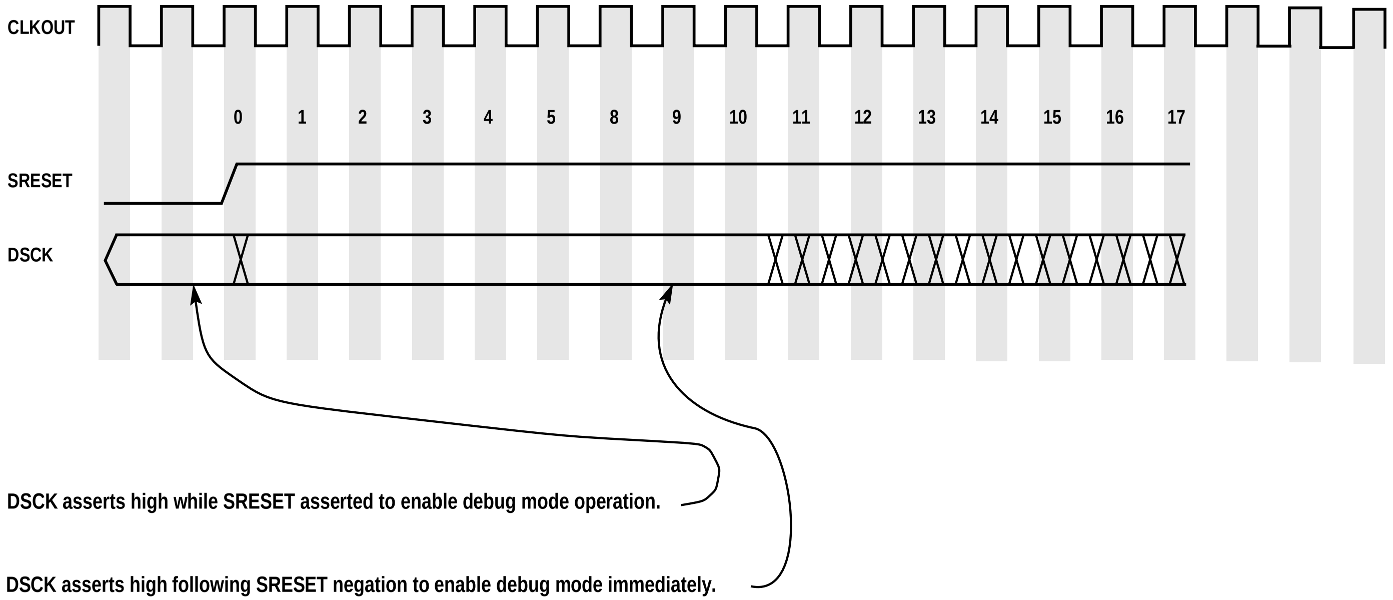
**19.3.1.2 DEBUG MODE ENABLE VS. DEBUG MODE DISABLE.** For protection purposes, there are two possible working modes—debug mode enable and debug mode disable. These working modes are selected once at reset. Debug mode is enabled by asserting the DSCK pin during reset. The state of this pin is sampled three clocks before the negation of $\overline{\text{SRESET}}$. If the DSCK pin is sampled negated, debug mode is disabled until a subsequent reset when the DSCK pin is asserted. When debug mode is disabled, the internal watchpoint/breakpoint hardware will still be operational and can be used by a software monitor program for debugging purposes. A timing diagram for the enabling debug mode is illustrated in Figure 19-7.

**NOTE**

> Since $\overline{\text{SRESET}}$ negation time is dependent on an external pull-up resistor. Any reference to $\overline{\text{SRESET}}$ negation time in this chapter refers to the time the MPC821 releases $\overline{\text{SRESET}}$. If the rise time of $\overline{\text{SRESET}}$ is long because of a large resistor, the setup time for the debug port signals should be adjusted accordingly.

When in debug mode disabled, all development support registers are accessible when $\text{MSR}_{\text{PR}}$ =0 and can be used by monitor debugger software. However, the processor never enters debug mode and the ICR and DER are only used for asserting and negating the freeze signal. For more information on the software monitor debugger support, refer to **Section 19.4 Software Monitor Debugger Support**. Only when the CPU is in debug mode, are all development support registers accessible. Therefore, the development system has full control of the CPU’s development support features. For more information, see Table 19-13.

**19.3.1.3 ENTERING DEBUG MODE.** Debug mode entry can be the result of a number of events. All events have a programmable enable bit so the user can selectively decide what events cause debug mode entry and what events require regular interrupt handling. Entering debug mode is also possible immediately out of reset, thus allowing the user to debug a ROM-less system. This is possible by specially programming the development port during reset. If the DSCK pin is asserted throughout $\overline{\text{SRESET}}$ assertion and then past $\overline{\text{SRESET}}$ negation, the processor will take a breakpoint exception and go directly to debug mode instead of fetching the reset vector.



DSCK asserts high while SRESET asserted to enable debug mode operation.

DSCK asserts high following SRESET negation to enable debug mode immediately.

Figure 19-7. Debug Mode Reset Configuration Timing Diagram

To avoid entering debug mode following reset, the DSCK pin must be negated no later than seven clock cycles after $\overline{\text{SRESET}}$ negates, thus allowing the processor to jumps to the reset vector and begin normal execution. Entering debug mode immediately after reset, Bit 31 (development port interrupt bit) of ICR is set. For details, refer to the timing diagram illustrated in Figure 19-7.

When debug mode is disabled, all events result in regular interrupt handling. The internal freeze signal is asserted whenever an enabled event occurs, regardless of whether or not debug mode is enabled or disabled. The internal freeze signal is connected to all relevant internal modules. These modules can be programmed to stop all operations in response to the assertion of the freeze signal. For more information, refer to **Section 19.4.1 Freeze Indication**. Furthermore, the freeze indication is negated when exiting the debug mode and **Section 19.3.1.7 Exiting Debug Mode** has more information on the issue.

The following list of events can cause the CPU to enter debug mode. Each event results in debug mode entry if debug mode is enabled and the corresponding enable bit is set. The reset values of the enable bits allow use of the debug mode features without programming the DER in most cases. For more information, see Table 19-24.

- System reset, as a result of the assertion of $\overline{\text{SRESET}}$. For details, refer to **Section 6.4.1.3.1 System Reset Interrupt**.
- Check stop. See Table 19-7 for details.
- Machine check interrupt.
- Implementation specific instruction TLB miss.
- Implementation specific instruction TLB error.
- Implementation specific data TLB miss.
- Implementation specific data TLB error.
- External interrupt, recognized when $\text{MSR}_{\text{EE}}$ =1.
- Alignment interrupt.
- Program interrupt.
- Floating-point unavailable interrupt.
- Decrementer interrupt, recognized when $\text{MSR}_{\text{EE}}$ =1.
- System call interrupt.
- Trace, asserted when in single or branch trace mode. For more information, refer to **Section 7.3.7.8 Trace Interrupt**.
- Implementation dependent software emulation interrupt.
- Instruction breakpoint is recognized only when $\text{MSR}_{\text{RI}}$ =1, when breakpoints are masked. When breakpoints are not masked, they are always recognized.

- Load/store breakpoint, when breakpoints are masked are recognized only when $MSR_{RI}$ =1. When breakpoints are not masked, they are always recognized.
- Peripherals breakpoint from the development port generated by external modules are recognized only when $MSR_{RI}$ =1.
- Development port nonmaskable interrupt, as a result of a debug station request. Useful in some catastrophic events like an endless loop when $MSR_{RI}$ =0. As a result of this event, the machine may enter a nonrestartable state. For more information, refer to **Section 6.3.5.1 Restartability After An Interrupt**.

The processor enters into the debug mode state when at least one of the bits in the ICR is set, the corresponding bit in the DER is enabled, and debug mode is enabled. When the debug mode is enabled and an enabled event occurs, the processor waits until it's pipeline is empty and then starts fetching the next instructions from the development port. For information on the exact value of the SRR0 and SRR1 registers, refer to **Section 7.3.7.3 Interrupt Definitions**. When the processor is in debug mode, the freeze indication is asserted, thus allowing any properly programmed peripheral to stop. The fact that the CPU is in debug mode is also broadcasted to the external world using the value b'11' on the VFLS pins. The freeze signal can be asserted by the software when debug mode is disabled. The development port should read the value of the ICR to get the cause of the debug mode entry. Reading the ICR clears all of it's bits.

**19.3.1.4 THE CHECKSTOP STATE AND DEBUG MODE.** The CPU enters the checkstop state if the machine check interrupt is disabled ($MSR_{ME}$ =0) and a machine check interrupt is detected. However, if a machine check interrupt is detected when $MSR_{ME}$ =0, debug mode is enabled and the checkstop enable bit in the DER is set, and the CPU enters debug mode rather then the checkstop state. The various actions taken by the CPU when a machine check interrupt is detected are provided in the following table.

**Table 19-7. Checkstop State and Debug Mode**

| $MSR_{ME}$ | DEBUG MODE ENABLE | CHSTPE[2] | MCIE[2] | ACTION PERFORMED BY THE CPU WHEN DETECTING A MACHINE CHECK INTERRUPT | INTERRUPT CAUSE REGISTER (ICR) VALUE |
|---|---|---|---|---|---|
| 0 | 0 | X | X | Enter the Checkstop State | 0x20000000 |
| 1 | 0 | X | X | Branch to the Machine Check Interrupt | 0x10000000 |
| 0 | 1 | 0 | X | Enter the Checkstop State | 0x20000000 |
| 0 | 1 | 1 | X | Enter Debug Mode | 0x20000000 |
| 1 | 1 | X | 0 | Branch to the Machine Check Interrupt | 0x10000000 |
| 1 | 1 | X | 1 | Enter Debug Mode | 0x10000000 |

NOTES: 1. Checkstop enable bit in the DER.

2. Machine check interrupt enable bit in the DER.

**19.3.1.5 SAVING MACHINE STATE WHEN ENTERING DEBUG MODE.** If entering debug mode is the result of any load/store-type exception, the DAR and DSISR contain critical information. These two registers must be saved before any other operation is performed. Failing to save these registers can result in information loss if another load/store-type exception occurs inside the development software. Since exceptions are treated differently in debug mode, there is no need to save SRR0 and SRR1 registers.

**19.3.1.6 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 a special control register, therefore, it is accessed using the mtspr and mfspr instructions, via special load/store cycles.

Exceptions are treated differently in debug mode. When already in debug mode the ICR is updated on recognition of an exception according to the event that caused the exception. A special error indication (ICR_OR) is asserted for one clock cycle to notify the development port that an exception occurred. Execution then continues in debug mode without any change in SRR0 and SRR1 registers. ICR_OR is asserted before the next fetch occurs to allow the development system to detect the excepting instruction. Not all exceptions are recognized when in debug mode. Breakpoints and watchpoints are not generated by the hardware when in debug mode, regardless of the value of the $MSR_{RI}$ bit. On entering debug mode, the $MSR_{EE}$ bit is cleared by the hardware, thus forcing the hardware to ignore external and decrementer interrupts.

**CAUTION**

> Setting the $MSR_{EE}$ bit while in the debug mode through the debug software is strictly forbidden.

The reason for this restriction is that the external interrupt event is a level signal. Because the CPU only reports exceptions in debug mode and does not perform exception processing, the CPU hardware does not clear the $MSR_{EE}$ bit. This event, if enabled, is then recognized on every clock. When the ICR_OR signal is asserted the development station should investigate the ICR to find what event caused the exception. Since the values in registers SRR0 and SRR1 registers do not change if an exception is recognized when in debug mode, they only change once when entering debug mode. However, saving SRR0 and SRR1 when entering debug mode is unnecessary.

**19.3.1.7 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 freeze (FRZ) line or status to make sure the MPC821 is out of debug mode. It is the responsibility of the software to read the ICR before performing the rfi instruction. Failing to do so will force the CPU to immediately reenter debug mode and to reassert the freeze indication if an asserted bit in the ICR has a corresponding enable bit set in the DER.

## 19.3.2 Development Port

The development port provides a full duplex serial interface for communications between the internal development support logic, including debug mode and an external development tool. The relationship of the development support logic to the rest of the CPU chip is illustrated in Figure 19-5. Notice that the development port support logic is shown as a separate block for clarity. It will be implemented as part of the SIU module.

**19.3.2.1 DEVELOPMENT PORT PINS.** The following development port pin functions are provided:

- Development serial clock
- Development serial data in
- Development serial data out
- Freeze

**19.3.2.1.1 Development Serial Clock.** 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 most-significant bit of the shift register is presented at the DSDO pin. In all further discussions, references to the DSCK signal imply the internal synchronized value of the clock. The DSCK input must be driven either high or low at all times and not be allowed to float. A typical target environment would pull this input low with a resistor.

The clock may be implemented as a free-running or gated clock. As discussed in **Section 19.3.2.4 Development Port Serial Communications–Trap Enable Mode** and **Section 19.3.2.5 Development Port Serial Communications–Debug Mode**, the shifting of data is controlled by the ready and start signals, so the clock does not need to be gated with the serial transmissions. The DSCK pin is used at reset to enable debug mode and either immediately following reset or for event-driven entry into debug mode.

**19.3.2.1.2 Development Serial Data In.** Data to be transferred into the development port shift register is presented at the development serial data in (DSDI) pin by external logic. When driven asynchronous with the system clock, the data presented to the DSDI pin must be stable at setup time before the rising edge of DSCK and at hold time after the rising edge of DSCK. When driven synchronous to the system clock, the data must be stable on DSDI or a setup time before system clock output (CLKOUT) rising edge and a hold time after the rising edge of CLKOUT. The DSDI pin is also used at reset to control the overall chip configuration mode and to determine the development port clock mode. Refer to **Section 19.3.2.3 Development Port Serial Communications–Clock Mode Selection** for more information.

**19.3.2.1.3 Development Serial Data Out.** The debug mode logic shifts data out of the development port shift register using the development serial data out (DSDO) pin. All transitions on DSDO are synchronous with DSCK or CLKOUT, depending on the clock mode. Data will be valid at setup time before the rising edge of the clock and remains valid at hold time after the rising edge of the clock. See Table 19-10 for DSDO data meaning.

**19.3.2.1.4 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 the FRZ pin and is generated synchronous to the system clock. This indication can be used to halt any off-chip device while in debug mode and is a handshake between the debug tool and port. In addition to the FRZ pin, the freeze state is indicated by the value b11 on the VFLS[0:1] pins. The internal freeze status can also be monitored through status in the data shifted out of the debug port.

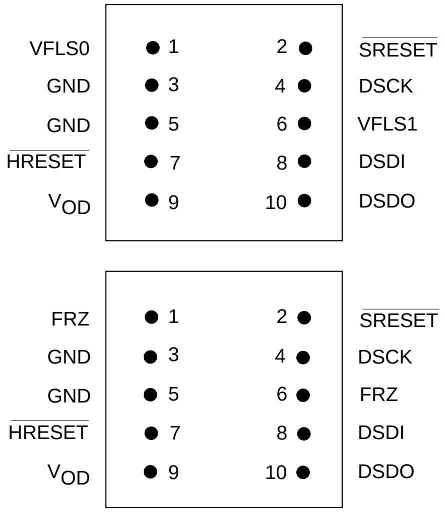

VFLS0
GND
GND
$\overline{\text{HRESET}}$
$V_{OD}$
1
3
5
7
9
2
4
6
8
10
$\overline{\text{SRESET}}$
DSCK
VFLS1
DSDI
DSDO
FRZ
GND
GND
$\overline{\text{HRESET}}$
$V_{OD}$
1
3
5
7
9
2
4
6
8
10
$\overline{\text{SRESET}}$
DSCK
FRZ
DSDI
DSDO

**Figure 19-8. Development Port/BDM Connector Pinout Options**

**19.3.2.2 DEVELOPMENT PORT REGISTERS.** The development port consists logically of three registers—development port instruction register (DPIR), development port data register (DPDR), and trap enable control register (TECR). These registers are physically implemented as two registers—the development port shift register and the TECR. The development port shift register acts as both the DPIR and DPDR, depending on the operation being performed. It is also used as a temporary holding register for data to be stored in the TECR.

**19.3.2.2.1 Development Port Shift Register.** The development port shift register is a 35-bit shift register. Instructions and data are serially shifted into it from the DSDI using either DSCK or CLKOUT as the shift clock, depending on the debug port clock mode. For more information refer to **Section 19.3.2.3 Development Port Serial Communications–Clock Mode Selection**.

The instructions or data are then transferred in parallel to the CPU and TECR. When the processor enters debug mode it fetches instructions from the DPIR that causes 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. In a similar way, data is transferred to the CPU. Data is shifted into the shift register and read by the processor as the result of executing a “move from special purpose register DPDR” instruction. Data is also parallel loaded into the development port shift register from the CPU core by executing a “move to special purpose register DPDR” instruction. It is then serially shifted out to DSDO using DSCK or CLKOUT as the shift clock.

**19.3.2.2.2 Trap Enable Control Register.** The trap enable control register is a 9-bit register that is loaded from the development port shift register. The contents of the control register drives the six trap enable signals, two breakpoint signals, and VSYNC signal to the CPU core. The Transfer Data to Trap Enable Control Register commands cause the appropriate bits to be transferred to the control register.

The trap enable control register is not accessed by the CPU, but supplies signals to the CPU 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 the ICTRL and LCTRL2 special registers. Refer to **Section 19.5.3 Development Support Registers Description** for more information on the support registers.

**19.3.2.2.3 Development Port Registers Decode.** The development port shift register is selected when the CPU accesses DPIR or DPDR. Accesses to these two special purpose registers occur in debug mode and appear on the internal bus as an address and the assertion of an address attribute signal indicating that a special purpose register is being accessed. The DPIR is read by the CPU core to fetch all instructions when in debug mode and the DPDR is read and written to transfer data between the CPU core and external development tools. The DPIR and DPDR are pseudo-registers. Decoding either of these registers causes the development port shift register to be accessed. The debug mode logic knows whether the CPU core is fetching instructions or reading or writing data. A sequence error is signaled to the external development tool when the CPU expected result and the GPR result do not match. An example of this would be when a instruction is received when data is what is expected.

**19.3.2.3 DEVELOPMENT PORT SERIAL COMMUNICATIONS–CLOCK MODE SELECTION.** All of the serial transmissions are clocked transmissions and are synchronous communications. The transmission clock may either be synchronous or asynchronous with CLKOUT. The development port allows the user to select two methods for clocking the serial transmissions. The first method allows the transmission to occur without being externally synchronized to CLKOUT. In this mode, a serial clock DSCK must be supplied to the MPC821. The other communication method requires a data to be externally synchronized with CLKOUT.

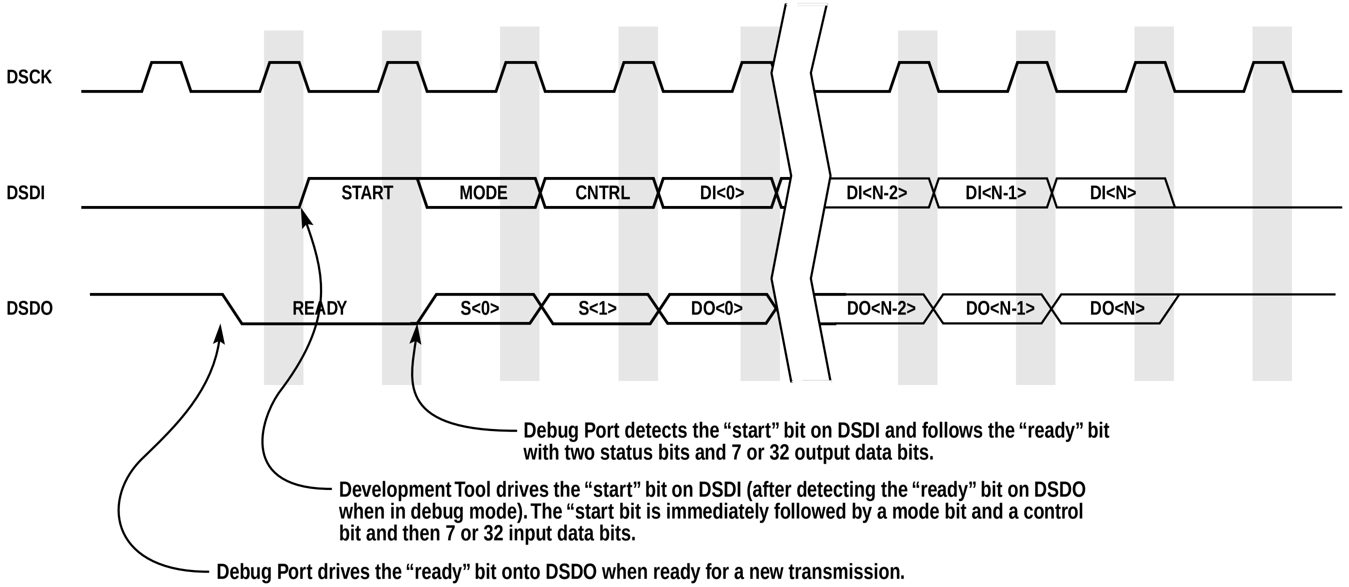
The first clock mode is called asynchronous clocked since the input clock DSCK is asynchronous with respect to CLKOUT. To be sure that data on DSDI is sampled correctly, transitions on DSDI must meet all setup and hold times in respect to the rising edge of DSCK. This clock mode allows communications with the port from a development tool which does not have access to the CLKOUT signal or where the CLKOUT signal has been delayed or skewed. The timing diagram in Figure 19-9 illustrates the serial communications asynchronous clocked timing.

The second clock mode is called synchronous self-clocked and does not require an input clock. Instead, the port is clocked by the system clock. The DSDI input is required to meet setup and hold time requirements, with respect to CLKOUT rising edge. The data rate for this mode is always the same as the system clock. The timing diagram in Figure 19-10 illustrates the serial communications synchronous self-clocked timing. The selection of clocked or self-clocked mode is made at reset. The state of the DSDI input is latched eight clocks after 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 19-11 illustrates the clock mode selection following 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 from being recognized as the start of a serial transmission. The port will not begin scanning for the start bit of a serial transmission until 16 clocks after 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.

**19.3.2.4 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.

**19.3.2.4.1 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 seven least-significant data bits. The encoding of data shifted into the development port shift register (through the DSDI pin) is shown in Table 19-8 and Table 19-9.



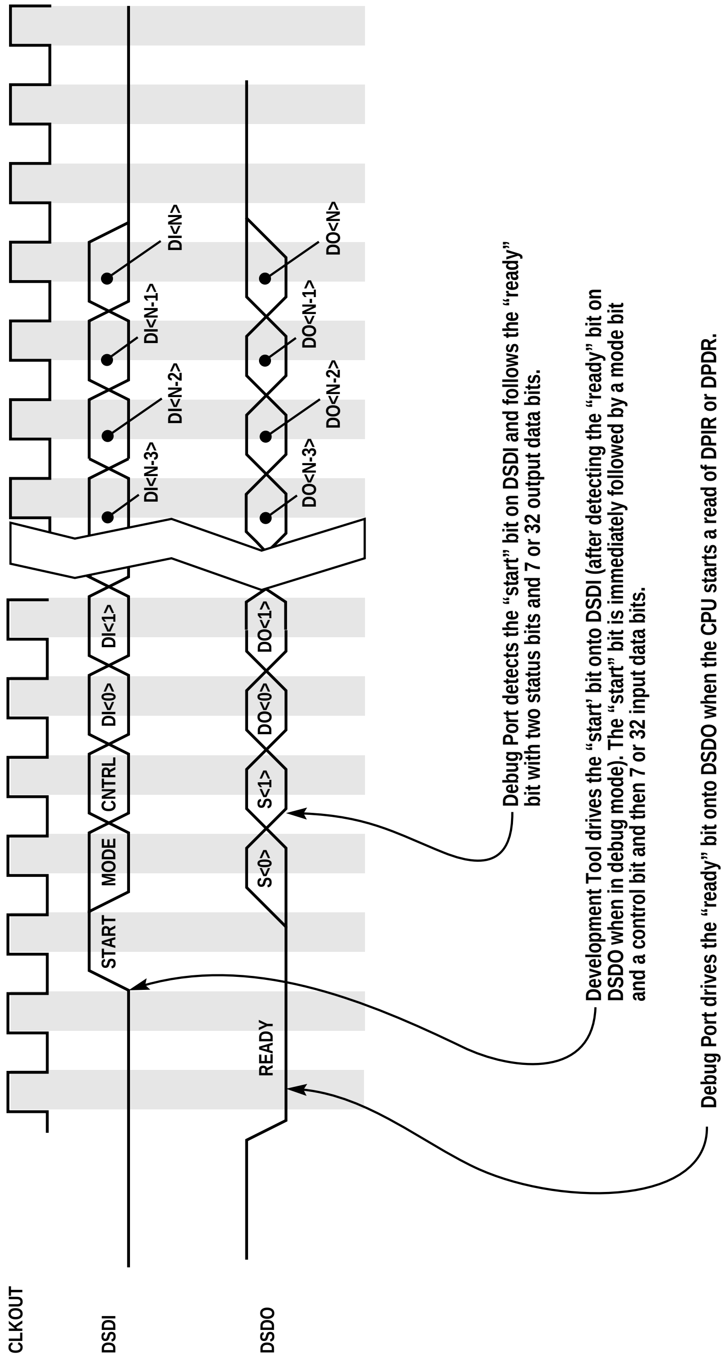
Debug Port detects the “start” bit on DSDI and follows the “ready” bit with two status bits and 7 or 32 output data bits.

Development Tool drives the “start” bit on DSDI (after detecting the “ready” bit on DSDO when in debug mode). The “start bit is immediately followed by a mode bit and a control bit and then 7 or 32 input data bits.

Debug Port drives the “ready” bit onto DSDO when ready for a new transmission.

NOTE: DSCK and DSDI transitions are not required to be synchronous with CLKOUT.

Figure 19-9. Asynchronous Clocked Serial Communications Timing Diagram

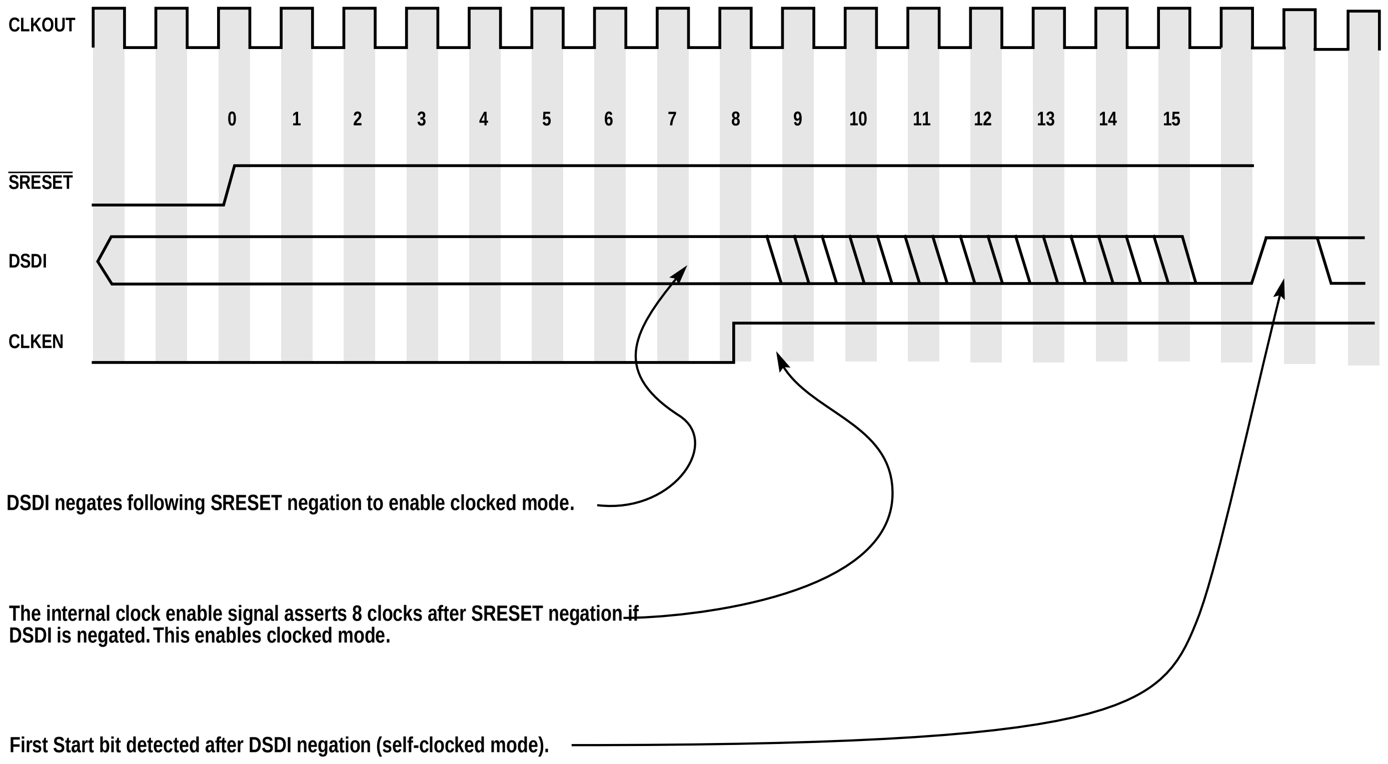


Debug Port detects the “start” bit on DSDI and follows the “ready” bit with two status bits and 7 or 32 output data bits.

Development Tool drives the “start’ bit onto DSDI (after detecting the “ready” bit on DSDO when in debug mode). The “start” bit is immediately followed by a mode bit and a control bit and then 7 or 32 input data bits.

Debug Port drives the “ready” bit onto DSDO when the CPU starts a read of DPIR or DPDR.

Figure 19-10. Synchronous Self-Clocked Serial Communications Timing Diagram



DSDI negates following SRESET negation to enable clocked mode.

The internal clock enable signal asserts 8 clocks after SRESET negation if DSDI is negated. This enables clocked mode.

First Start bit detected after DSDI negation (self-clocked mode).

Figure 19-11. Enabling Clock Mode Following Reset Timing Diagram

### Table 19-8. 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<br>1 = Enabled | | | | | | | Transfer Data to Trap Enable Control Register |

### Table 19-9. 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 | Hard Reset Request |
| | | | | | 00010 | Soft Reset Request |
| | | | 0 | x | 00011 | Reserved |
| | | | 1 | 0 | 00011 | End Download Procedure |
| | | | 1 | 1 | 00011 | Start Download Procedure |
| | | | x | x | 00100 — 11110 | Reserved |
| | | | x | 0 | 11111 | Negate Maskable Breakpoint |
| | | | x | 1 | 11111 | Assert Maskable Breakpoint |
| | | | 0 | x | 11111 | Negate Nonmaskable Breakpoint |
| | | | 1 | x | 11111 | Assert Nonmaskable Breakpoint |

The watchpoint trap enables and VSYNC functions are described in **Section 19.2 Watchpoints And Breakpoints Support** and **Section 19.1 Program Flow Tracking**. 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.

**19.3.2.4.2 Serial Data Out of Development Port.** In trap enable mode there is no data out of the development port. Data out of the development port in the trap enable mode is shown in Table 19-10.

Table 19-10. 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<br>DEPENDING ON THE INPUT MODE | |
| (0) | 0 | 0 | DATA | | | Valid Data From CPU |
| (0) | 0 | 1 | Freeze Status | Download Procedure In Progress | 1s | Sequencing Error |
| (0) | 1 | 0 | | | 1s | CPU Interrupt |
| (0) | 1 | 1 | | | 1s | Null |

NOTES: 1. The freeze status is set to 1 when the core is in debug mode. Otherwise it is set to 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.

When in trap enable mode the “Valid Data from CPU” and “CPU Interrupt” status cannot occur. 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 operation (NOP) function. The null output encoding is used to indicate that the previous transmission did not have any associated errors. When not in debug mode, ready will be 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.

**19.3.2.5 DEVELOPMENT PORT SERIAL COMMUNICATIONS–DEBUG MODE.** When in debug mode the development port starts communications by setting DSDO low to indicate that the CPU is trying to read an instruction from DPIR or data from DPDR. When the CPU writes data to the port to be shifted out the ready bit is not set. The port waits for the CPU to read the next instruction before asserting ready. This allows duplex operation of the serial port while allowing the port to control all transmissions from the external development tool. After detecting this ready status the external development tool begins the transmission to the development port with a start bit (logic high) on the DSDI pin.

**19.3.2.5.1 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 CPU are transmitted with the mode bit cleared indicating a 32-bit data field. The encoding of data shifted into the development port shift register through the DSDI pin is shown in Table 19-11. Data values in the last two functions other than those specified are reserved.

**Table 19-11. 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 | CPU Instruction | | Transfer Instruction to CPU |
| 1 | 0 | 1 | CPU Data | | Transfer Data to CPU |
| 1 | 1 | 0 | Trap Enable Bits | Not Exist | Transfer Data to Trap Enable Control Register |
| 1 | 1 | 1 | 0011111 | Not Exist | Negate Breakpoint Requests to Core |
| 1 | 1 | 1 | 0 | Not Exist | NOP |

NOTE: See Table 19-8 for details on trap enable bits.

All transmissions from the debug port on DSDO begin with a zero or ready bit. This indicates that the CPU is trying to read an instruction or data from the port. The external development tool must wait until it sees DSDO go low to begin sending the next transmission. The control bit differentiates between instructions and data allowing the development port to detect that an instruction was entered when the CPU 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.

**19.3.2.5.2 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 and is shown in Table 19-10. The valid data encoding is used when data has been transferred from the CPU to the development port shift register. This is the result of an instruction to move the contents of a general-purpose register to the DPDR. The valid data encoding has the highest priority of all status outputs and will be reported even if an interrupt occurs at the same time. Since it is not possible for a sequencing error to occur and also have valid data, there is no priority conflict with the sequencing error status. Also, any interrupt that is recognized at the same time that there is valid data, is not related to the execution of an instruction. Therefore, a valid data status will be output and the interrupt status will be saved for the next transmission.

The sequencing error encoding indicates that the inputs from the external development tool are not what the development port and/or the CPU was expecting. There are two possibilities for the cause of 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 CPU to signal that an interrupt exception has occurred. Since a status of sequencing error is of higher priority than an exception, the port reports the sequencing error first and the CPU interrupt on the next transmission. The development port ignores the command, instruction, or data shifted in while the sequencing error or CPU 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 CPU 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 has been transferred from the CPU to the development port shift register.

**19.3.2.5.3 Fast Download Procedure.** The fast download procedure is used to download a block of data from the debug tool into the system memory. This procedure can be accomplished by repeating the following sequence of transactions from the development tool to the debug port for the number of data words to be downloaded.

```
INIT:       Save RX, RY
            RY <- Memory Block address- 4

            •••

repeat:     mfspr    RX, DPDR
            DATA word to be moved to memory
            stwu     RX, 0x4(RY)
until here

            •••

            Restore RX,RY
```

**Figure 19-12. Download Procedure Code Example**

For large blocks of data this sequence may take a significant amount of time to complete. The fast download procedure of the debug port may be used to reduce this time. This time reduction is achieved by eliminating the need to transfer the instructions in the loop to the debug port. The only transactions needed are those required to transfer the data to be placed in the system memory. Figure 19-13 and Figure 19-14 illustrate the time benefit of the fast download procedure.

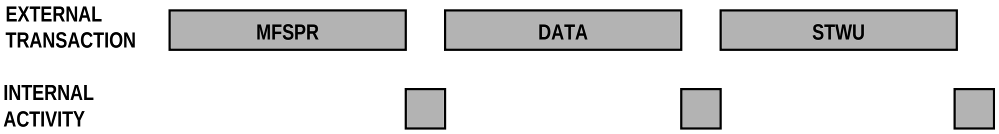


**Figure 19-13. Slow Download Procedure Loop**

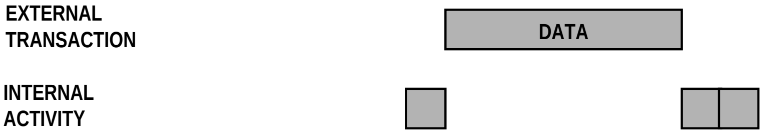


**Figure 19-14. Fast Download Procedure Loop**

The sequence of the instructions used in the fast download procedure is the one illustrated in Figure 19-12, with RX = r31 and RY = r30. This sequence is repeated infinitely until the end download procedure command is issued to the debug port. The internal general-purpose register 31 is used for temporary storage of the data value. Before beginning the fast download procedure by the start download procedure command, the value of the first memory block address -4 must be written into the general-purpose register 30. To end the download procedure, an end download procedure command should be issued to the debug port, and then an additional data transaction should be sent by the development tool. This data word will not be placed into the system memory, but it is needed to stop the procedure.

## 19.4 SOFTWARE MONITOR DEBUGGER SUPPORT

When in debug mode disable, a software monitor debugger can make use of all of the development support features defined in the CPU. When debug mode is disabled all events result in regular interrupt handling (the processor resumes execution in the corresponding interrupt handler). The ICR and DER only influence the assertion and negation of the freeze signal.

### 19.4.1 Freeze Indication

The internal freeze signal is connected to all relevant internal modules. These modules can be programmed to stop all operations in response to the assertion of the freeze signal. To enable a software monitor debugger to broadcast the fact that the debug software is now executed, it is possible to assert and negate the internal freeze signal when debug mode is disabled.

The assertion of the freeze signal is broadcasted to the external world over the FRZ line. The assertion and negation of the freeze signal when in debug mode disable is controlled by the ICR and DER, as illustrated in Figure 19-6.

To assert the freeze signal, the software needs to program the relevant bits in the DER, but to negate the freeze line, the software needs to read the ICR to clear it and perform an rfi instruction. If the ICR is not cleared before the rfi instruction is performed, 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, although rfi may be performed a few times. Only before the last rfi instruction does the software need to clear the ICR. The above mechanism enables the software to accurately control the assertion and negation of the freeze line.

## 19.5 DEVELOPMENT SUPPORT PROGRAMMING MODEL

### 19.5.1 Development Support Registers List

Table 19-12. Development Support Registers

| REGISTER NAME | MNEMONIC |
|---|---|
| Comparator A Value Register | CMPA |
| Comparator B Value Register | CMPB |
| Comparator C Value Register | CMPC |
| Comparator D Value Register | CMPD |
| Comparator E Value Register | CMPE |
| Comparator F Value Register | CMPF |
| Comparator G Value Register | CMPG |
| Comparator H Value Register | CMPH |
| Breakpoint Address Register | BAR |
| Instruction Support Control Register | ICTRL |
| Load/Store Support Comparators Control Register | LCTRL1 |
| Load/Store Support AND-OR Control Register | LCTRL2 |
| Breakpoint Counter A Value and Control Register | COUNTA |
| Breakpoint Counter B Value and Control Register | COUNTB |
| Interrupt Cause Register | ICR |
| Debug Enable Register | DER |
| Development Port Data Register | DPDR |

These registers reside in the control registers space and can be accessed using the mtspr and mfspr instructions. The addresses of these registers are in Table 6-9.

### 19.5.2 Development Support Registers Protection

The development support registers are protected according to the following table. Notice the special behavior of the ICR and DPDR.

Table 19-13. Development Support Registers Protection

| OPERATION | $MSR_{PR}$ | DEBUG MODE ENABLE | IN DEBUG MODE | RESULT |
|---|---|---|---|---|
| Read Register | 0 | 0 | X | Read is Performed,<br>When Reading ICR, It is Also Cleared. |
| | 0 | 1 | 0 | Read is Performed,<br>When Reading ICR, It is Not Cleared. |
| | 0 | 1 | 1 | Read is Performed,<br>When Reading ICR, It is Also Cleared. |
| | 1 | X | X | Read is Not Performed,<br>Program Interrupt is Generated,<br>When Reading ICR, It is Not Cleared. |
| Write Register | 0 | 0 | X | Write is Performed,<br>Write to ICR is Ignored,<br>Write to DPDR is Ignored. |
| | 0 | 1 | 0 | Write is Ignored. |
| | 0 | 1 | 1 | Write is Performed,<br>Write to ICR is Ignored. |
| | 1 | X | X | Write is Not **P**erformed,<br>Program Interrupt is Generated. |

NOTE: Ignored means the register is not modified and no interrupt is generated.

### 19.5.3 Development Support Registers Description

Table 19-14. Comparator A-D Value Register(CMPA-D)

| BITS | MNEMONIC | DESCRIPTION |
|---|---|---|
| 0-29 | CMPV(0-29) | Address Bits to be Compared. |
| 30-31 | — | Reserved. |

NOTE: Reset value: undefined.

Table 19-15. Comparator E-F Value Register (CMPE-F)

| BITS | MNEMONIC | DESCRIPTION |
|---|---|---|
| 0-31 | CMPV(0-31) | Address Bits to be Compared. |

NOTE: Reset value: undefined.

### Table 19-16. Comparator G-H Value Register (CMPG-H)

| BITS | MNEMONIC | DESCRIPTION |
|---|---|---|
| 0-31 | CMPV(0-31) | Data Bits to be Compared. |

NOTE: Reset value: undefined.

### Table 19-17. Breakpoint Address Register (BAR)

| BITS | MNEMONIC | DESCRIPTION |
|---|---|---|
| 0-31 | BARV(0-31) | The Address of the Load/Store Cycle That Generated the Breakpoint. |

NOTE: Reset value: undefined.

### Table 19-18. Instruction Support Control Register (CTRL)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-2 | CTA(0:2) | Compare Type of Comparator A. | 0xx - Not Active (Reset Value)<br>100 - Equal<br>101 - Less Than<br>110 - Greater Than<br>111 - Not Equal |
| 3-5 | CTB(0:2) | Compare Type of Comparator B. | |
| 6-8 | CTC(0:2) | Compare Type of Comparator C. | |
| 9-11 | CTD(0:2) | Compare Type of Comparator D. | |
| 12-13 | IW0(0:1) | Instruction 1st Watchpoint Programming. | 0x - Not Active (Reset Value)<br>10 - Match From Comparator A<br>11 - Match From Comparators (A & B) |
| 14-15 | IW1(0:1) | Instruction 2nd Watchpoint Programming. | 0x - Not Active (Reset Value)<br>10 - Match From Comparator B<br>11 - Match From Comparators (A \| B) |
| 16-17 | IW2(0:1) | Instruction 3rd Watchpoint Programming. | 0x - Not Active (Reset Value)<br>10 - Match From Comparator C<br>11 - Match From Comparators (C & D) |
| 18-19 | IW3(0:1) | Instruction 4th Watchpoint Programming. | 0x - Not Active (Reset Value)<br>10 - Match From Comparator D<br>11 - Match From Comparators (C \| D) |

Table 19-18. Instruction Support Control Register (CTRL) (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 20 | SIW0EN | Software Trap Enable Selection of the 1st Instruction Watchpoint. | 0 - Trap Disabled (Reset Value)<br>1 - Trap Enabled |
| 21 | SIW1EN | Software Trap Enable Selection of the 2nd Instruction Watchpoint. | |
| 22 | SIW2EN | Software Trap Enable Selection of the 3rd Instruction Watchpoint. | |
| 23 | SIW3EN | Software Trap Enable Selection of the 4th Instruction Watchpoint. | |
| 24 | DIW0EN | Development Port Trap Enable Selection of the 1st Instruction Watchpoint (Read-Only Bit). | 0 - Trap Disabled (Reset Value)<br>1 - Trap Enabled |
| 25 | DIW1EN | Development Port Trap Enable Selection of the 2nd Instruction Watchpoint (Read-Only Bit). | |
| 26 | DIW2EN | Development Port Trap Enable Selection of the 3rd Instruction Watchpoint (Read-Only Bit). | |
| 27 | DIW3EN | Development Port Trap Enable Selection of the 4th Instruction Watchpoint (Read-Only Bit). | |
| 28 | IFM | Ignore First Match,<br>Only For Instruction Breakpoints. | 0 - Do Not Ignore First Match, Used For “Go To x” (Reset Value).<br>1 - Ignore First Match (Used For “Continue”). |

### Table 19-18. Instruction Support Control Register (CTRL) (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 29-31 | ISCT_SER[2] | Instruction Fetch Show Cycle and Core Serialize Control. | 000 - Core Is Fully Serialized and Show Cycle Will be Performed For All Fetched Instructions (Reset Value).[1]<br>001 - Core is Fully Serialized and Show Cycle Will be Performed For All Changes in the Program Flow.<br>010 - Core is Fully Serialized and Show Cycle Will be Performed For All Indirect Changes in the Program Flow.<br>011 - Core is Fully Serialized and No Show Cycles Will be Performed For Fetched Instructions.<br>100 - Illegal.<br>101 - Core is Not Serialized (Normal Mode) and Show Cycle Will be Performed For All Changes in the Program Flow.[3]<br>110 - Core is Not Serialized (Normal Mode) and Show Cycle Will be Performed For All Indirect Changes in the Program Flow.<br>111 - Core is Not Serialized (Normal Mode) and No Show Cycles Will be Performed For Fetched Instructions. |

NOTES:

1. Reset value: 0x00000000.
2. Changing the Instruction show cycle programming starts to take effect only from the 2nd instruction after the actual mtspr to ICTRL.
3. In case the fetch of the target of a direct branch is aborted by the core (because of an exceptions/interrupt), the target is not always visible on the external pins. Program trace is not affected by this phenomena.
4. When ICTRL(29:31) is set to 010 or 110, the $\overline{\text{STS}}$ functionality of the OP2/MODCK1/$\overline{\text{STS}}$ pin must be enabled by writing 10 or 11 to the DBGC field of the SIUMCR. The address on the external bus should only be sampled when $\overline{\text{STS}}$ is asserted.

### Table 19-19. Load/Store Support Comparators Control Register (LCTRL1)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-2 | CTE(0:2) | Compare Type, Comparator E | 0xx - Not Active (Reset Value)<br>100 - Equal<br>101 - Less Than<br>110 - Greater Than<br>111 - Not Equal |
| 3-5 | CTF(0:2) | Compare Type, Comparator F | |
| 6-8 | CTG(0:2) | Compare Type, Comparator G | |
| 9-11 | CTH(0:2) | Compare Type, Comparator H | |
| 12-13 | CRWE(0:1) | Select Match on Read/Write of Comparator E | 0x - Don't Care (Reset Value)<br>10 - Match On Read<br>11 - Match On Write |
| 14-15 | CRWF(0:1) | Select Match on Read/Write of Comparator F | |
| 16-17 | CSG(0:1) | Compare Size, Comparator G | 00 - Reserved<br>01 - Word<br>10 - Half-Word<br>11 - Byte |
| 18-19 | CSH(0:1) | Compare Size, Comparator H | |
| 20 | SUSG | Signed/Unsigned Operating Mode For Comparator G | 0 - Unsigned<br>1 - Signed |
| 21 | SUSH | Signed/Unsigned Operating Mode For Comparator H | |
| 22-25 | CGBMSK | Byte Mask For Comparator G | 0000 - All Bytes Are Not Masked<br>0001 - Last Byte of the Word is Masked<br>•<br>•<br>•<br>1111 - All Bytes Are Masked |
| 26-29 | CHBMSK | Byte Mask For Comparator H | |
| 30-31 | — | Reserved | |

NOTE: Reset value: 0x00000000.

### Table 19-20. Load/Store Support AND-OR Control Register (LCTRL2)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0 | LW0EN | 1st Load/Store Watchpoint Enable Bit. | 0 - Watchpoint Not Enabled (Reset Value)<br>1 - Watchpoint Enabled |
| 1-2 | LW0IA(0:1) | 1st Load/Store Watchpoint I-addr Watchpoint Selection. | 00 - First Instruction Watchpoint<br>01 - Second Instruction Watchpoint<br>10 - Third Instruction Watchpoint<br>11 - Fourth Instruction Watchpoint |
| 3 | LW0IADC | 1st Load/Store Watchpoint Care/Don't Care I-addr Events. | 0 - Don't Care<br>1 - Care |
| 4-5 | LW0LA(0:1) | 1st Load/Store Watchpoint L-addr Events Selection. | 00 - Match From Comparator E<br>01 - Match From Comparator F<br>10 - Match From Comparators (E & F)<br>11 - Match From Comparators (E \| F) |
| 6 | LW0LADC | 1st Load/Store Watchpoint Care/Don't Care L-addr Events. | 0 - Don't Care<br>1 - Care |
| 7-8 | LW0LD(0:1) | 1st Load/Store Watchpoint L-data Events Selection. | 00 - Match From Comparator G<br>01 - Match From Comparator H<br>10 - Match From Comparators (G & H)<br>11 - Match From Comparators (G \| H) |
| 9 | LW0LDDC | 1st Load/Store Watchpoint Care/Don't Care L-data Events. | 0 - Don't Care<br>1 - Care |
| 10 | LW1EN | 2nd Load/Store Watchpoint Enable Bit. | 0 - Watchpoint Not Enabled (Reset Value)<br>1 - Watchpoint Enabled |
| 11-12 | LW1IA(0:1) | 2nd Load/Store Watchpoint I-addr Watchpoint Selection. | 00 - First Instruction Watchpoint<br>01 - Second Instruction Watchpoint<br>10 - Third Instruction Watchpoint<br>11 - Fourth Instruction Watchpoint |
| 13 | LW1IADC | 2nd Load/Store Watchpoint Care/Don't Care I-addr Events. | 0 - Don't Care<br>1 - Care |
| 14 -15 | LW1LA(0:1) | 2nd Load/Store Watchpoint L-addr Events Selection. | 00 - Match From Comparator E<br>01 - Match From Comparator F<br>10 - Match From Comparators (E & F)<br>11 - Match From Comparators (E \| F) |
| 16 | LW1LADC | 2nd Load/Store Watchpoint Care/Don't Care L-addr Events. | 0 - Don't Care<br>1 - Care |
| 17 -18 | LW1LD(0:1) | 2nd Load/Store Watchpoint L-data Events Selection. | 00 - Match From Comparator G<br>01 - Match From Comparator H<br>10 - Match From Comparators (G & H)<br>11 - Match From Comparator (G \| H) |

### Table 19-20. Load/Store Support AND-OR Control Register (LCTRL2) (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 19 | LW1LDDC | 2nd Load/Store Watchpoint Care/Don't Care L-data Events. | 0 - Don't Care<br>1 - Care |
| 20 | BRKNOMSK | Internal Breakpoints Nonmask Bit Controls Both Instruction Breakpoints and Load/Store Breakpoints. | 0 - Masked Mode, Breakpoints Are Recognized Only When $MSR_{RI}$ =1 (Reset Value).<br>1 - Nonmasked Mode, Breakpoints Are Always Recognized. |
| 21 -27 | — | Reserved | |
| 28 | DLW0EN | Development Port Trap Enable Selection of the 1st Load/Store Watchpoint (Read-Only Bit). | 0 - Trap Disabled (Reset Value)<br>1 - Trap Enabled |
| 29 | DLW1EN | Development Port Trap Enable Selection of the 2nd Load/Store Watchpoint (Read-Only Bit). | |
| 30 | SLW0EN | Software Trap Enable Selection of the 1st Load/Store Watchpoint. | 0 - Trap Disabled (Reset Value)<br>1 - Trap Enabled |
| 31 | SLW1EN | Software Trap Enable Selection of the 2nd Load/Store Watchpoint. | |

NOTE: Reset value: 0x00000000.

Each watchpoint programming consists of three control register fields—LWxIA, LWxLA, and LWxLD. All three conditions must be detected to assert a watchpoint.

### Table 19-21. Breakpoint Counter A Value and Control Register (COUNTA)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-15 | CNTV(0:15) | Counter Preset Value | |
| 16-29 | — | Reserved | |
| 30-31 | CNTC(0:1) | Counter Source Select | 00 - Not Active (Reset Value)<br>01 - Instruction 1st Watchpoint<br>10 - Load/Store 1st Watchpoint<br>11 - Reserved |

NOTE: Reset value: 0-15 undefined, 16-31 0x0000.

### Table 19-22. Breakpoint Counter B Value and Control Register (COUNTB)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-15 | CNTV(0:15) | Counter Preset Value | |
| 16-29 | — | Reserved | |
| 30-31 | CNTC(0:1) | Counter Source Select | 00 - Not Active (Reset Value)<br>01 - Instruction 2nd Watchpoint<br>10 - Load/Store 2nd Watchpoint<br>11 - Reserved |

NOTE: Reset value: 0-15 undefined, 16-31 0x0000.

## 19.5.3.1 DEBUG MODE REGISTERS.

### Table 19-23. Interrupt Cause Register (ICR)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0 | — | Reserved | |
| 1 | RST | Reset Interrupt Bit | This Bit is Set When the System Reset Pin is Asserted. This Pin is Not Implemented in the CPU Chip. |
| 2 | CHSTP | Check Stop Bit | This Bit is 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 the Check Stop State. |
| 3 | MCI | Machine Check Interrupt Bit | This Bit is Set When the Machine Check Interrupt is Asserted and $MSR_{ME}$ =1. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 4-5 | — | Reserved | |
| 6 | EXTI | External Interrupt Bit | This Bit is Set When the External Interrupt is Asserted. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 7 | ALI | Alignment Interrupt Bit | This Bit is Set When the Alignment Interrupt is Asserted. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 8 | PRI | Program Interrupt Bit | This Bit is Set When the Program Interrupt is Asserted. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 9 | FPUVI | Floating-Point Unavailable Interrupt Bit | This Bit is Set When the Floating-Point Unavailable Interrupt is Asserted. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 10 | DECI | Decrementer Interrupt Bit | This Bit is Set When the Decrementer Interrupt is Asserted. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |

### Table 19-23. Interrupt Cause Register (ICR) (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 11-12 | — | Reserved | |
| 13 | SYSI | System Call Interrupt Bit | This Bit is Set When the System Call Interrupt is Asserted. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 14 | TR | Trace Interrupt Bit | This Bit is Set When in Single-Step Mode or When in Branch Trace Mode. Results in 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 | This Bit is Set When the Floating-Point Assist Interrupt is Asserted. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 18 | ITLBMS | Implementation Specific Instruction TLB Miss | This Bit is Set as a Result of an Instruction TLB Miss. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 19 | ITLBER | Implementation Specific Instruction TLB Error | This Bit is Set as a Result of an Instruction TLB Error. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 20 | DTLBMS | Implementation Specific Data TLB Miss | This Bit is Set as a Result of an Data TLB Miss. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 21 | DTLBER | Implementation Specific Data TLB Error | This Bit is Set as a Result of an Data TLB 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 | This Bit is Set as a Result of the Assertion of an Load/Store Breakpoint. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 29 | IBRK | Instruction Breakpoint Interrupt Bit | This Bit is Set as a Result of the Assertion of an Instruction Breakpoint. Results in 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) | This Bit is Set as a Result of the Assertion of an External Breakpoint. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |
| 31 | DPI | Development Port Interrupt Bit | This Bit is Set by the Development Port as a Result of a Debug Station Nonmaskable Request or When Entering Debug Mode Immediately Out of Reset. Results in Debug Mode Entry if Debug Mode is Enabled and the Corresponding Enable Bit is Set. |

NOTE: Reset value: 0x00000000.

This register provides the user with the reason for entering into the debug mode. All bits are set by the hardware and cleared when the register is read. Any attempt to write to this register will be ignored. All bits are cleared to zero when exiting reset.

**Table 19-24. Debug Enable Register (DER)**

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0 | — | Reserved | |
| 1 | RSTE | Reset Interrupt Enable Bit | 0 - Debug Mode Entry is Disabled (Reset Value)<br>1 - Debug Mode Entry is Enabled |
| 2 | CHSTPE | Check Stop Enable Bit | 0 - Debug Mode Entry is Disabled<br>1 - Debug Mode Entry is Enabled (Reset Value) |
| 3 | MCIE | Machine Check Interrupt Enable Bit | 0 - Debug Mode Entry is Disabled (Reset Value)<br>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<br>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)<br>1 - Debug Mode Entry is Enabled |
| 18 | ITLBMSE | Implementation Specific Instruction TLB Miss Enable Bit | |
| 19 | ITLBERE | Implementation Specific Instruction TLB Error Enable Bit | |
| 20 | DTLBMSE | Implementation Specific Data TLB Miss Enable Bit | |
| 21 | DTLBERE | Implementation Specific Data TLB Error Enable Bit | |

Table 19-24. Debug Enable Register (DER) (Continued)

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 22-27 | — | Reserved | |
| 28 | LBRKE | Load/Store Breakpoint Interrupt Enable Bit | 0 - Debug Mode Entry is Disabled<br>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 | |

NOTE: Reset value: 0x2002000.

This register enables the user to selectively enable the events that can cause the processor to enter into the debug mode.

**19.5.3.1.1 Development Port Data Register.** This 32-bit special purpose register physically resides in the development port logic. It is used for data interchange between the core and the development system. An access to this register is initiated using the mtspr and mfspr instructions and implemented using a special bus cycle on the internal bus. For details, see Table 6-9.

# SECTION 20
# IEEE 1149.1 TEST ACCESS PORT

The MPC821 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 proposed standard under the sponsorship of the Test Technology Committee of IEEE and the joint test action group (JTAG). The MPC821 implementation supports circuit-board test strategies based on this standard.

The TAP consists of five dedicated signal pins—a 16-state TAP controller and two test data registers. A boundary scan register links all device signal pins into a single shift register. The test logic, which is implemented using static logic design, is independent of the device system logic. The MPC821 implementation provides the capability to:

- Perform boundary scan operations to check circuit-board electrical continuity.
- Bypass the MPC821 for a given circuit-board test by effectively reducing the boundary scan register to a single cell.
- Sample the MPC821 system pins during operation and transparently shift out the result in the boundary scan register.
- Disable the output drive to pins during circuit-board testing.

**NOTE**

Certain precautions must be observed to ensure that the IEEE 1149.1-like test logic does not interfere with nontest operation. Refer to **Section 20.6 Nonscan Chain Operation** for details.

## 20.1 OVERVIEW

The MPC821 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). An overview of the MPC821 scan chain implementation is illustrated in the figure below. The TAP consists of the following signals:

- TCK—A test clock input to synchronize the test logic.
- TMS—A test mode select input (with an internal pull-up resistor) that is sampled on the rising edge of TCK to sequence the TAP controller's state machine.
- TDI—A test data input (with an internal pull-up resistor) that is sampled on the rising edge of TCK.
- TDO—A three-stateable test data output that is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCK.
- $\overline{\text{TRST}}$—An asynchronous reset with an internal pull-up resistor that provides initialization of the TAP controller and other logic required by the standard.

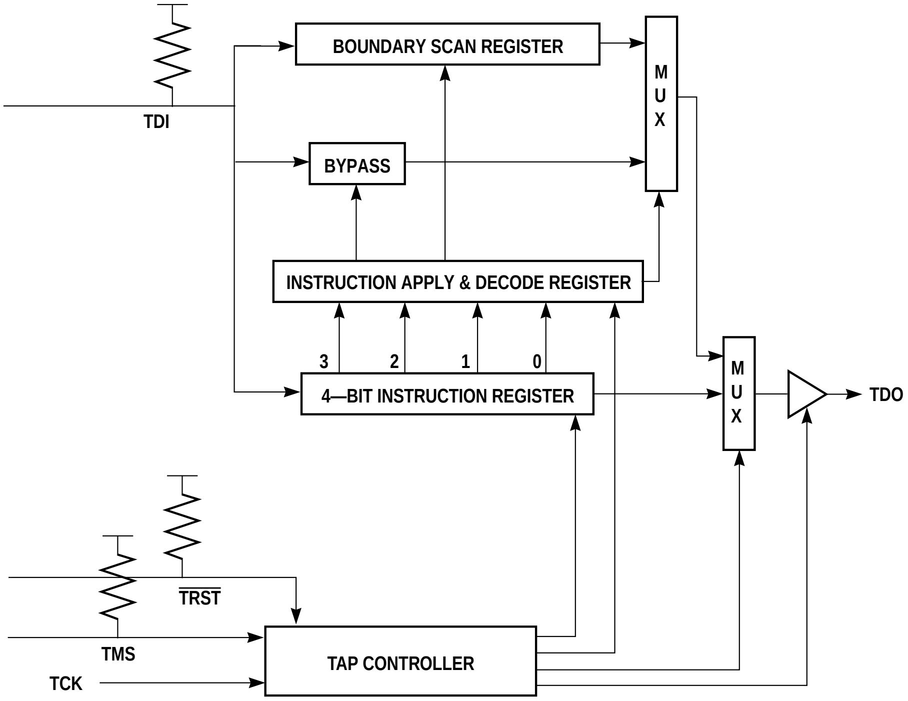


**Figure 20-1. Test Logic Block Diagram**

## 20.2 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 the TCK signal. An illustration of the state machine is provided in the figure below.

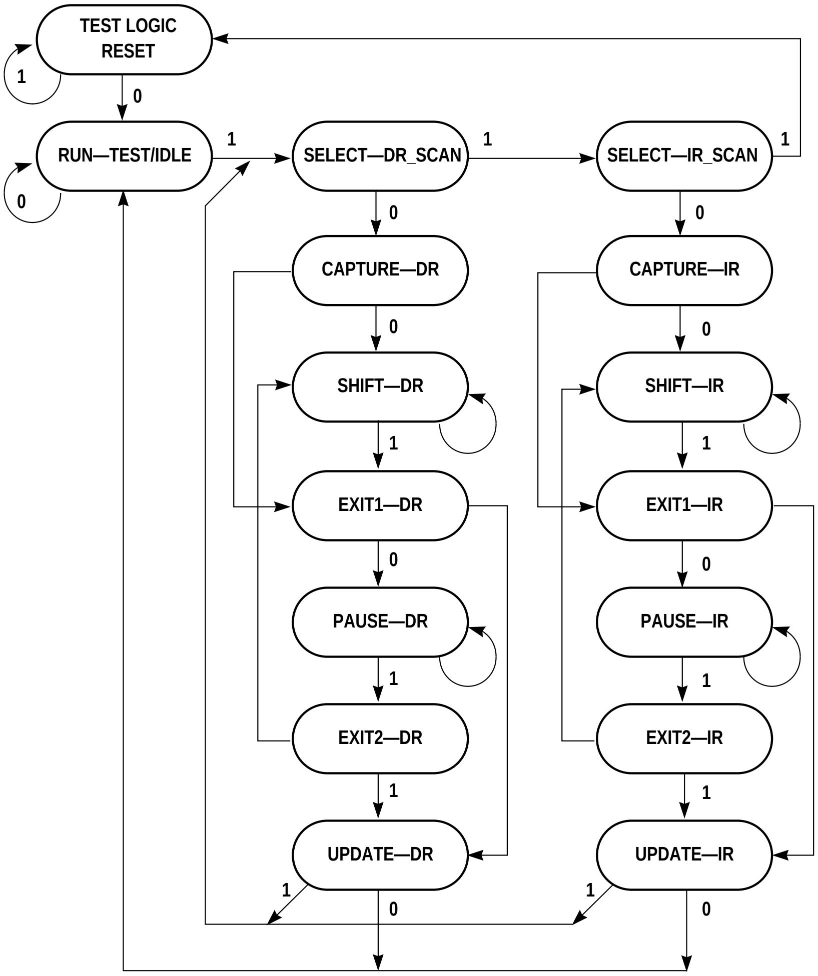


Figure 20-2. TAP Controller State Machine

## 20.3 BOUNDARY SCAN REGISTER

The MPC821 scan chain implementation has a 475-bit boundary scan register that contains bits for all device signal, clock pins, and associated control signals. The XTAL, EXTAL, and XFC pins are associated with analog signals and are not included in the boundary scan register. An IEEE-1149.1 compliant boundary scan register has been included on the MPC821. This 475-bit boundary scan register can be connected between TDI and TDO when EXTEST or SAMPLE/PRELOAD instructions are selected. It is used for capturing signal pin data on the input pins, forcing fixed values on the output signal pins, and selecting the direction and drive characteristics (a logic value or high impedance) of the bidirectional and three-state signal pins. Figure 20-3 through Figure 20-6 depict the various cell types.

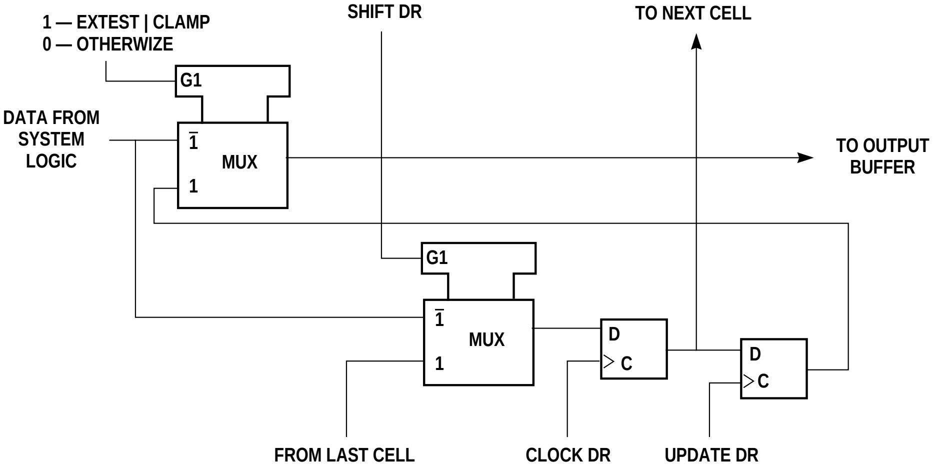


Figure 20-3. Output Pin Cell (O.Pin)

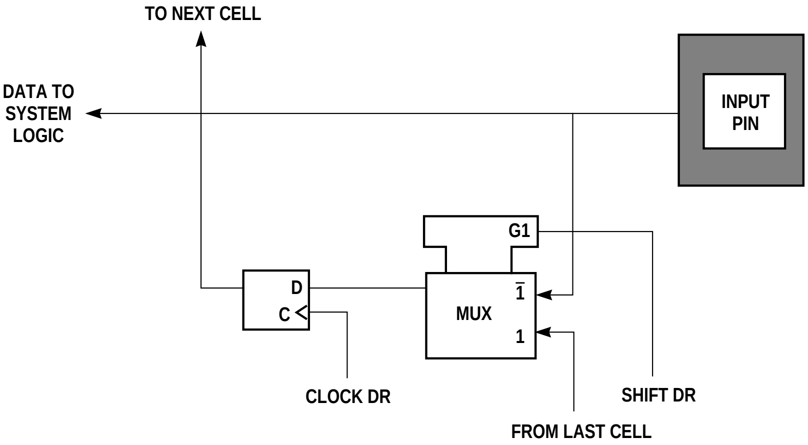


Figure 20-4. Observe-Only Input Pin Cell (I.Obs)

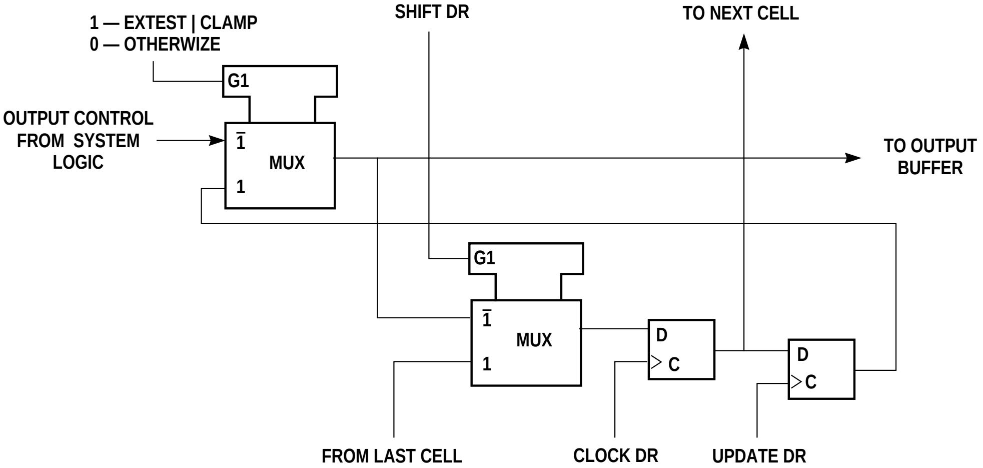


Figure 20-5. Output Control Cell (IO.CTL)

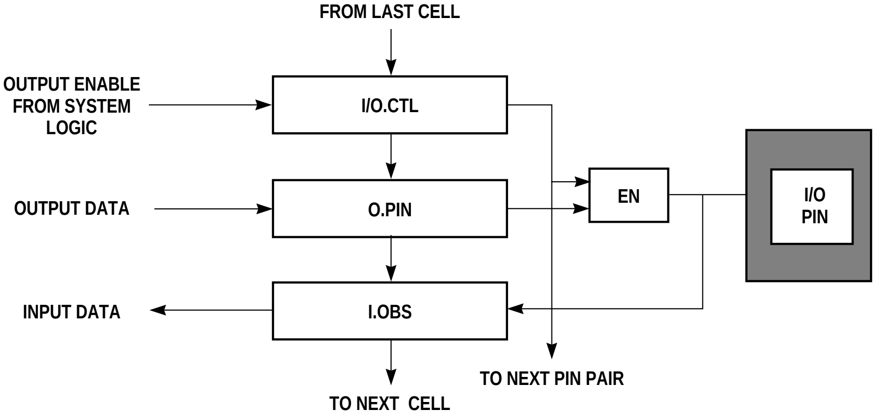


Figure 20-6. General Arrangement of Bidirectional Pin Cells

The value of the control bit controls the output function of the bidirectional pin. One or more bidirectional data cells can be serially connected to a control cell. Bidirectional pins include two scan cell for data (IO.Cell) as illustrated in Figure 20-6 and these bits are controlled by the cell illustrated in Figure 20-5.

It is important to know the boundary scan bit order and pins that are associated with them. The bit order starting with the TDO output and ending with the TDI input is shown in Table 20-1. The first column of the table defines the bit’s ordinal position in the boundary scan register. The shift register cell nearest TDO (first to be shifted in) is defined as Bit 1 and the last bit to be shifted in is Bit 475. The second column references one of the three MPC821 cell types depicted in Figure 20-3 through Figure 20-5 that describe the cell structure for each type. The third column lists the pin name for all pin-related cells and defines the name of the bidirectional control register bits. The fourth column lists the pin type and the last column indicates the associated boundary scan register control bit for the bidirectional output pins.

**Table 20-1. Boundary Scan Bit Definition**

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 1 | I.OBS | PA[11] | I/O | — |
| 2 | O.PIN | PA[11] | I/O | g73.ctl |
| 3 | IO.CTL | G73.CTL | — | — |
| 4 | I.OBS | PB[26] | I/O | — |
| 5 | O.PIN | PB[26] | I/O | g56.ctl |
| 6 | IO.CTL | G56.CTL | — | — |
| 7 | I.OBS | PC[12] | I/O | — |
| 8 | O.PIN | PC[12] | I/O | g34.ctl |
| 9 | IO.CTL | G34.CTL | — | — |
| 10 | I.OBS | PA[12] | I/O | — |
| 11 | O.PIN | PA[12] | I/O | g74.ctl |
| 12 | IO.CTL | G74.CTL | — | — |
| 13 | I.OBS | PB[27] | I/O | — |
| 14 | O.PIN | PB[27] | I/O | g57.ctl |
| 15 | IO.CTL | G57.CTL | — | — |
| 16 | I.OBS | PC[13] | I/O | — |
| 17 | O.PIN | PC[13] | I/O | g35.ctl |
| 18 | IO.CTL | G35.CTL | — | — |
| 19 | I.OBS | PA[13] | I/O | — |
| 20 | O.PIN | PA[13] | I/O | g75.ctl |
| 21 | IO.CTL | G75.CTL | — | — |

## Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 22 | I.OBS | PB[28] | I/O | — |
| 23 | O.PIN | PB[28] | I/O | g58.ctl |
| 24 | IO.CTL | G58.CTL | — | — |
| 25 | I.OBS | PC[14] | I/O | — |
| 26 | O.PIN | PC[14] | I/O | g36.ctl |
| 27 | IO.CTL | G36.CTL | — | — |
| 28 | I.OBS | PA[14] | I/O | — |
| 29 | O.PIN | PA[14] | I/O | g76.ctl |
| 30 | IO.CTL | G76.CTL | — | — |
| 31 | I.OBS | PB[29] | I/O | — |
| 32 | O.PIN | PB[29] | I/O | g59.ctl |
| 33 | IO.CTL | G59.CTL | — | — |
| 34 | I.OBS | PC[15] | I/O | — |
| 35 | O.PIN | PC[15] | I/O | g37.ctl |
| 36 | IO.CTL | G37.CTL | — | — |
| 37 | I.OBS | PB[30] | I/O | — |
| 38 | O.PIN | PB[30] | I/O | g60.ctl |
| 39 | IO.CTL | G60.CTL | — | — |
| 40 | I.OBS | PA[15] | I/O | — |
| 41 | O.PIN | PA[15] | I/O | g77.ctl |
| 42 | IO.CTL | G77.CTL | — | — |
| 43 | I.OBS | PB[31] | I/O | — |
| 44 | O.PIN | PB[31] | I/O | g61.ctl |
| 45 | IO.CTL | G61.CTL | — | — |
| 46 | I.OBS | A[0] | I/O | — |
| 47 | O.PIN | A[0] | I/O | g203.ctl |
| 48 | IO.CTL | G203.CTL | — | — |
| 49 | I.OBS | A[1] | I/O | — |
| 50 | O.PIN | A[1] | I/O | g203.ctl |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 51 | I.OBS | A[2] | I/O | — |
| 52 | O.PIN | A[2] | I/O | g203.ctl |
| 53 | I.OBS | A[3] | I/O | — |
| 54 | O.PIN | A[3] | I/O | g203.ctl |
| 55 | I.OBS | A[4] | I/O | — |
| 56 | O.PIN | A[4] | I/O | g203.ctl |
| 57 | I.OBS | A[5] | I/O | — |
| 58 | O.PIN | A[5] | I/O | g203.ctl |
| 59 | I.OBS | A[6] | I/O | — |
| 60 | O.PIN | A[6] | I/O | g203.ctl |
| 61 | I.OBS | A[7] | I/O | — |
| 62 | O.PIN | A[7] | I/O | g203.ctl |
| 63 | I.OBS | A[8] | I/O | — |
| 64 | O.PIN | A[8] | I/O | g202.ctl |
| 65 | IO.CTL | G202.CTL | — | — |
| 66 | I.OBS | A[9] | I/O | — |
| 67 | O.PIN | A[9] | I/O | g202.ctl |
| 68 | I.OBS | A[10] | I/O | — |
| 69 | O.PIN | A[10] | I/O | g202.ctl |
| 70 | I.OBS | A[11] | I/O | — |
| 71 | O.PIN | A[11] | I/O | g202.ctl |
| 72 | I.OBS | A[12] | I/O | — |
| 73 | O.PIN | A[12] | I/O | g202.ctl |
| 74 | I.OBS | A[13] | I/O | — |
| 75 | O.PIN | A[13] | I/O | g202.ctl |
| 76 | I.OBS | A[14] | I/O | — |
| 77 | O.PIN | A[14] | I/O | g202.ctl |
| 78 | I.OBS | A[15] | I/O | — |
| 79 | O.PIN | A[15] | I/O | g202.ctl |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 80 | I.OBS | A[16] | I/O | — |
| 81 | O.PIN | A[16] | I/O | g201.ctl |
| 82 | IO.CTL | G201.CTL | — | — |
| 83 | I.OBS | A[17] | I/O | — |
| 84 | O.PIN | A[17] | I/O | g201.ctl |
| 85 | I.OBS | A[27] | I/O | — |
| 86 | O.PIN | A[27] | I/O | g201.ctl |
| 87 | I.OBS | A[19] | I/O | — |
| 88 | O.PIN | A[19] | I/O | g201.ctl |
| 89 | I.OBS | A[20] | I/O | — |
| 90 | O.PIN | A[20] | I/O | g201.ctl |
| 91 | I.OBS | A[21] | I/O | — |
| 92 | O.PIN | A[21] | I/O | g201.ctl |
| 93 | I.OBS | A[29] | I/O | — |
| 94 | O.PIN | A[29] | I/O | g201.ctl |
| 95 | I.OBS | A[23] | I/O | — |
| 96 | O.PIN | A[23] | I/O | g201.ctl |
| 97 | I.OBS | A[24] | I/O | — |
| 98 | O.PIN | A[24] | I/O | g200.ctl |
| 99 | IO.CTL | G200.CTL | — | — |
| 100 | I.OBS | A[25] | I/O | — |
| 101 | O.PIN | A[25] | I/O | g200.ctl |
| 102 | I.OBS | A[30] | I/O | — |
| 103 | O.PIN | A[30] | I/O | g200.ctl |
| 104 | I.OBS | A[18] | I/O | — |
| 105 | O.PIN | A[18] | I/O | g200.ctl |
| 106 | I.OBS | A[28] | I/O | — |
| 107 | O.PIN | A[28] | I/O | g200.ctl |
| 108 | I.OBS | A[22] | I/O | — |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 109 | O.PIN | A[22] | I/O | g200.ctl |
| 110 | I.OBS | A[26] | I/O | — |
| 111 | O.PIN | A[26] | I/O | g200.ctl |
| 112 | I.OBS | A[31] | I/O | — |
| 113 | O.PIN | A[31] | I/O | g200.ctl |
| 114 | I.OBS | TSIZ0_REG_B | I/O | — |
| 115 | O.PIN | TSIZ0_REG_B | I/O | g204.ctl |
| 116 | I.OBS | TSIZ1 | I/O | — |
| 117 | O.PIN | TSIZ1 | I/O | g204.ctl |
| 118 | IO.CTL | G204.CTL | — | — |
| 119 | O.PIN | BSA3_B | OUTPUT | — |
| 120 | O.PIN | BSA2_B | OUTPUT | — |
| 121 | O.PIN | BSA1_B | OUTPUT | — |
| 122 | O.PIN | BSA0_B | OUTPUT | — |
| 123 | I.OBS | SPARE1 | I/O | — |
| 124 | O.PIN | SPARE1 | I/O | g87.ctl |
| 125 | IO.CTL | G87.CTL | — | — |
| 126 | O.PIN | WE0_B_BSB0_B_IORD_B | OUTPUT | — |
| 127 | O.PIN | WE1_B_BSB1_B_IOWR_B | OUTPUT | — |
| 128 | O.PIN | WE2_B_BSB2_B_PCOE_B | OUTPUT | — |
| 129 | O.PIN | WE3_B_BSB3_B_PCWE_B | OUTPUT | — |
| 130 | O.PIN | GPLA0_B_GPLB0_B | OUTPUT | — |
| 131 | O.PIN | OE_B_GPLAB1_B | OUTPUT | — |
| 132 | O.PIN | GPLAB2_B_CS2_B | OUTPUT | — |
| 133 | O.PIN | GPLAB3_B_CS3_B | OUTPUT | — |
| 134 | O.PIN | CS4_B | OUTPUT | — |
| 135 | O.PIN | CS5_B | OUTPUT | — |
| 136 | O.PIN | CS6_CE1B_B | OUTPUT | — |
| 137 | O.PIN | CS7_B_CE2B_B | OUTPUT | — |

Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 138 | O.PIN | $\overline{\text{CE2A}}$_B | OUTPUT | — |
| 139 | O.PIN | $\overline{\text{CE1A}}$_B | OUTPUT | — |
| 140 | O.PIN | $\overline{\text{CS3}}$_B | OUTPUT | — |
| 141 | O.PIN | $\overline{\text{CS2}}$_B | OUTPUT | — |
| 142 | O.PIN | $\overline{\text{CS1}}$_B | OUTPUT | — |
| 143 | O.PIN | $\overline{\text{CS0}}$_B | OUTPUT | — |
| 144 | I.OBS | WR_B | I/O | — |
| 145 | O.PIN | WR_B | I/O | g96.ctl |
| 146 | IO.CTL | G96.CTL | — | — |
| 147 | I.OBS | $\overline{\text{GPLB4}}$_B | I/O | — |
| 148 | O.PIN | $\overline{\text{GPLB4}}$_B | I/O | g24.ctl |
| 149 | IO.CTL | G24.CTL | — | — |
| 150 | O.PIN | $\overline{\text{GPLA5}}$_B | OUTPUT | — |
| 151 | I.OBS | $\overline{\text{GPLA4}}$_B | I/O | — |
| 152 | O.PIN | $\overline{\text{GPLA4}}$_B | I/O | g25.ctl |
| 153 | IO.CTL | G25.CTL | — | — |
| 154 | O.PIN | $\overline{\text{BDIP}}$_B_$\overline{\text{GPLB5}}$_B | OUTPUT | — |
| 155 | I.OBS | $\overline{\text{BI}}$_B | I/O | — |
| 156 | O.PIN | $\overline{\text{BI}}$_B | I/O | g23.ctl |
| 157 | IO.CTL | G23.CTL | — | — |
| 158 | I.OBS | $\overline{\text{TA}}$_B | I/O | — |
| 159 | O.PIN | $\overline{\text{TA}}$_B | I/O | g22.ctl |
| 160 | IO.CTL | G22.CTL | — | — |
| 161 | I.OBS | $\overline{\text{TEA}}$_B | I/O | — |
| 162 | O.PIN | $\overline{\text{TEA}}$_B | I/O | g89.ctl |
| 163 | IO.CTL | G89.CTL | — | — |
| 164 | I.OBS | $\overline{\text{TS}}$_B | I/O | — |
| 165 | O.PIN | $\overline{\text{TS}}$_B | I/O | g97.ctl |
| 166 | IO.CTL | G97.CTL | — | — |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 167 | I.OBS | $\overline{\text{BR}}$_B | I/O | — |
| 168 | O.PIN | $\overline{\text{BR}}$_B | I/O | g21.ctl |
| 169 | IO.CTL | G21.CTL | — | — |
| 170 | I.OBS | $\overline{\text{BG}}$_B | I/O | — |
| 171 | O.PIN | $\overline{\text{BG}}$_B | I/O | g20.ctl |
| 172 | IO.CTL | G20.CTL | — | — |
| 173 | I.OBS | $\overline{\text{BB}}$_B | I/O | — |
| 174 | O.PIN | $\overline{\text{BB}}$_B | I/O | g11.ctl |
| 175 | IO.CTL | G11.CTL | — | — |
| 176 | I.OBS | SPARE4 | I/O | — |
| 177 | O.PIN | SPARE4 | I/O | g86.ctl |
| 178 | IO.CTL | G86.CTL | — | — |
| 179 | I.OBS | FRZ_B_$\overline{\text{IRQ6}}$_B | I/O | — |
| 180 | O.PIN | FRZ_B_$\overline{\text{IRQ6}}$_B | I/O | g90.ctl |
| 181 | IO.CTL | G90.CTL | — | — |
| 182 | I.OBS | CR_B_$\overline{\text{IRQ3}}$_B | INPUT | — |
| 183 | I.OBS | $\overline{\text{BURST}}$_B | I/O | — |
| 184 | O.PIN | $\overline{\text{BURST}}$_B | I/O | g205.ctl |
| 185 | IO.CTL | G205.CTL | — | — |
| 186 | I.OBS | RSV_B_$\overline{\text{IRQ2}}$_B | I/O | — |
| 187 | O.PIN | RSV_B_$\overline{\text{IRQ2}}$_B | I/O | g17.ctl |
| 188 | IO.CTL | G17.CTL | — | — |
| 189 | I.OBS | IPB5_LWP1_VF1 | I/O | — |
| 190 | O.PIN | IPB5_LWP1_VF1 | I/O | g15.ctl |
| 191 | IO.CTL | G15.CTL | — | — |
| 192 | I.OBS | IPB4_LWP0_VF0 | I/O | — |
| 193 | O.PIN | IPB4_LWP0_VF0 | I/O | g150.ctl |
| 194 | IO.CTL | G150.CTL | — | — |
| 195 | I.OBS | IPB3_LWP2_VF2 | I/O | — |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 196 | O.PIN | IPB3_LWP2_VF2 | I/O | g15.ctl |
| 197 | I.OBS | IPB1_IWP1_VFLS1 | I/O | — |
| 198 | O.PIN | IPB1_IWP1_VFLS1 | I/O | g150.ctl |
| 199 | I.OBS | IPB0_IWP0_VFLS0 | I/O | — |
| 200 | O.PIN | IPB0_IWP0_VFLS0 | I/O | g150.ctl |
| 201 | I.OBS | IPB7_PTR_AT3 | I/O | — |
| 202 | O.PIN | IPB7_PTR_AT3 | I/O | g13.ctl |
| 203 | IO.CTL | G13.CTL | — | — |
| 204 | I.OBS | IPB2_$\overline{\text{IOIS16B}}$_B_AT2 | I/O | — |
| 205 | O.PIN | IPB2_$\overline{\text{IOIS16B}}$_B_AT2 | I/O | g41.ctl |
| 206 | IO.CTL | G41.CTL | — | — |
| 207 | I.OBS | ALEB_DSCK_AT1 | I/O | — |
| 208 | O.PIN | ALEB_DSCK_AT1 | I/O | g16.ctl |
| 209 | IO.CTL | G16.CTL | — | — |
| 210 | I.OBS | IPB6_DSDI_AT0 | I/O | — |
| 211 | O.PIN | IPB6_DSDI_AT0 | I/O | g14.ctl |
| 212 | IO.CTL | G14.CTL | — | — |
| 213 | O.PIN | ALEA | OUTPUT | — |
| 214 | I.OBS | $\overline{\text{KR}}$_B_$\overline{\text{IRQ4}}$_B | I/O | — |
| 215 | O.PIN | $\overline{\text{KR}}$_B_$\overline{\text{IRQ4}}$_B | I/O | g95.ctl |
| 216 | IO.CTL | G95.CTL | — | — |
| 217 | I.OBS | $\overline{\text{AS}}$_B | I/O | — |
| 218 | O.PIN | $\overline{\text{AS}}$_B | I/O | g82.ctl |
| 219 | IO.CTL | G82.CTL | — | — |
| 220 | I.OBS | BADDR[30] | I/O | — |
| 221 | O.PIN | BADDR[30] | I/O | g83.ctl |
| 222 | IO.CTL | G83.CTL | — | — |
| 223 | O.PIN | OP0 | OUTPUT | — |
| 224 | O.PIN | OP1 | OUTPUT | — |

Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 225 | I.OBS | OP2_MODCK1_$\overline{\text{STS}}$_B | I/O | — |
| 226 | O.PIN | OP2_MODCK1_$\overline{\text{STS}}$_B | I/O | g92.ctl |
| 227 | IO.CTL | G92.CTL | — | — |
| 228 | I.OBS | OP3_MODCK2_DSDO | I/O | — |
| 229 | O.PIN | OP3_MODCK2_DSDO | I/O | g91.ctl |
| 230 | IO.CTL | G91.CTL | — | — |
| 231 | I.OBS | BADDR[29] | I/O | — |
| 232 | O.PIN | BADDR[29] | I/O | g84.ctl |
| 233 | IO.CTL | G84.CTL | — | — |
| 234 | I.OBS | BADDR[28] | I/O | — |
| 235 | O.PIN | BADDR[28] | I/O | g85.ctl |
| 236 | IO.CTL | G85.CTL | — | — |
| 237 | I.OBS | CLK4IN | INPUT | — |
| 238 | O.PIN | TEXP | OUTPUT | — |
| 239 | I.OBS | $\overline{\text{HRESET}}$_B | I/O | — |
| 240 | O.PIN | $\overline{\text{HRESET}}$_B | I/O | g94.ctl |
| 241 | IO.CTL | G94.CTL | — | — |
| 242 | I.OBS | $\overline{\text{SRESET}}$_B | I/O | — |
| 243 | O.PIN | $\overline{\text{SRESET}}$_B | I/O | g93.ctl |
| 244 | IO.CTL | G93.CTL | — | — |
| 245 | I.OBS | $\overline{\text{RSTCONF}}$_B | INPUT | — |
| 246 | I.OBS | $\overline{\text{PORESET}}$_B | INPUT | — |
| 247 | I.OBS | $\overline{\text{WAITA}}$_B | INPUT | — |
| 248 | I.OBS | $\overline{\text{WAITB}}$_B | INPUT | — |
| 249 | I.OBS | IPA7 | INPUT | — |
| 250 | I.OBS | IPA0 | INPUT | — |
| 251 | I.OBS | IPA1 | INPUT | — |
| 252 | I.OBS | IPA2_$\overline{\text{IOIS16A}}$_B | INPUT | — |
| 253 | I.OBS | IPA3 | INPUT | — |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 254 | I.OBS | IPA4 | INPUT | — |
| 255 | I.OBS | IPA5 | INPUT | — |
| 256 | I.OBS | IPA6 | INPUT | — |
| 257 | O.PIN | CLKOUT | OUTPUT | — |
| 258 | I.OBS | DP0_$\overline{\text{IRQ3}}$_B | I/O | — |
| 259 | O.PIN | DP0_$\overline{\text{IRQ3}}$_B | I/O | g18.ctl |
| 260 | I.OBS | DP3_$\overline{\text{IRQ6}}$_B | I/O | — |
| 261 | O.PIN | DP3_$\overline{\text{IRQ6}}$_B | I/O | g18.ctl |
| 262 | IO.CTL | G18.CTL | — | — |
| 263 | I.OBS | DP2_$\overline{\text{IRQ5}}$_B | I/O | — |
| 264 | O.PIN | DP2_$\overline{\text{IRQ5}}$_B | I/O | g18.ctl |
| 265 | I.OBS | DP1_$\overline{\text{IRQ4}}$_B | I/O | — |
| 266 | O.PIN | DP1_$\overline{\text{IRQ4}}$_B | I/O | g18.ctl |
| 267 | I.OBS | D[31] | I/O | — |
| 268 | O.PIN | D[31] | I/O | g103.ctl |
| 269 | IO.CTL | G103.CTL | — | — |
| 270 | I.OBS | D[30] | I/O | — |
| 271 | O.PIN | D[30] | I/O | g103.ctl |
| 272 | I.OBS | D[29] | I/O | — |
| 273 | O.PIN | D[29] | I/O | g103.ctl |
| 274 | I.OBS | D[28] | I/O | — |
| 275 | O.PIN | D[28] | I/O | g103.ctl |
| 276 | I.OBS | D[7] | I/O | — |
| 277 | O.PIN | D[7] | I/O | g103.ctl |
| 278 | I.OBS | D[26] | I/O | — |
| 279 | O.PIN | D[26] | I/O | g103.ctl |
| 280 | I.OBS | D[25] | I/O | — |
| 281 | O.PIN | D[25] | I/O | g103.ctl |
| 282 | I.OBS | D[24] | I/O | — |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 283 | O.PIN | D[24] | I/O | g103.ctl |
| 284 | I.OBS | D[6] | I/O | — |
| 285 | O.PIN | D[6] | I/O | g102.ctl |
| 286 | IO.CTL | G102.CTL | — | — |
| 287 | I.OBS | D[22] | I/O | — |
| 288 | O.PIN | D[22] | I/O | g102.ctl |
| 289 | I.OBS | D[21] | I/O | — |
| 290 | O.PIN | D[21] | I/O | g102.ctl |
| 291 | I.OBS | D[20] | I/O | — |
| 292 | O.PIN | D[20] | I/O | g102.ctl |
| 293 | I.OBS | D[19] | I/O | — |
| 294 | O.PIN | D[19] | I/O | g102.ctl |
| 295 | I.OBS | D[18] | I/O | — |
| 296 | O.PIN | D[18] | I/O | g102.ctl |
| 297 | I.OBS | D[5] | I/O | — |
| 298 | O.PIN | D[5] | I/O | g102.ctl |
| 299 | I.OBS | D[16] | I/O | — |
| 300 | O.PIN | D[16] | I/O | g102.ctl |
| 301 | I.OBS | D[15] | I/O | — |
| 302 | O.PIN | D[15] | I/O | g101.ctl |
| 303 | IO.CTL | G101.CTL | — | — |
| 304 | I.OBS | D[14] | I/O | — |
| 305 | O.PIN | D[14] | I/O | g101.ctl |
| 306 | I.OBS | D[3] | I/O | — |
| 307 | O.PIN | D[3] | I/O | g101.ctl |
| 308 | I.OBS | D[2] | I/O | — |
| 309 | O.PIN | D[2] | I/O | g101.ctl |
| 310 | I.OBS | D[11] | I/O | — |
| 311 | O.PIN | D[11] | I/O | g101.ctl |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 312 | I.OBS | D[10] | I/O | — |
| 313 | O.PIN | D[10] | I/O | g101.ctl |
| 314 | I.OBS | D[9] | I/O | — |
| 315 | O.PIN | D[9] | I/O | g101.ctl |
| 316 | I.OBS | D[1] | I/O | — |
| 317 | O.PIN | D[1] | I/O | g101.ctl |
| 318 | I.OBS | D[27] | I/O | — |
| 319 | O.PIN | D[27] | I/O | g100.ctl |
| 320 | IO.CTL | G100.CTL | — | — |
| 321 | I.OBS | D[23] | I/O | — |
| 322 | O.PIN | D[23] | I/O | g100.ctl |
| 323 | I.OBS | D[17] | I/O | — |
| 324 | O.PIN | D[17] | I/O | g100.ctl |
| 325 | I.OBS | D[4] | I/O | — |
| 326 | O.PIN | D[4] | I/O | g100.ctl |
| 327 | I.OBS | D[13] | I/O | — |
| 328 | O.PIN | D[13] | I/O | g100.ctl |
| 329 | I.OBS | D[12] | I/O | — |
| 330 | O.PIN | D[12] | I/O | g100.ctl |
| 331 | I.OBS | D[8] | I/O | — |
| 332 | O.PIN | D[8] | I/O | g100.ctl |
| 333 | I.OBS | D[0] | I/O | — |
| 334 | O.PIN | D[0] | I/O | g100.ctl |
| 335 | I.OBS | $\overline{\text{IRQ0}}$_B | INPUT | — |
| 336 | I.OBS | $\overline{\text{IRQ1}}$_B | INPUT | — |
| 337 | I.OBS | $\overline{\text{IRQ7}}$_B | INPUT | — |
| 338 | I.OBS | SPARE3 | I/O | — |
| 339 | O.PIN | SPARE3 | I/O | g10.ctl |
| 340 | IO.CTL | G10.CTL | — | — |

Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 341 | I.OBS | SHIFT_CLK_SDACK2_B | I/O | — |
| 342 | O.PIN | SHIFT_CLK_SDACK2_B | I/O | g9.ctl |
| 343 | IO.CTL | G9.CTL | — | — |
| 344 | I.OBS | FRAME_VSYNC | I/O | — |
| 345 | O.PIN | FRAME_VSYNC | I/O | g8.ctl |
| 346 | IO.CTL | G8.CTL | — | — |
| 347 | I.OBS | LCDAC_LOE | I/O | — |
| 348 | O.PIN | LCDAC_LOE | I/O | g7.ctl |
| 349 | IO.CTL | G7.CTL | — | — |
| 350 | I.OBS | LOAD_HSYNC_SDACK1_B | I/O | — |
| 351 | O.PIN | LOAD_HSYNC_SDACK1_B | I/O | g6.ctl |
| 352 | IO.CTL | G6.CTL | — | — |
| 353 | I.OBS | LD0 | I/O | — |
| 354 | O.PIN | LD0 | I/O | g5.ctl |
| 355 | IO.CTL | G5.CTL | — | — |
| 356 | I.OBS | LD1 | I/O | — |
| 357 | O.PIN | LD1 | I/O | g4.ctl |
| 358 | IO.CTL | G4.CTL | — | — |
| 359 | I.OBS | LD2 | I/O | — |
| 360 | O.PIN | LD2 | I/O | g3.ctl |
| 361 | IO.CTL | G3.CTL | — | — |
| 362 | I.OBS | LD3 | I/O | — |
| 363 | O.PIN | LD3 | I/O | g2.ctl |
| 364 | IO.CTL | G2.CTL | — | — |
| 365 | I.OBS | LD4 | I/O | — |
| 366 | O.PIN | LD4 | I/O | g1.ctl |
| 367 | IO.CTL | G1.CTL | — | g1.ctl |
| 368 | I.OBS | LD5 | I/O | — |
| 369 | O.PIN | LD5 | I/O | g81.ctl |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 370 | IO.CTL | G81.CTL | — | g81.ctl |
| 371 | I.OBS | LD6 | I/O | — |
| 372 | O.PIN | LD6 | I/O | g80.ctl |
| 373 | IO.CTL | G80.CTL | — | g80.ctl |
| 374 | I.OBS | LD7 | I/O | — |
| 375 | O.PIN | LD7 | I/O | g79.ctl |
| 376 | IO.CTL | G79.CTL | — | g79.ctl |
| 377 | I.OBS | LD8 | I/O | — |
| 378 | O.PIN | LD8 | I/O | g78.ctl |
| 379 | IO.CTL | G78.CTL | — | g78.ctl |
| 380 | I.OBS | PA[0] | I/O | — |
| 381 | O.PIN | PA[0] | I/O | g62.ctl |
| 382 | IO.CTL | G62.CTL | — | g62.ctl |
| 383 | I.OBS | PB[14] | I/O | — |
| 384 | O.PIN | PB[14] | I/O | g38.ctl |
| 385 | IO.CTL | G38.CTL | — | g38.ctl |
| 386 | I.OBS | PC[4] | I/O | — |
| 387 | O.PIN | PC[4] | I/O | g26.ctl |
| 388 | IO.CTL | G26.CTL | — | g26.ctl |
| 389 | I.OBS | PA[1] | I/O | — |
| 390 | O.PIN | PA[1] | I/O | g63.ctl |
| 391 | IO.CTL | G63.CTL | — | g63.ctl |
| 392 | I.OBS | PB[15] | I/O | — |
| 393 | O.PIN | PB[15] | I/O | g39.ctl |
| 394 | IO.CTL | G39.CTL | — | g39.ctl |
| 395 | I.OBS | PC[5] | I/O | — |
| 396 | O.PIN | PC[5] | I/O | g27.ctl |
| 397 | IO.CTL | G27.CTL | — | g27.ctl |
| 398 | I.OBS | PA[2] | I/O | — |

### Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 399 | O.PIN | PA[2] | I/O | g64.ctl |
| 400 | IO.CTL | G64.CTL | — | g64.ctl |
| 401 | I.OBS | PB[16] | I/O | — |
| 402 | O.PIN | PB[16] | I/O | g40.ctl |
| 403 | IO.CTL | G40.CTL | — | g40.ctl |
| 404 | I.OBS | PC[6] | I/O | — |
| 405 | O.PIN | PC[6] | I/O | g28.ctl |
| 406 | IO.CTL | G28.CTL | — | g28.ctl |
| 407 | I.OBS | PA[3] | I/O | — |
| 408 | O.PIN | PA[3] | I/O | g65.ctl |
| 409 | IO.CTL | G65.CTL | — | g65.ctl |
| 410 | I.OBS | PB[17] | I/O | — |
| 411 | O.PIN | PB[17] | I/O | g47.ctl |
| 412 | IO.CTL | G47.CTL | — | g47.ctl |
| 413 | I.OBS | PC[7] | I/O | — |
| 414 | O.PIN | PC[7] | I/O | g29.ctl |
| 415 | IO.CTL | G29.CTL | — | g29.ctl |
| 416 | I.OBS | PA[4] | I/O | — |
| 417 | O.PIN | PA[4] | I/O | g66.ctl |
| 418 | IO.CTL | G66.CTL | — | g66.ctl |
| 419 | I.OBS | PB[18] | I/O | — |
| 420 | O.PIN | PB[18] | I/O | g48.ctl |
| 421 | IO.CTL | G48.CTL | — | g48.ctl |
| 422 | I.OBS | PA[5] | I/O | — |
| 423 | O.PIN | PA[5] | I/O | g67.ctl |
| 424 | IO.CTL | G67.CTL | — | g67.ctl |
| 425 | I.OBS | PB[19] | I/O | — |
| 426 | O.PIN | PB[19] | I/O | g49.ctl |
| 427 | IO.CTL | G49.CTL | — | g49.ctl |

Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 428 | I.OBS | PA[6] | I/O | — |
| 429 | O.PIN | PA[6] | I/O | g68.ctl |
| 430 | IO.CTL | G68.CTL | — | g68.ctl |
| 431 | I.OBS | PB[20] | I/O | — |
| 432 | O.PIN | PB[20] | I/O | g50.ctl |
| 433 | IO.CTL | G50.CTL | — | g50.ctl |
| 434 | I.OBS | PC[8] | I/O | — |
| 435 | O.PIN | PC[8] | I/O | g30.ctl |
| 436 | IO.CTL | G30.CTL | — | g30.ctl |
| 437 | I.OBS | PA[7] | I/O | — |
| 438 | O.PIN | PA[7] | I/O | g69.ctl |
| 439 | IO.CTL | G69.CTL | — | g69.ctl |
| 440 | I.OBS | PB[21] | I/O | — |
| 441 | O.PIN | PB[21] | I/O | g51.ctl |
| 442 | IO.CTL | G51.CTL | — | g51.ctl |
| 443 | I.OBS | PC[9] | I/O | — |
| 444 | O.PIN | PC[9] | I/O | g31.ctl |
| 445 | IO.CTL | G31.CTL | — | g31.ctl |
| 446 | I.OBS | PA[8] | I/O | — |
| 447 | O.PIN | PA[8] | I/O | g70.ctl |
| 448 | IO.CTL | G70.CTL | — | g70.ctl |
| 449 | I.OBS | PB[22] | I/O | — |
| 450 | O.PIN | PB[22] | I/O | g52.ctl |
| 451 | IO.CTL | G52.CTL | — | g52.ctl |
| 452 | I.OBS | PC[10] | I/O | — |
| 453 | O.PIN | PC[10] | I/O | g32.ctl |
| 454 | IO.CTL | G32.CTL | — | g32.ctl |
| 455 | I.OBS | PA[9] | I/O | — |
| 456 | O.PIN | PA[9] | I/O | g71.ctl |

Table 20-1. Boundary Scan Bit Definition (Continued)

| BIT | CELL TYPE | PIN/CELL NAME | PIN TYPE | OUTPUT CTL CELL |
|---|---|---|---|---|
| 457 | IO.CTL | G71.CTL | — | g71.ctl |
| 458 | I.OBS | PB[23] | I/O | — |
| 459 | O.PIN | PB[23] | I/O | g53.ctl |
| 460 | IO.CTL | G53.CTL | — | g53.ctl |
| 461 | I.OBS | PC[11] | I/O | — |
| 462 | O.PIN | PC[11] | I/O | g33.ctl |
| 463 | IO.CTL | G33.CTL | — | g33.ctl |
| 464 | I.OBS | PB[24] | I/O | — |
| 465 | O.PIN | PB[24] | I/O | g54.ctl |
| 466 | IO.CTL | G54.CTL | — | g54.ctl |
| 467 | I.OBS | PA[10] | I/O | — |
| 468 | O.PIN | PA[10] | I/O | g72.ctl |
| 469 | IO.CTL | G72.CTL | — | g72.ctl |
| 470 | I.OBS | PB[25] | I/O | — |
| 471 | O.PIN | PB[25] | I/O | g55.ctl |
| 472 | IO.CTL | G55.CTL | — | g55.ctl |
| 473 | I.OBS | SPARE2 | I/O | — |
| 474 | O.PIN | SPARE2 | I/O | g88.ctl |
| 475 | IO.CTL | G88.CTL | — | g88.ctl |

## 20.4 INSTRUCTION REGISTER

The MPC821 JTAG implementation includes the public instructions (EXTEST, SAMPLE/PRELOAD, and BYPASS), and also supports the CLAMP instruction. One additional public instruction (HI-Z) provides the capability for disabling all device output drivers. The MPC821 includes a 4-bit instruction register (no parity) that consists of a shift register with four parallel outputs. Data is transferred from the shift register to the parallel outputs during the update-IR controller state. The four bits are used to decode the five unique instructions listed in Table 20-2.

**Table 20-2. Instruction Decoding**

| CODE | | | | INSTRUCTION |
|---|---|---|---|---|
| B3 | B2 | B1 | B0 | |
| 0 | 0 | 0 | 0 | EXTEST |
| 0 | 0 | 0 | 1 | SAMPLE/PRELOAD |
| 0 | X | 1 | X | BYPASS |
| 0 | 1 | 0 | 0 | HI—Z |
| | 1 | 0 | 1 | CLAMP and BYPASS |

NOTE: B0 (LSB) is shifted first.

The parallel output of the instruction register is reset to all ones in the test-logic-reset controller state. Notice that this preset state is equivalent to the BYPASS instruction. During the capture-IR controller state, the parallel inputs to the instruction shift register are loaded with the CLAMP command code.

### 20.4.1 EXTEST

The external test (EXTEST) instruction selects the 475-bit boundary scan register. EXTEST also asserts an internal reset for the MPC821 system logic to force a known beginning internal state while performing external boundary scan operations. By using the TAP, the register is capable of scanning user-defined values into the output buffers, capturing values presented to input pins, and controlling the output drive of three-stateable output or bidirectional pins. For more details on the function and use of EXTEST, refer to the IEEE 1149.1 standard.

### 20.4.2 SAMPLE/PRELOAD

The SAMPLE/PRELOAD instruction initializes the boundary scan register output cells prior to the selection of EXTEST. This initialization ensures that known data will appear on the outputs when entering the EXTEST instruction. The SAMPLE/PRELOAD instruction also provides an opportunity to obtain a snapshot of system data and control signals.

**NOTE**

Since there is no internal synchronization between the TCK and CLKOUT, 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.

## 20.4.3 BYPASS

The BYPASS instruction creates a shift register path from TDI to the bypass register and, finally, to TDO, circumventing the 475-bit boundary scan register. This instruction is used to enhance test efficiency when a component other than the MPC821 becomes the device under test. It selects the single-bit bypass register as illustrated in Figure 20-7.

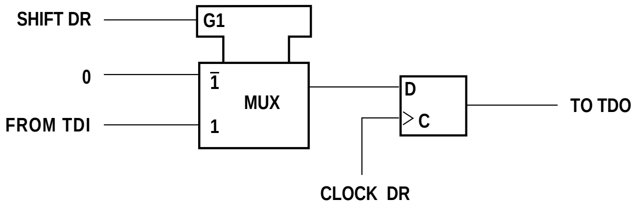


**Figure 20-7. Bypass Register**

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.

## 20.4.4 CLAMP

The CLAMP instruction selects the single-bit bypass register as illustrated in Figure 20-7 above, and the state of all signals driven from the system output pins is completely defined by the data previously shifted into the boundary scan register. For example, using the SAMPLE/PRELOAD instruction.

## 20.4.5 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 HI-Z is invoked all output drivers, including the two-state drivers, are turned off (high impedance). The instruction selects the bypass register.

## 20.5 MPC821 RESTRICTIONS

The control afforded by the output enable signals using the boundary scan register and the EXTEST instruction requires a compatible circuit-board test environment to avoid device-destructive configurations. The user must avoid situations in which the MPC821 output drivers are enabled into actively driven networks.

The MPC821 features a low-power stop mode. The interaction of the scan chain interface with low-power stop mode is as follows:

1. The TAP controller must be in the test-logic-reset state to either enter or remain in the low-power stop mode. Leaving the TAP controller in the test-logic-reset state negates the ability to achieve low-power, but does not otherwise affect device functionality.
2. The TCK input is not disabled in low-power stop mode. To consume minimal power, the TCK input should be externally connected to $V_{CC}$ or ground while in low power mode or normal mode (nonscan chain).
3. The TMS, TDI, and $\overline{\text{TRST}}$ pins include on-chip pull-up resistors. In low-power stop mode, these two pins should remain either unconnected or connected to $V_{CC}$ to achieve minimal power consumption. For proper reset of the scan chain test logic the best approach is to pull active $\overline{\text{TRST}}$ at power-on reset. The easiest way to reset the scan chain logic is to connect $\overline{\text{TRST}}$ to $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$, or $\overline{\text{PORESET}}$.

## 20.6 NONSCAN CHAIN OPERATION

In nonscan chain operation, there are two constraints. First, the TCK input does not include an internal pull-up resistor and should be tied high or low to preclude mid-level inputs. The second constraint is to ensure that the scan chain test logic is kept transparent to the system logic by forcing TAP into the test-logic-reset controller state, using either of two methods. The first method is to connect pin $\overline{\text{TRST}}$ to logic 0 (or one of the reset pins). The second method is to assure TMS is sampled as a logic one for five consecutive TCK rising edges. If TMS remains connected to $V_{CC}$ or does not change state, then the TAP controller cannot leave the test-logic-reset state, regardless of the state of TCK.

## 20.7 MOTOROLA MPC821 BSDL DESCRIPTION

The BSDL file for exercising the scan chain logic on the MPC821 is available at the Motorola web site (`www.mot.com/pps`) in the MPCxxx Embedded PowerPC area.

# SECTION 21
# ELECTRICAL CHARACTERISTICS

This section contains detailed information on power considerations, DC/AC electrical characteristics, and AC timing specifications for the MPC821.

**NOTE**

The MPC821 electrical specifications are preliminary and many specs are from design simulations. These specifications may not be fully tested or guaranteed at this early stage of the product life cycle. Finalized specifications will be published after thorough characterization and device qualifications have been completed.

## 21.1 MAXIMUM RATINGS (GND = 0V)

| RATING | SYMBOL | VALUE | UNIT |
|---|---|---|---|
| Supply Voltage | VDDH | -0.3 to 4.0 | V |
| | VDD | -0.3 to 4.0 | V |
| | KAPWR | -0.3 to 4.0 | V |
| | VDDSYN | -0.3 to 4.0 | V |
| Input Voltage | VIN | GND-0.3 to 5.8 | V |
| Junction Temperature | $T_j$ | 90@25 MHz<br>or<br>TBD @ 40 MHz | ˚C |
| Storage Temperature Range | $T_{STG}$ | -55 to +150 | ˚C |

NOTE: 1. Functional operating conditions are given in **Section 21.3 Power Considerations**. Absolute maximum ratings are stress ratings only, and functional operation at the maxima is not guaranteed. Stress beyond those listed may affect device reliability or cause permanent damage to the device.

2. **CAUTION**: All “5 Volt Friendly” Input voltages cannot be more than 2.5 V greater than supply voltage, this restriction applies also on “power-on” as well as on normal operation.

3. “5 Volt Friendly” inputs are inputs that tolerate 5 volts.

This device contains circuitry protecting against damage due to high static voltage or electrical fields; however, it is advised that normal precautions be taken to avoid application of any voltages higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (either GND or $V_{CC}$).

## 21.2 THERMAL CHARACTERISTICS

| CHARACTERISTIC | SYMBOL | VALUE | UNIT |
|---|---|---|---|
| Thermal Resistance for BGA | $\theta_{JA}$ | 47[1] | °C/W |
| | $\theta_{JA}$ | 30[2] | °C/W |
| | $\theta_{JA}$ | 15[3] | °C/W |

NOTE:

1. Assumes natural convection and a single layer board (no thermal vias).
2. Assumes natural convection, a mult ilayer board with thermal vias[4], 1 watt MPC821 dissipation, and a board temperature rise of 20°C above ambient.
3. Assumes natural convection, a multilayer board with thermal vias[4],1 watt MPC821dissipation, and a board temperature rise of 10°C above ambient.
4. For more information on the design of thermal vias on multilayer boards and BGA layout considerations in general, refer to AN-1231/D, *Plastic Ball Grid Array Application Note* available from your local Motorola sales office.

$$T_J = T_A + (P_D \bullet \theta_{JA})$$

$$P_D = (V_{DD} \bullet I_{DD}) + P_{I/O}$$

where:

$P_{I/O}$ is the power dissipation on pins.

## 21.3 POWER CONSIDERATIONS

The average chip-junction temperature, $T_J$, in °C can be obtained from

$$T_J = T_A + (P_D \bullet q_{JA}) \qquad (1)$$

where

$T_A$ = Ambient Temperature, ∞C

$q_{JA}$ = Package Thermal Resistance, Junction to Ambient, ∞C/W

$P_D$ = $P_{INT} + P_{I/O}$

$P_{INT}$ = $I_{DD}$ x $V_{DD}$, Watts—Chip Internal Power

$P_{I/O}$ = Power Dissipation on Input and Output Pins—User Determined

For most applications $P_{I/O} < 0.3 \bullet P_{INT}$ and can be neglected. If $P_{I/O}$ is neglected, an approximate relationship between $P_D$ and $T_J$ is:

$$P_D = K \prod (T_J + 273∞C) \qquad (2)$$

Solving equations (1) and (2) for K gives

$$K = P_D \bullet (T_A + 273∞C) + q_{JA} \bullet P_D^2 \qquad (3)$$

where K is a constant pertaining to the particular part. K can be determined from equation (3) by measuring $P_D$ (at equilibrium) for a known $T_A$. Using this value of K, the values of $P_D$ and $T_J$ can be obtained by solving equations (1) and (2) iteratively for any value of $T_A$.

### 21.3.1 Layout Practices

Each $V_{CC}$ pin on the MPC821 should be provided with a low-impedance path to the board’s supply. Each GND pin should likewise be provided with a low-impedance path to ground. The power supply pins drive distinct groups of logic on chip. The $V_{CC}$ power supply should be bypassed to ground using at least four 0.1 µF by-pass capacitors located as close as possible to the four sides of the package. The capacitor leads and associated printed circuit traces connecting to chip $V_{CC}$ and GND should be kept to less than half an inch per capacitor lead. A four-layer board is recommended, employing two inner layers as $V_{CC}$ and GND planes.

All output pins on the MPC821 have fast rise and fall times. Printed circuit (PC) trace interconnection length should be minimized in order to minimize undershoot and reflections caused by these fast output switching times. This recommendation particularly applies to the address and data busses. Maximum PC trace lengths of six inches are recommended. Capacitance calculations should consider all device loads as well as parasitic capacitances due to the PC traces. Attention to proper PCB layout and bypassing becomes especially critical in systems with higher capacitive loads because these loads create higher transient currents in the $V_{CC}$ and GND circuits. Pull up all unused inputs or signals that will be inputs during reset. Special care should be taken to minimize the noise levels on the PLL supply pins.

## 21.4 DC ELECTRICAL CHARACTERISTICS

($T_A$ = 0 ˚C to 70 ˚C; $V_{CC}$ = 3.0 - 3.6 Vdc ±5%; GND = 0 Vdc; unless otherwise noted)

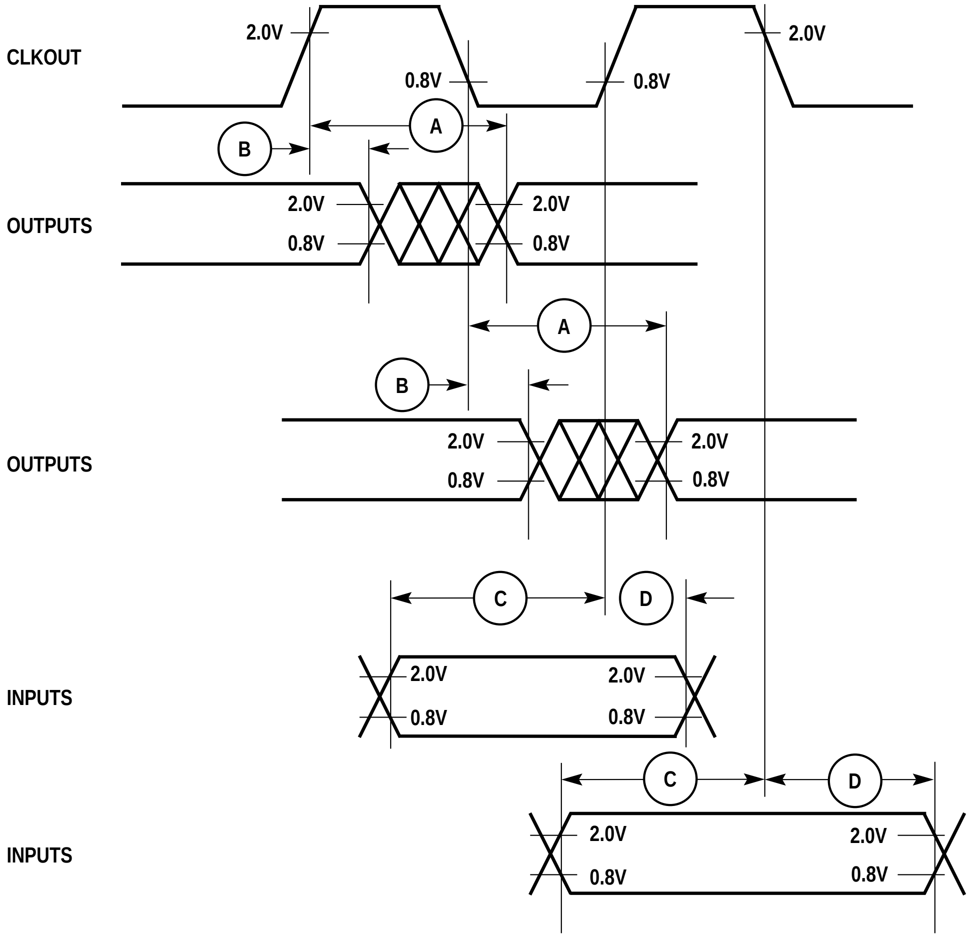
| CHARACTERISTIC | SYMBOL | MIN | MAX | UNIT |
|---|---|---|---|---|
| Input High Voltage, all inputs except EXTAL and CLK4IN * | $V_{IH}$ | 2.0 | 5.5 | V |
| Input Low Voltage | $V_{IL}$ | GND | 0.8 | V |
| EXTAL, EXTCLK Input High Voltage | $V_{IHC}$ | 0.7*($V_{CC}$) | $V_{CC}$+0.3 | V |
| Input Leakage Current, $V_{IN}$ = 5.5 V | $I_{IN}$ | — | TBD | mA |
| Hi-Z (Off State) Leakage Current, $V_{IN}$ = TBD V | $I_{OZ}$ | — | TBD | mA |
| Signal Low Input Current, $V_{IL}$ = 0.8 V | $I_L$ | TBD | TBD | mA |
| Signal High Input Current, $V_{IH}$ = 2.0 V | $I_H$ | TBD | TBD | mA |
| Output High Voltage, $I_{OH}$ = –20 mA, $V_{DDH}$ = 3.0 V<br>Except XTAL, XFC, and Open Drain Pins | $V_{OH}$ | 2.4 | — | V |
| Output Low Voltage,<br>$I_{OL}$ = 2.0 mA CLKOUT<br>$I_{OL}$ = 3.2 mA A(0:31), TSIZ0 / $\overline{\text{REG}}$, TSIZ1,<br>D(0:31), DP(0:3) / $\overline{\text{IRQ}}$(3:6),<br>RD / $\overline{\text{WR}}$, $\overline{\text{BURST}}$, RSV / $\overline{\text{IRQ2}}$,<br>IP_B(0:1) / IWP(0:1) / VFLS(0:1),<br>IP_B2 / $\overline{\text{IOIS16\_B}}$ / AT2,<br>IP_B3 / IWP2 / VF2,<br>IP_B4 / LWP0 / VF0,<br>IP_B5 / LWP1 / VF1,<br>IP_B6 / DSDI / AT0,<br>IP_B7 / PTR / AT3,<br>RXD1 / PA15, RXD2 / PA13,<br>L1TXDB / PA11,<br>L1RXDB / PA10, L1TXDA / PA9,<br>L1RXDA / PA8,<br>TIN1 / L1RCLKA / BRGO1 / CLK1 / PA7,<br>BRGCLK1 / $\overline{\text{TOUT1}}$ / CLK2 / PA6,<br>TIN2 / L1TCLKA / BRGO2 / CLK3 / PA5,<br>$\overline{\text{TOUT2}}$ / CLK4 / PA4,<br>TIN3 / BRGO3 / CLK5 / PA3,<br>BRGCLK2 / L1RCLKB / $\overline{\text{TOUT3}}$ / CLK6 / PA2,<br>TIN4 / BRGO4 / CLK7 / PA1,<br>L1TCLKB / $\overline{\text{TOUT4}}$ / CLK8 / PA0,<br>REJECT1 / $\overline{\text{SPISEL}}$ / PB31,<br>SPICLK / PB30, SPIMOSI / PB29, | $V_{OL}$ | — | 0.5 | V |

| CHARACTERISTIC | SYMBOL | MIN | MAX | UNIT |
|---|---|---|---|---|
| Output Low Voltage (continued),<br>$I_{OL}$ = 3.2 mA BRGO4 / SPIMISO / PB28,<br>BRGO1 / I2CSDA / PB27,<br>BRGO2 / I2CSCL / PB26,<br>SMTXD1 / PB25, SMRXD1 / PB24,<br>$\overline{\text{SMSYN1}}$ / SDACK1 / PB23,<br>$\overline{\text{SMSYN2}}$ / SDACK2 / PB22,<br>SMTXD2 / L1CLKOB / PB21,<br>SMRXD2 / L1CLKOA / PB20,<br>L1ST1 / $\overline{\text{RTS1}}$ / PB19,<br>L1ST2 / $\overline{\text{RTS2}}$ / PB18,<br>L1ST3 / L1RQB / PB17,<br>L1ST4 / L1RQA / PB16,<br>BRGO3 / PB15, $\overline{\text{RSTRT1}}$ / PB14,<br>L1ST1 / $\overline{\text{RTS1}}$ / DREQ0 / PC15,<br>L1ST2 / $\overline{\text{RTS2}}$ / DREQ1 / PC14,<br>L1ST3 / L1RQB / PC13,<br>L1ST4 / L1RQA / PC12,<br>$\overline{\text{CTS1}}$ / PC11, $\overline{\text{TGATE1}}$ / CD1 / PC10,<br>$\overline{\text{CTS2}}$ / PC9, $\overline{\text{TGATE2}}$ / CD2 / PC8,<br>SDACK2 / L1TSYNCB / PC7,<br>L1RSYNCB / PC6,<br>SDACK1 / L1TSYNCA / PC5,<br>L1RSYNCA / PC4, LD8,<br>LD7, LD6, LD5, LD4, LD3,<br>LD2, LD1, FRAME / VSYNC,<br>LCD_AC / $\overline{\text{LOE}}$, LD0,<br>LOAD / HSYNC, SHIFT / CLK<br>$I_{OL}$ = 5.3 mA $\overline{\text{BDIP}}$ / $\overline{\text{GPL\_B}}$(5), $\overline{\text{BR}}$, $\overline{\text{BG}}$,<br>FRZ / $\overline{\text{IRQ6}}$, $\overline{\text{CS}}$(0:5),<br>$\overline{\text{CS}}$(6) / $\overline{\text{CE}}$(1)_B,<br>$\overline{\text{CS}}$(7) / $\overline{\text{CE}}$(2)_B,<br>$\overline{\text{WE0}}$ / $\overline{\text{BS\_B0}}$ / $\overline{\text{IORD}}$,<br>$\overline{\text{WE1}}$ / $\overline{\text{BS\_B1}}$ / $\overline{\text{IOWR}}$,<br>$\overline{\text{WE2}}$ / $\overline{\text{BS\_B2}}$ / $\overline{\text{PCOE}}$,<br>$\overline{\text{WE3}}$ / $\overline{\text{BS\_B3}}$ / $\overline{\text{PCWE}}$,<br>$\overline{\text{BS\_A}}$(0:3), $\overline{\text{GPL\_A0}}$ / $\overline{\text{GPL\_B0}}$,<br>$\overline{\text{OE}}$ / $\overline{\text{GPL\_A1}}$ / $\overline{\text{GPL\_B1}}$,<br>$\overline{\text{GPL\_A}}$(2:3) / $\overline{\text{GPL\_B}}$(2:3) / $\overline{\text{CS}}$(2:3),<br>$\overline{\text{UPWAITA}}$ / $\overline{\text{GPL\_A4}}$,<br>$\overline{\text{UPWAITB}}$ / $\overline{\text{GPL\_B4}}$,<br>$\overline{\text{GPL\_A5}}$, ALE_A, $\overline{\text{CE1\_A}}$, $\overline{\text{CE2\_A}}$,<br>ALE_B / DSCK / AT1, OP(0:1),<br>OP2 / MODCK0 / $\overline{\text{STS}}$,<br>OP3 / MODCK1 / DSDO<br>$I_{OL}$ = 7.0 mA TXD1 / PA14, TXD2 / PA12<br>$I_{OL}$ = 8.9 mA $\overline{\text{TS}}$, $\overline{\text{TA}}$, $\overline{\text{TEA}}$, $\overline{\text{BI}}$, $\overline{\text{BB}}$, $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$ | $V_{OL}$ | — | 0.5 | V |

NOTE: 1. Input pin voltage specifications are $V_{CC}$ = +4 V or 5.8 V, whichever is less.

2. AC timings are based on a 50 p$f$ load.

## 21.5 MPC821 AC ELECTRICAL SPECIFICATIONS



A = MAXIMUM OUTPUT DELAY SPECIFICATION

B = MINIMUM OUTPUT HOLD TIME

C = MINIMUM INPUT SETUP TIME SPECIFICATION

D = MINIMUM INPUT HOLD TIME SPECIFICATION

**Figure 21-1. AC Electrical Specifications Control Timing Diagram**

### Table 21-1. Bus Operation Timing

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ |  | 50 MHZ |  | UNIT |
|---|---|---|---|---|---|---|---|
|  |  |  | MIN | MAX | MIN | MAX |  |
| B1 | CLKOUT Period | TC | — | — | — | — | ns |
| B2 | Clock Pulse Width Low |  | — | — | — | — | ns |
| B3 | Clock Pulse Width High |  | — | — | — | — | ns |
| B4 | CLKOUT Rise Time |  | — | — | — | — | ns |
| B5 | CLKOUT Fall Time |  | — | — | — | — | ns |
| B6 | Circuit Parameter TCC |  | 9 | — | 6 | — | — |
| B7 | CLKOUT to A(0:31), RD/$\overline{\text{WR}}$, $\overline{\text{BURST}}$, D(0:31), DP(0:3) Invalid | 0.25TC + 1 | 10 | — | 5 | — | ns |
| B7a | CLKOUT to TSIZ(0:1), $\overline{\text{REG}}$, $\overline{\text{RSV}}$, AT(0:3), $\overline{\text{BDIP}}$, PTR Invalid | 0.25TC + 1 | 10 | — | 5 | — | ns |
| B7b | CLKOUT to $\overline{\text{BR}}$, $\overline{\text{BG}}$, FRZ, VFLS(0:1), VF(0:2), IWP(0:2), LWP(0:1), $\overline{\text{STS}}$ Invalid [1] | 0.25TC + 1 | 10 | — | 5 | — | ns |
| B8 | CLKOUT to A(0:31), RD/$\overline{\text{WR}}$, $\overline{\text{BURST}}$, D(0:31), DP(0:3) Valid | 0.25TC + TCC | 10 | 19 | — | 13 | ns |
| B8a | CLKOUT to TSIZ(0:1), $\overline{\text{REG}}$, $\overline{\text{RSV}}$, AT(0:3), $\overline{\text{BDIP}}$, PTR Valid | 0.25TC + TCC | 10 | 19 | — | 13 | ns |
| B8b | CLKOUT to $\overline{\text{BR}}$, $\overline{\text{BG}}$, VFLS(0:1), VF(0:2), IWP(0:2), FRZ, LWP(0:1), $\overline{\text{STS}}$ Valid. [1] | 0.25TC + TCC | 10 | 19 | — | 13 | ns |
| B9 | CLKOUT to A(0:31), RD/$\overline{\text{WR}}$, $\overline{\text{BURST}}$, D(0:31), DP(0:3), TSIZ(0:1), $\overline{\text{REG}}$, $\overline{\text{RSV}}$, AT(0:3), PTR High-Z | 0.25TC + TCC | 10 | 19 | 5 | 13 | ns |
| B10 |  |  | — | — | — | — | ns |
| B11 | CLKOUT to $\overline{\text{TS}}$, $\overline{\text{BB}}$ Assertion | 0.25TC + TCC | 10 | 19 | 5 | 13 | ns |
| B11a | CLKOUT to $\overline{\text{TA}}$, $\overline{\text{BI}}$ Assertion (When Driven by the Memory Controller or PCMCIA Interface) |  | — | 11 | — | 10 | ns |
| B12 | CLKOUT to $\overline{\text{TS}}$, $\overline{\text{BB}}$ Negation | 0.25TC + TCC | 10 | 19 | 5 | 13 | ns |
| B12a | CLKOUT to $\overline{\text{TA}}$, $\overline{\text{BI}}$ Negation (When Driven by the Memory Controller or PCMCIA Interface) |  | — | 11 | — | 11 | ns |
| B13 | CLKOUT to $\overline{\text{TS}}$, $\overline{\text{BB}}$ High-Z | 0.25TC +14 | 10 | 24 | 5 | 21 | ns |
| B13a | CLKOUT to $\overline{\text{TA}}$, $\overline{\text{BI}}$ High-Z (When Driven by the Memory Controller or PCMCIA Interface) |  | — | 15 | — | 15 | ns |
| B14 | CLKOUT to $\overline{\text{TEA}}$ Assertion |  | — | 11 | — | 11 | ns |
| B15 | CLKOUT to $\overline{\text{TEA}}$ High-Z |  | — | 15 | — | 15 | ns |

### Table 21-1. Bus Operation Timing (Continued)

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| B16 | $\overline{\text{TA}}$, $\overline{\text{BI}}$, $\overline{\text{BB}}$, $\overline{\text{BG}}$, $\overline{\text{BR}}$ Valid to CLKOUT (Setup Time) [2,3] | | 9 | — | 7 | — | ns |
| B16a | $\overline{\text{TEA}}$, $\overline{\text{KR}}$, $\overline{\text{RETRY}}$, $\overline{\text{CR}}$ Valid to CLKOUT (Setup Time) | | 11 | — | 10 | — | ns |
| B17 | CLKOUT to $\overline{\text{TA}}$, $\overline{\text{TEA}}$, $\overline{\text{BI}}$, $\overline{\text{BB}}$, $\overline{\text{BG}}$, $\overline{\text{BR}}$ Valid (Hold Time) [2] | | 2 | — | 2 | — | ns |
| B17a | CLKOUT to $\overline{\text{KR}}$, $\overline{\text{RETRY}}$, $\overline{\text{CR}}$ Valid (Hold Time) | | 2 | — | 2 | — | ns |
| B18 | D(0:31), DP(0:3) Valid to CLKOUT Rising Edge (Setup Time) [4] | | 6 | — | 6 | — | ns |
| B19 | CLKOUT Rising Edge to D(0:31), DP(0:3) Valid (Hold Time) [4] | | 2 | — | 1 | — | ns |
| B20 | D(0:31), DP(0:3) Valid to CLKOUT Falling Edge (Setup Time) [5] | | 4 | — | 4 | — | ns |
| B21 | CLKOUT Falling Edge to D(0:31), DP(0:3) Valid (Hold Time) [5] | | 2 | — | 2 | — | ns |
| B22 | CLKOUT Rising Edge to $\overline{\text{CS}}$ Asserted -GPCM- ACS = 00 | 0.25TC+TCC+1 | 10 | 20 | 5 | 13 | ns |
| B22a | CLKOUT Falling Edge to $\overline{\text{CS}}$ Asserted -GPCM- ACS = 10, TRLX = 0 | TCC + 1 | — | 10 | — | 8 | ns |
| B22b | CLKOUT Falling Edge to $\overline{\text{CS}}$ Asserted -GPCM- ACS = 11, TRLX = 0 | 0.25TC+TCC+1 | 10 | 20 | 5 | 13 | ns |
| B23 | CLKOUT Rising Edge to $\overline{\text{CS}}$ Negated -GPCM- Read Access | TCC + 1 | 3 | 10 | 2 | 8 | ns |
| B24 | A(0:31) to $\overline{\text{CS}}$ Asserted -GPCM- ACS = 10, TRLX = 0 | | 8 | — | 3 | — | ns |
| B24a | A(0:31) to $\overline{\text{CS}}$ Asserted -GPCM- ACS = 11, TRLX = 0 | | 18 | — | 8 | — | ns |
| B25 | CLKOUT Rising Edge to $\overline{\text{OE}}$, $\overline{\text{WE}}$(0:3) Asserted | | — | 11 | — | 9 | ns |
| B26 | CLKOUT Rising Edge to $\overline{\text{OE}}$ Negated | | 3 | 11 | 2 | 9 | ns |
| B27 | A(0:31) to $\overline{\text{CS}}$ Asserted -GPCM- ACS = 10, TRLX = 1 | | 48 | — | 23 | — | ns |
| B27a | A(0:31) to $\overline{\text{CS}}$ Asserted -GPCM- ACS = 11, TRLX = 1 | | 58 | — | 28 | — | ns |
| B28 | CLKOUT Rising Edge to $\overline{\text{WE}}$(0:3) Negated -GPCM- Write Access, CSNT = '0' | | — | 11 | — | 9 | ns |

### Table 21-1. Bus Operation Timing (Continued)

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| B28a | CLKOUT Falling Edge to $\overline{\text{WE}}$(0:3) Negated -GPCM-Write Access, TRLX = ‘0’, CSNT = ‘1’ | 0.25TC + TCC + 1 | 10 | 20 | 5 | 13 | ns |
| B28b | CLKOUT Falling Edge to $\overline{\text{CS}}$ Negated -GPCM-Write Access, TRLX = ‘0’, CSNT = ‘1’, ACS = ‘10’, or ACS=’11’ | 0.25TC + TCC + 1 | — | 20 | — | 13 | ns |
| B29 | $\overline{\text{WE}}$(0:3) Negated to D(0:31), DP(0:3) High-Z -GPCM-Write Access, CSNT = ‘0’ | | 8 | — | 3 | — | ns |
| B29a | $\overline{\text{WE}}$(0:3) Negated to D(0:31), DP(0:3) High-Z -GPCM-Write Access, TRLX = ‘0’, CSNT = ‘1’ | | 18 | — | 8 | — | ns |
| B29b | $\overline{\text{CS}}$ Negated to D(0:31), DP(0:3) High-Z -GPCM-Write Access, ACS = ‘00’, TRLX = ‘0’, and CSNT = ‘0’ | | 8 | — | 3 | — | ns |
| B29c | $\overline{\text{CS}}$ Negated to D(0:31), DP(0:3) High-Z -GPCM-Write Access, TRLX = ‘0’, CSNT = ‘1’, ACS = ‘10’, or ACS=’11’ | | 18 | — | 8 | — | ns |
| B29d | $\overline{\text{WE}}$(0:3) Negated to D(0:31), DP(0:3) High-Z -GPCM-Write Access, TRLX = ‘1’, CSNT = ‘1’ | | 58 | — | 28 | — | ns |
| B29e | $\overline{\text{CS}}$ Negated to D(0:31), DP(0:3) High-Z -GPCM-Write Access, TRLX = ‘1’, CSNT = ‘1’, ACS = ‘10’, or ACS=’11’ | | 58 | — | 28 | — | ns |
| B30 | $\overline{\text{CS}}$, $\overline{\text{WE}}$(0:3) Negated to A(0:31) Invalid -GPCM-Write Access [6] | | 8 | — | 3 | — | — |
| B30a | $\overline{\text{WE}}$(0:3) Negated to A(0:31) Invalid -GPCM-Write Access, TRLX=’0’, CSNT = ‘1’<br>$\overline{\text{CS}}$ negated to A(0:31) Invalid -GPCM- Write Access, TRLX=’0’, CSNT = ‘1’, ACS = ‘10’, ACS = =’11’ | | 18 | — | 8 | — | ns |
| B30b | $\overline{\text{WE}}$(0:3) Negated to A(0:31) Invalid -GPCM-Write Access, TRLX=’1’, CSNT = ‘1’<br>$\overline{\text{CS}}$ Negated to A(0:31) Invalid -GPCM- Write Access, TRLX=’1’, CSNT = ‘1’, ACS = ‘10’, ACS = =’11’ | | 58 | — | 28 | — | ns |
| B31 | CLKOUT Falling Edge to $\overline{\text{CS}}$ Valid - As Requested by Control Bit CST4 in the Corresponding Word in the UPM | TCC + 1 | — | 10 | — | 8 | ns |
| B31a | CLKOUT Falling Edge to $\overline{\text{CS}}$ Valid - As Requested by Control Bit CST1 in the Corresponding Word in the UPM | 0.25TC + TCC + 1 | 10 | 20 | 5 | 13 | ns |
| B31b | CLKOUT Rising Edge to $\overline{\text{CS}}$ Valid - As Requested by Control Bit CST2 in the Corresponding Word in the UPM | TCC + 1 | — | 10 | — | 8 | ns |

### Table 21-1. Bus Operation Timing (Continued)

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| B31c | CLKOUT Rising Edge to $\overline{CS}$ Valid - As Requested by Control Bit CST3 in the Corresponding Word in the UPM | 0.25TC + TCC + 1 | 10 | 20 | 5 | 13 | ns |
| B32 | CLKOUT Falling Edge to $\overline{BS}$ Valid - As Requested by Control Bit BST4 in the Corresponding Word in the UPM | TCC + 1 | — | 10 | — | 8 | ns |
| B32a | CLKOUT Falling Edge to $\overline{BS}$ Valid - As Requested by Control Bit BST1 in the Corresponding Word in the UPM | 0.25TC + TCC + 1 | 10 | 20 | 5 | 13 | ns |
| B32b | CLKOUT Rising Edge to $\overline{BS}$ Valid - As Requested by Control Bit BST2 in the Corresponding Word in the UPM | TCC + 1 | — | 10 | — | 8 | ns |
| B32c | CLKOUT Rising Edge to $\overline{BS}$ Valid - As Requested by Control Bit BST3 in the Corresponding Word in the UPM | 0.25TC + TCC + 1 | 10 | 20 | 5 | 13 | ns |
| B33 | CLKOUT Falling Edge to $\overline{GPL}$ Valid - As Requested by Control Bit GxT4 in the Corresponding Word in the UPM | TCC + 1 | — | 10 | — | 8 | ns |
| B33a | CLKOUT Rising Edge to $\overline{GPL}$ Valid - As Requested by Control Bit GxT3 in the Corresponding Word in the UPM | 0.25TC + TCC + 1 | 10 | 20 | 5 | 13 | ns |
| B34 | A(0:31) to $\overline{CS}$ Valid - As Requested by Control Bit $\overline{CST4}$ in the Corresponding Word in the UPM | | 8 | — | 3 | — | ns |
| B34a | A(0:31) to $\overline{CS}$ Valid - As Requested by Control Bit $\overline{CST1}$ in the Corresponding Word in the UPM | | 18 | — | 8 | — | ns |
| B34b | A(0:31) to $\overline{CS}$ Valid - As Requested by Control Bit $\overline{CST2}$ in the Corresponding Word in the UPM | | 28 | — | 13 | — | ns |
| B35 | A(0:31) to $\overline{BS}$ Valid - As Requested by Control Bit BST4 in the Corresponding Word in the UPM | | 8 | — | 3 | — | ns |
| B35a | A(0:31) to $\overline{BS}$ Valid - As Requested by Control Bit BST1 in the Corresponding Word in the UPM | | 18 | — | 8 | — | ns |
| B35b | A(0:31) to $\overline{BS}$ Valid - As Requested by Control Bit BST2 in the Corresponding Word in the UPM | | 28 | — | 13 | — | ns |
| B36 | A(0:31) to $\overline{GPL}$ Valid - As Requested by Control Bit GxT4 in the Corresponding Word in the UPM | | 8 | — | 3 | — | ns |
| B37 | UPWAIT Valid to CLKOUT Falling Edge [7] | | 6 | — | — | — | ns |
| B38 | CLKOUT Falling Edge to UPWAIT Valid [7] | | 1 | — | — | — | ns |
| B39 | $\overline{AS}$ Valid to CLKOUT Rising Edge [8] | | 9 | — | 7 | — | ns |

## Table 21-1. Bus Operation Timing (Continued)

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| B40 | A(0:31), TSIZ(0:1), RD/$\overline{WR}$, $\overline{BURST}$, Valid to CLKOUT Rising Edge | | 9 | — | 7 | — | ns |
| B41 | $\overline{TS}$ Valid to CLKOUT Rising Edge (Setup Time) | | 9 | — | 7 | — | ns |
| B42 | CLKOUT Rising Edge to $\overline{TS}$ Valid (Hold Time) | | 2 | — | 2 | — | ns |
| B43 | $\overline{AS}$ Negation to Memory Controller Signals Negation | | — | TBD | — | TBD | ns |

NOTES:

1. The timing for $\overline{BR}$ output is relevant when the MPC821 is selected to work with external bus arbiter.
   The timing for $\overline{BG}$ output is relevant when the MPC821 is selected to work with internal bus arbiter.
2. The setup times required for $\overline{TA}$, $\overline{TEA}$ and $\overline{BI}$ are relevant only when they are supplied by an external device (and not when the memory controller or the PCMCIA interface drive them).
3. The timing required for $\overline{BR}$ input is relevant when the MPC821 is selected to work with internal bus arbiter.
   The timing for $\overline{BG}$ input is relevant when the MPC821 is selected to work with external bus arbiter.
4. The D(0:31) and DP(0:3) input timings b18and b19refer to the rising edge of the CLKOUT in which the $\overline{TA}$ input signal is asserted.
5. The D(0:31) and DP(0:3) input timings b20 and b21refer to the falling edge of the CLKOUT. This timing is valid only under control of the UPM in the memory controller.
6. The timing b30refers to $\overline{CS}$ when ACS = '00' and to $\overline{WE}$(0:3) when CSNT = '0'.
7. The signal UPWAIT is considered asynchronous to the CLKOUT and synchronized internally. The timings specified in B37 and b38are specified to enable the freeze of the UPM output signals as described in Figure 21-15.
8. The $\overline{AS}$ signal is considered asynchronous to the CLKOUT. The timing b39is specified to allow the behavior specified in Figure 21-18.

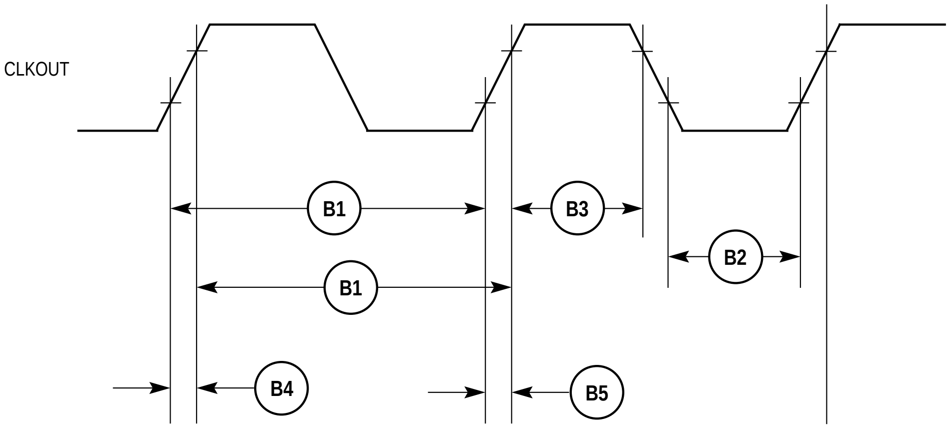


Figure 21-2. External Clock Timing Diagram

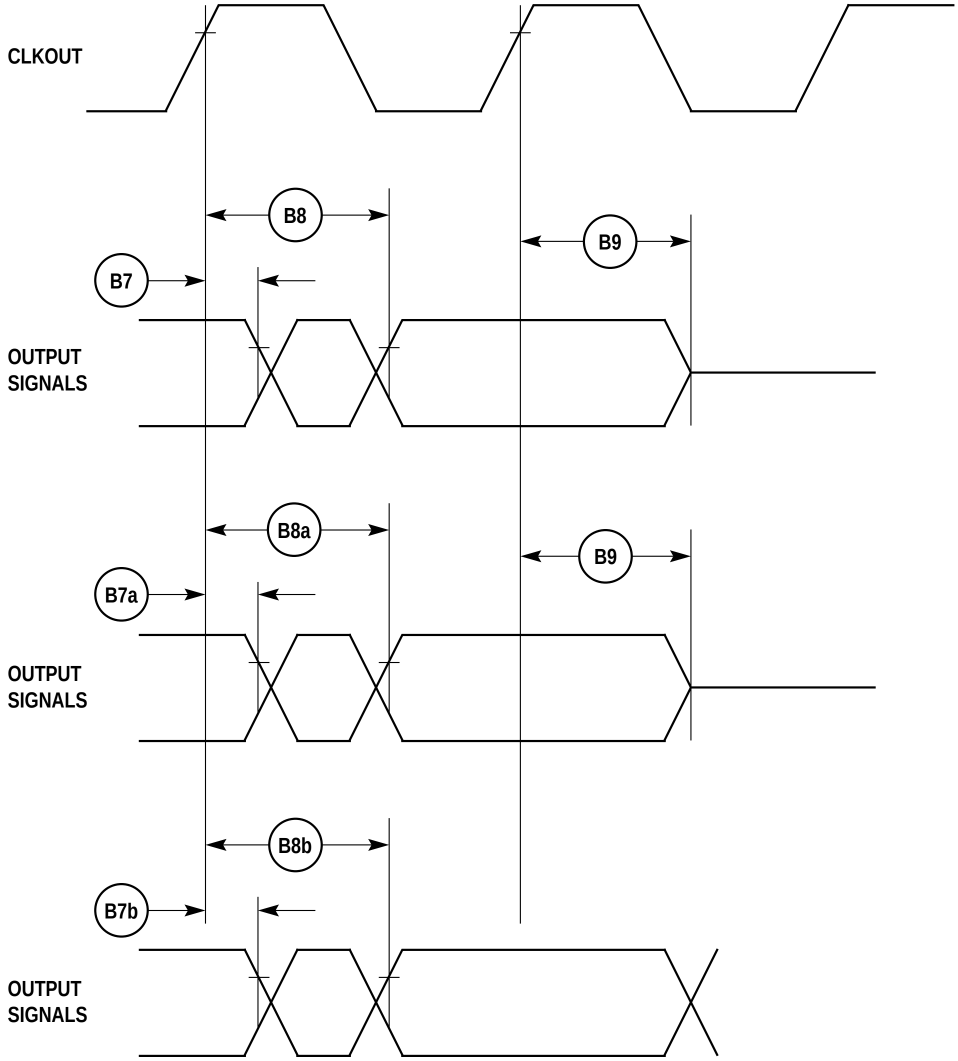


Figure 21-3. Synchronous Output Signals Timing Diagram

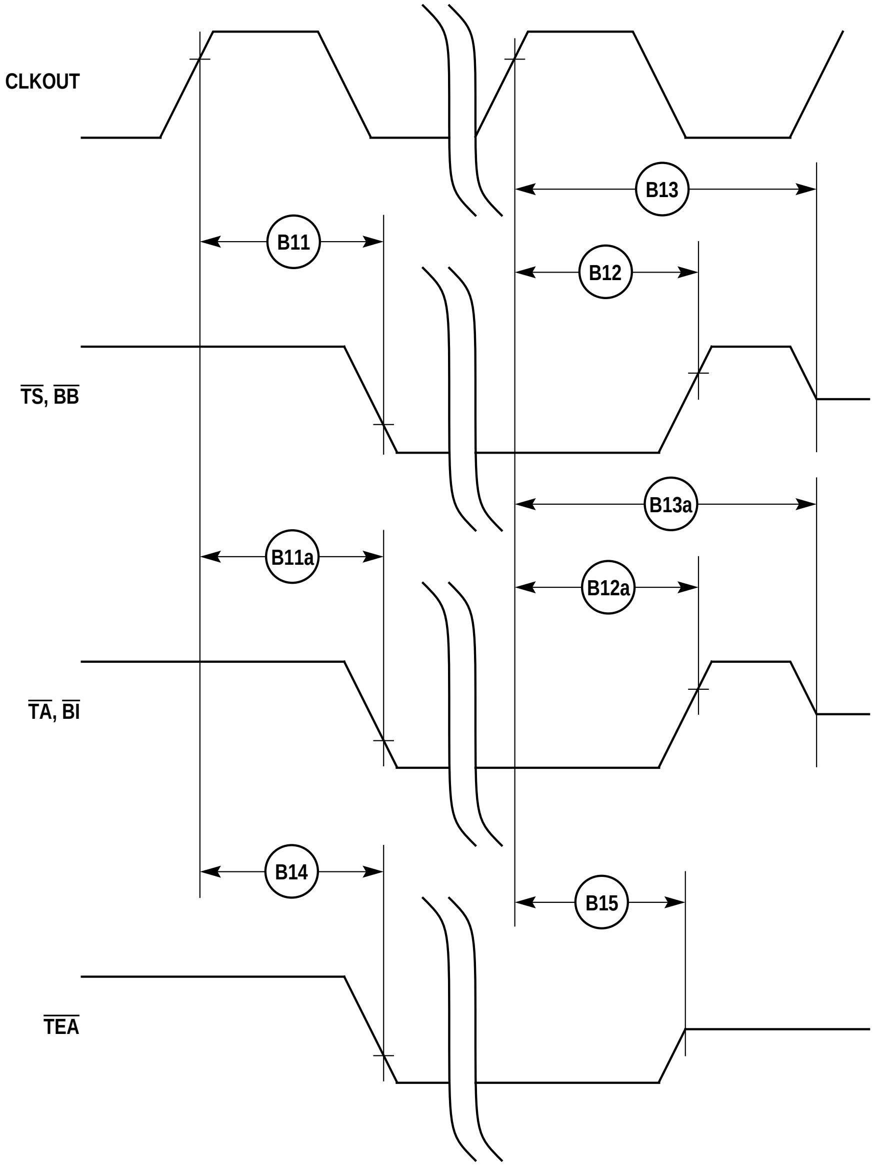


**Figure 21-4. Synchronous Active Pull-Up and Open Drain Outputs Signals Timing Diagram**

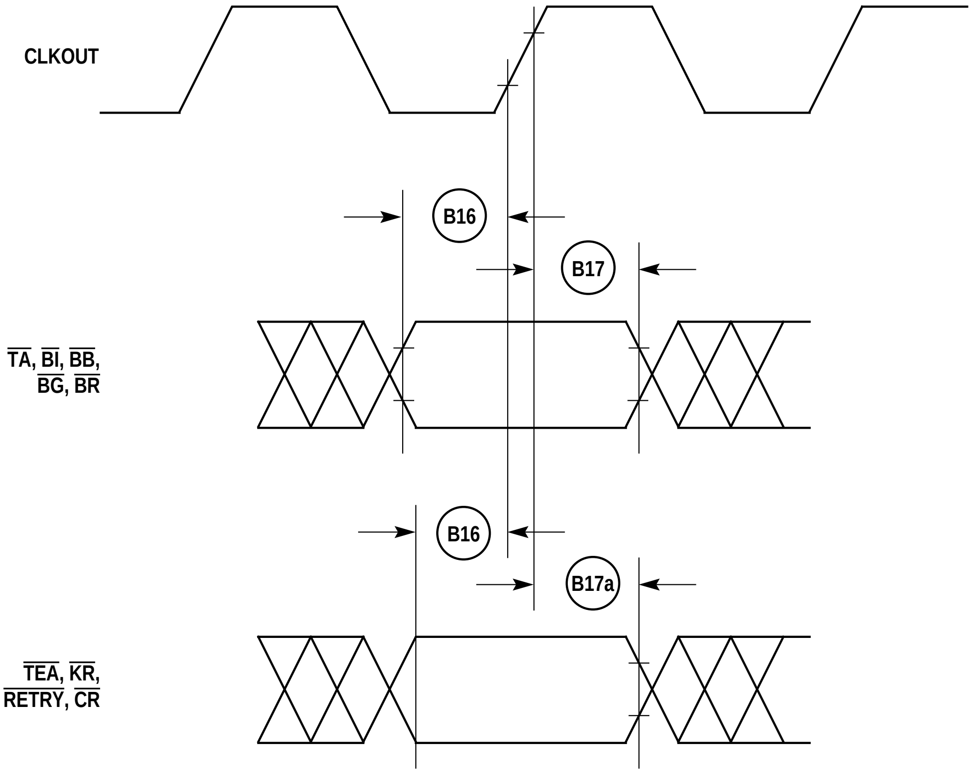


Figure 21-5. Synchronous Input Signals Timing Diagram

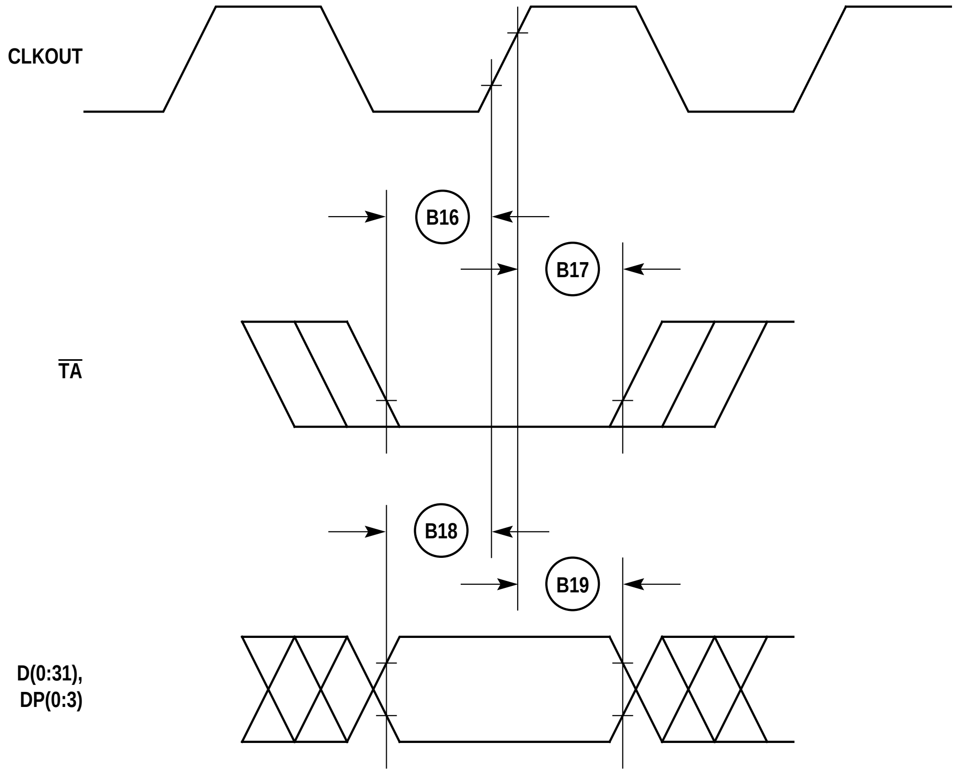


**Figure 21-6. Input Data In Normal Case Timing Diagram**

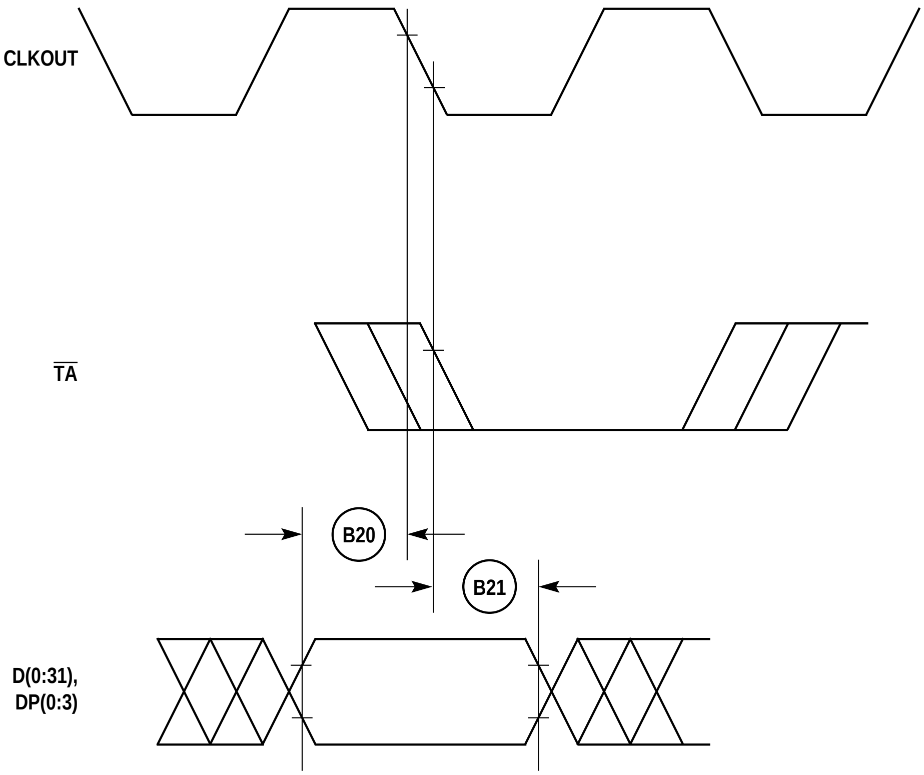


**Figure 21-7. Input Data When Controlled by the UPM in the Memory Controller Timing Diagram**

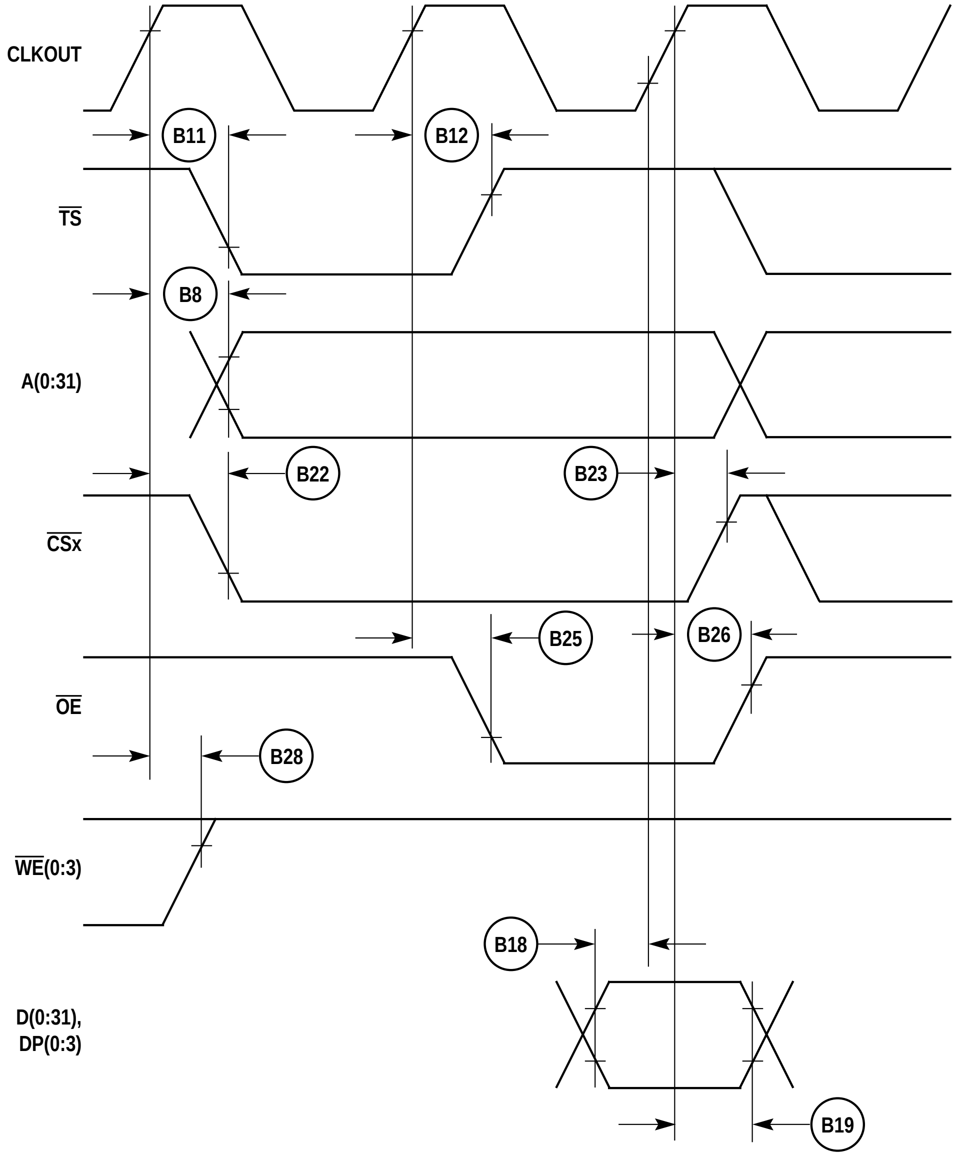


**Figure 21-8. External Bus Read Timing Diagram (GPCM Controlled–ACS = ‘00’)**

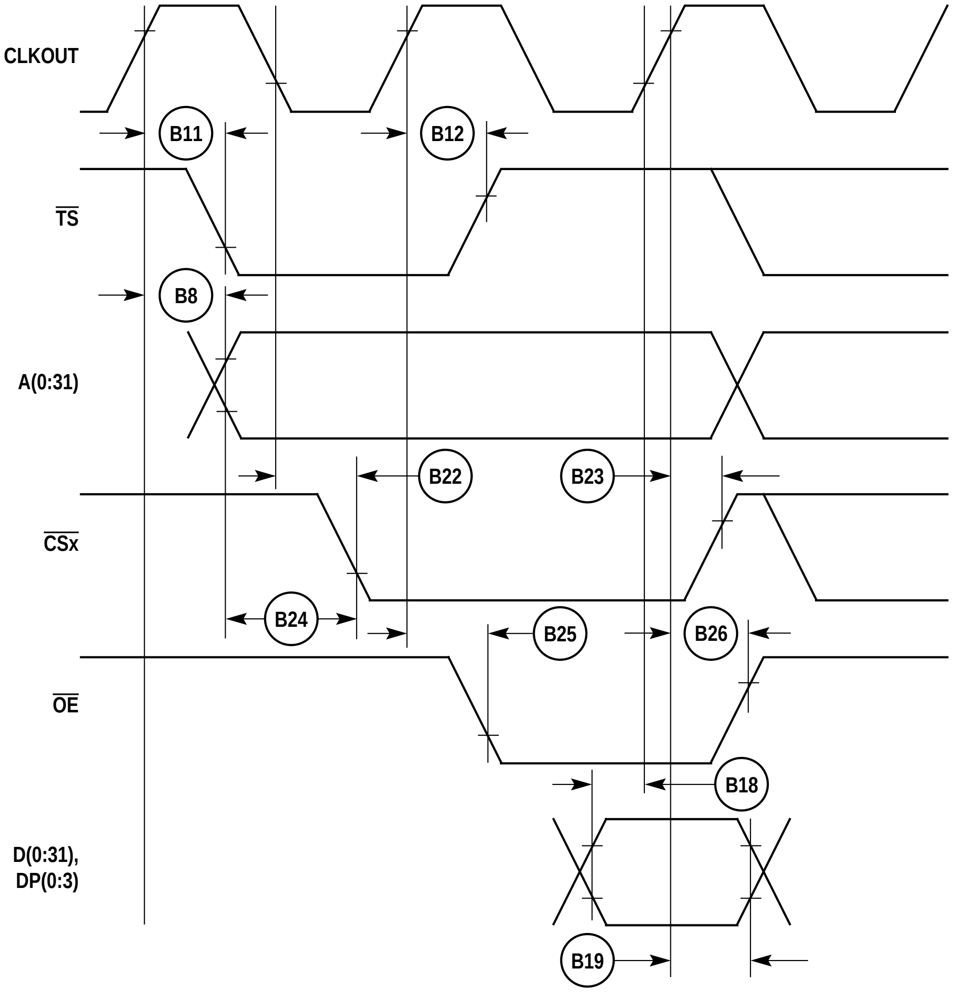


**Figure 21-9. External Bus Read Timing Diagram (GPCM Controlled–TRLX = ‘0’, ACS = ‘10’)**

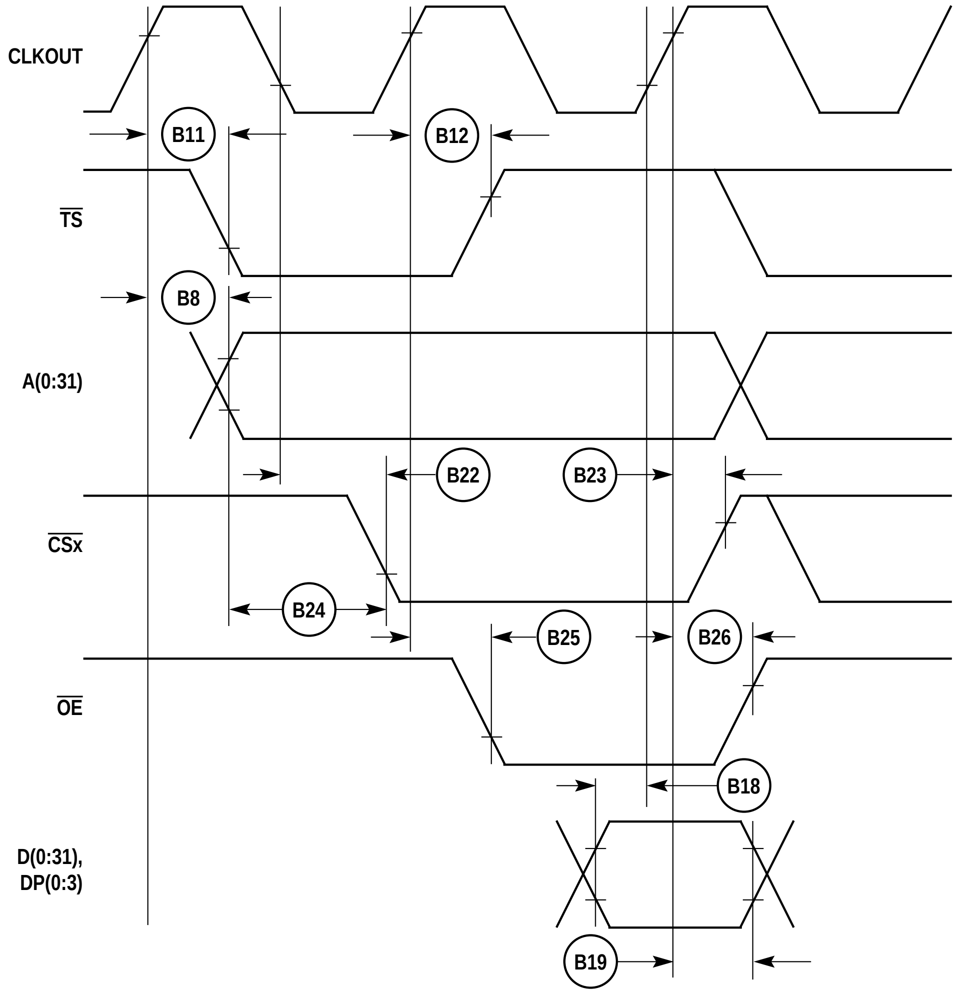


**Figure 21-10. External Bus Read Timing Diagram (GPCM Controlled–TRLX = ‘0’, ACS = ‘11’)**

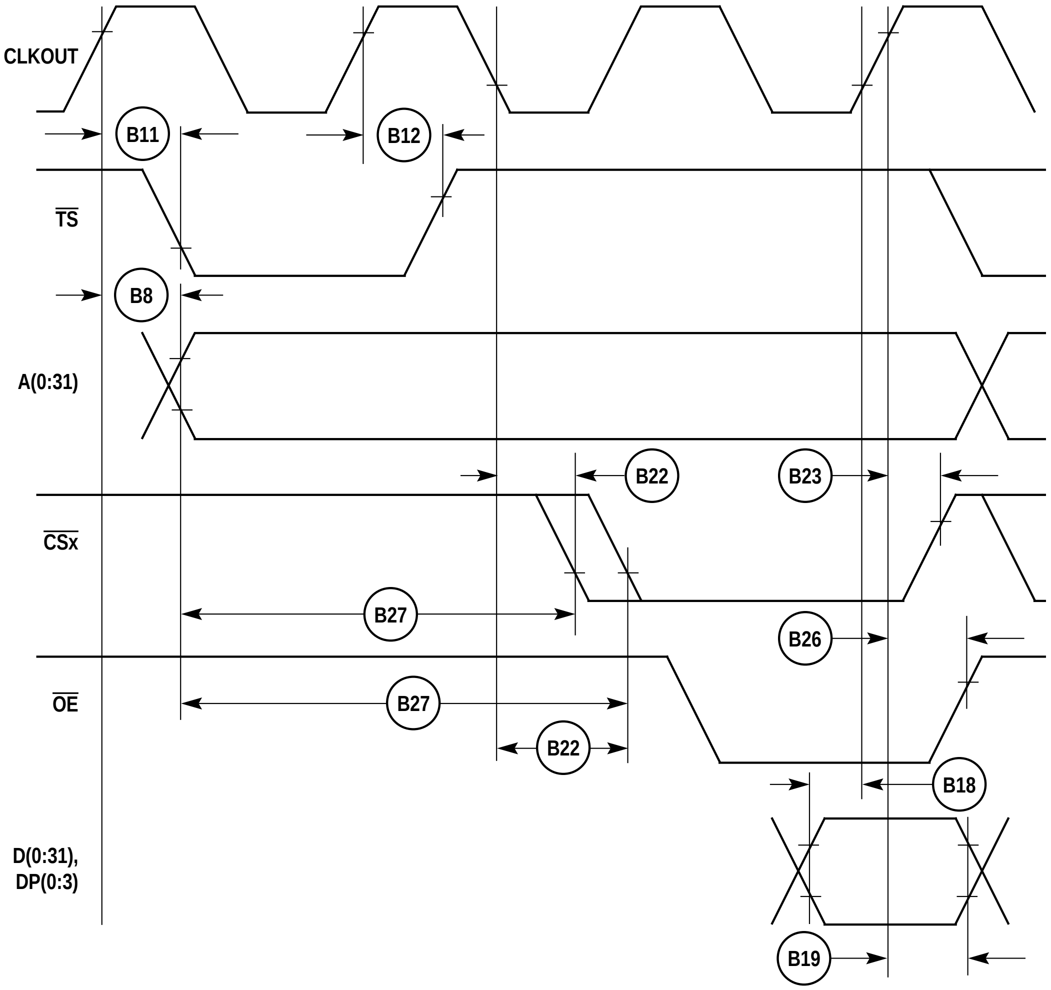


**Figure 21-11. External Bus Read Timing Diagram (GPCM Controlled–TRLX = ‘1’, ACS = ‘10’, ACS = ‘11’)**

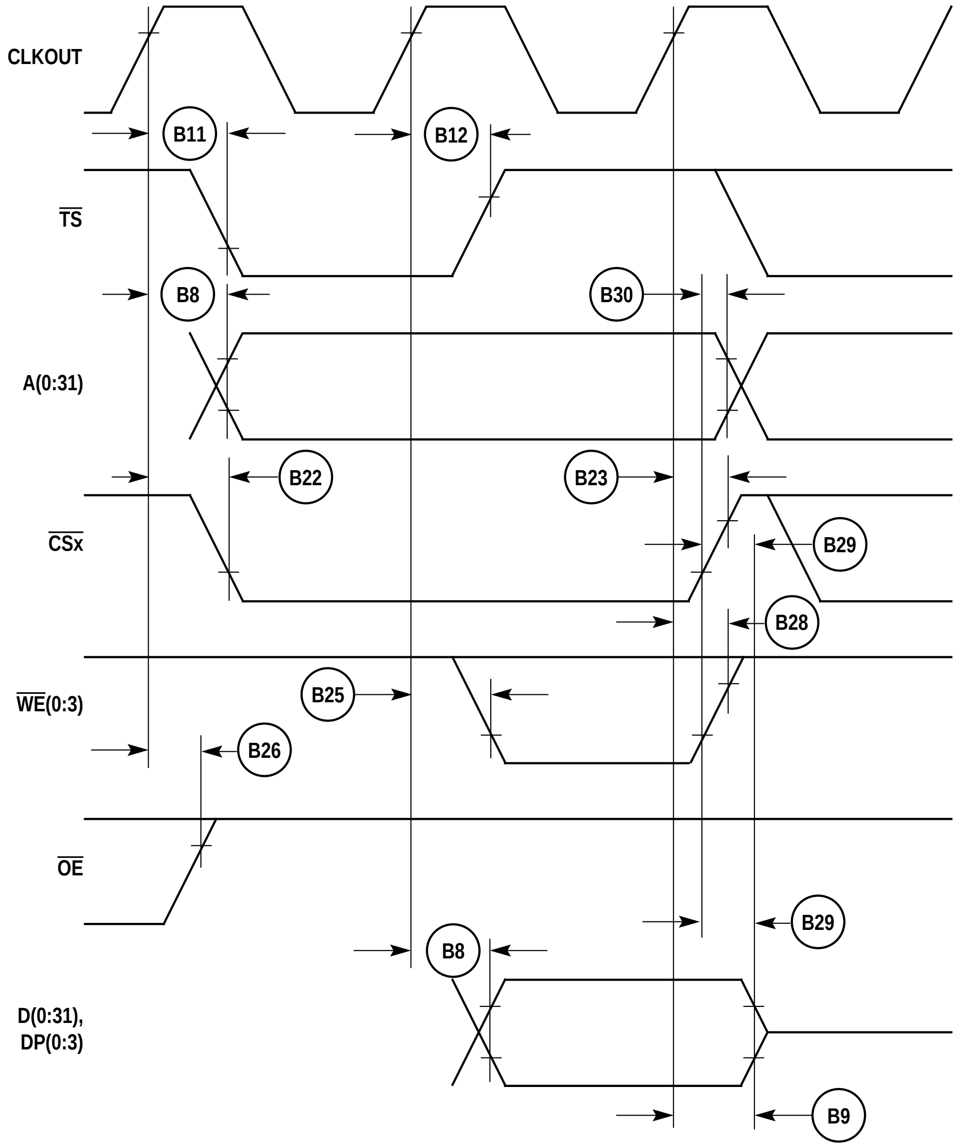


**Figure 21-12. External Bus Write Timing Diagram (GPCM Controlled–TRLX = ‘0’, CSNT = ‘0’)**

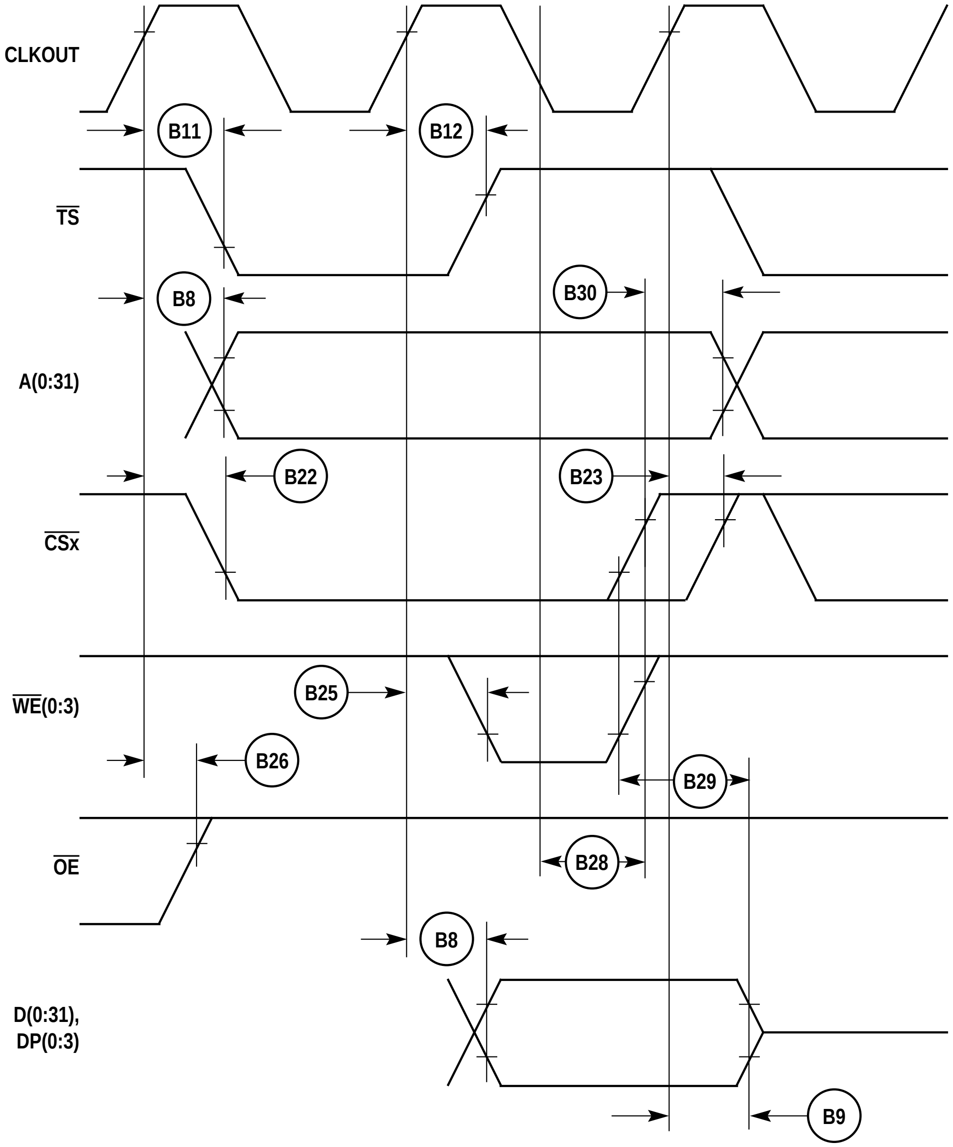


**Figure 21-13. External Bus Write Timing Diagram (GPCM Controlled–TRLX = ‘0’, CSNT = ‘1’)**

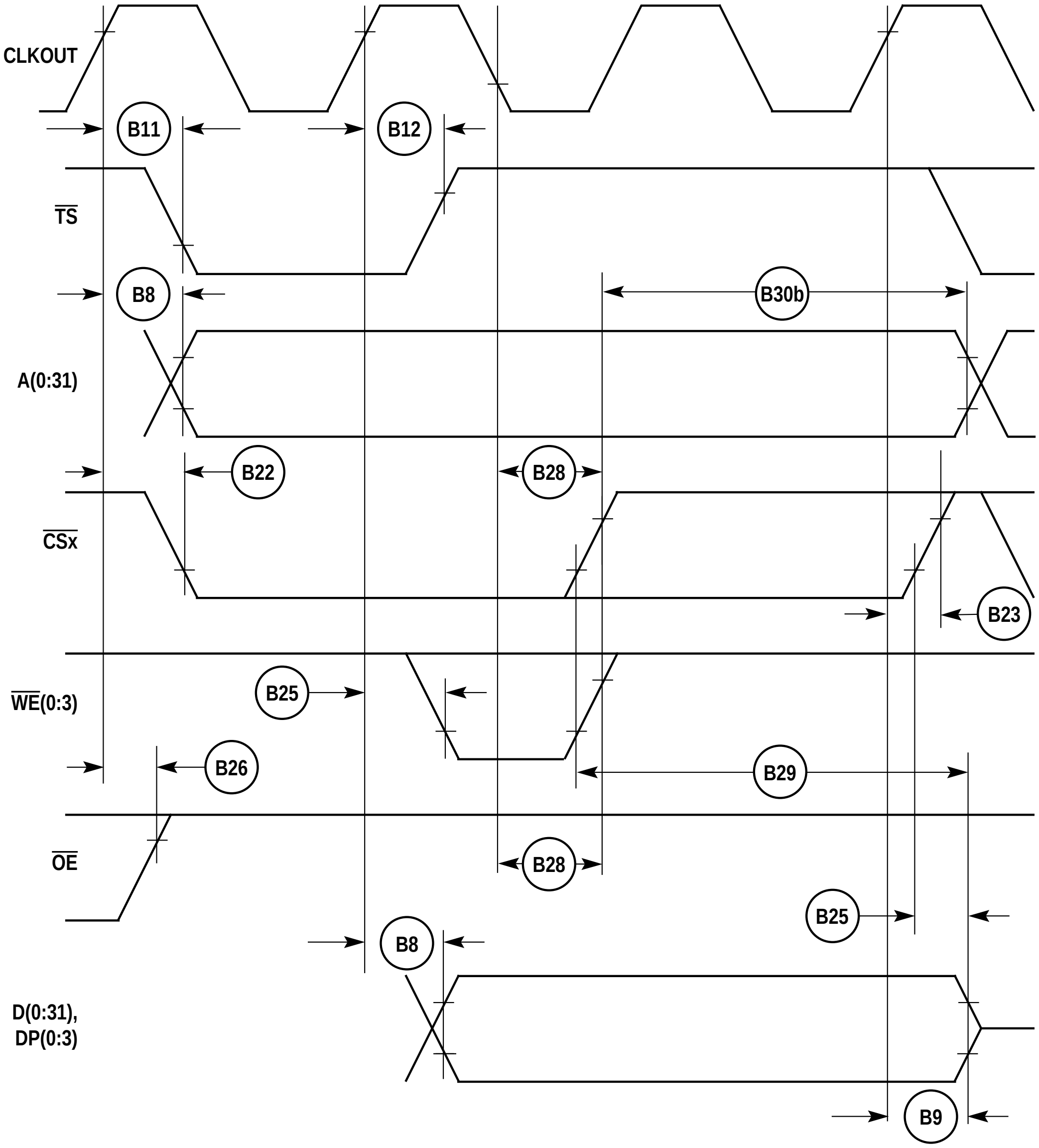


**Figure 21-14. External Bus Write Timing Diagram (GPCM Controlled–TRLX = ‘1’, CSNT = ‘1’)**

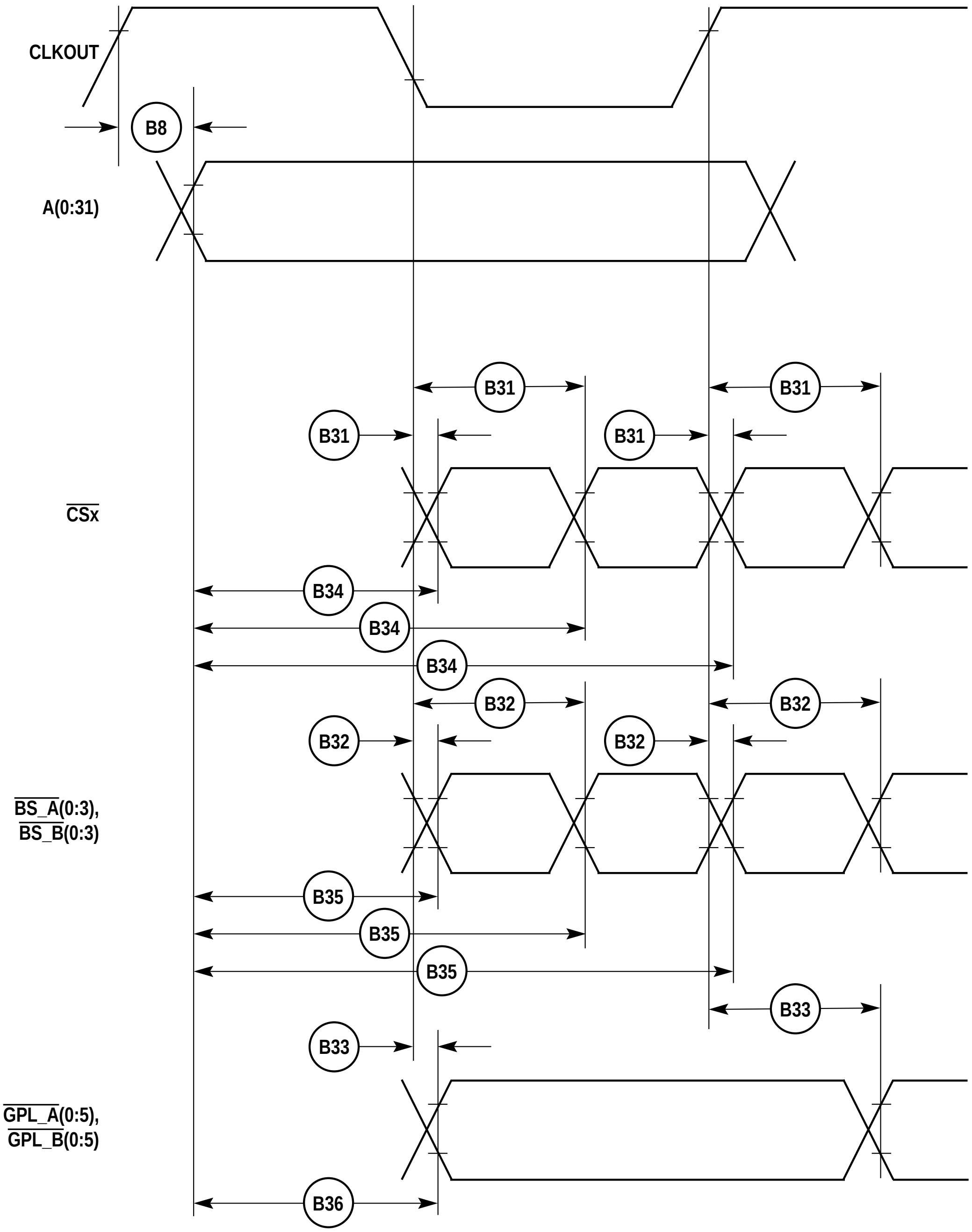


**Figure 21-15. External Bus Timing Diagram (UPM Controlled Signals)**

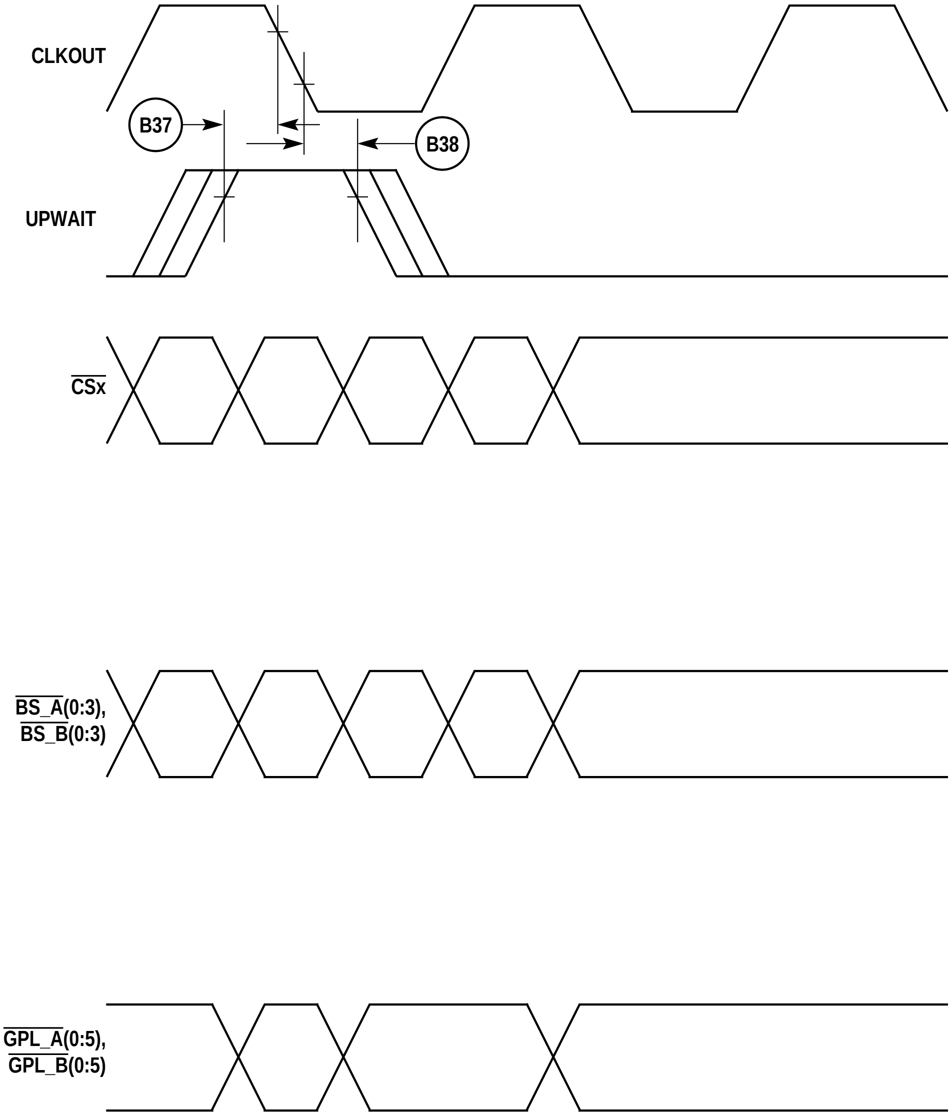


**Figure 21-16. Asynchronous UPWAIT Asserted Detection in UPM Handled Cycles Timing Diagram**

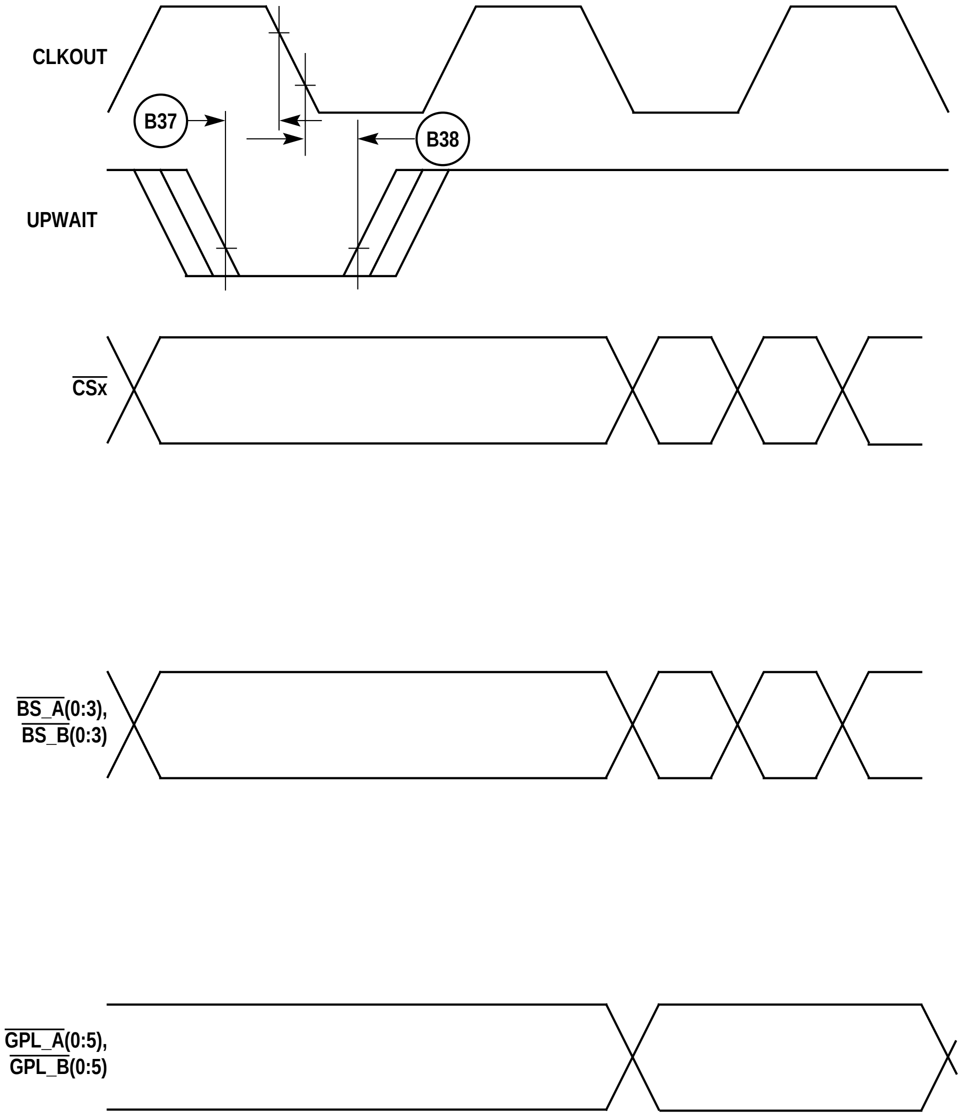

CLKOUT
B37
B38
UPWAIT
$\overline{\text{CSx}}$
$\overline{\text{BS\_A}}$(0:3),
$\overline{\text{BS\_B}}$(0:3)
$\overline{\text{GPL\_A}}$(0:5),
$\overline{\text{GPL\_B}}$(0:5)

**Figure 21-17. Asynchronous UPWAIT Negated Detection in UPM Handled Cycles Timing Diagram**

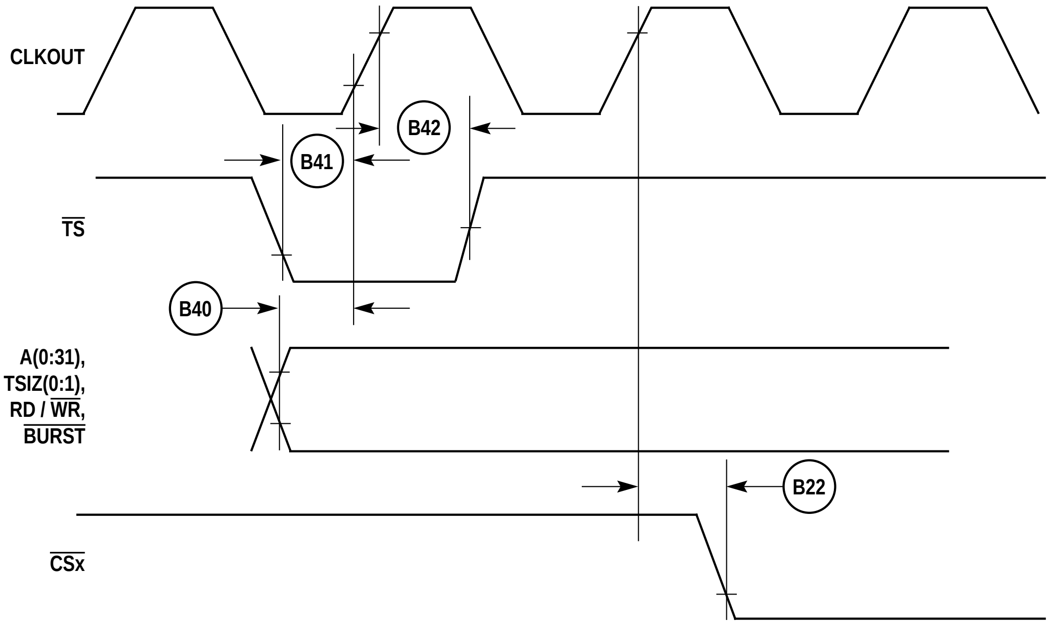


**Figure 21-18. Synchronous External Master Access Timing Diagram (GPCM Handled–ACS = ‘00’)**

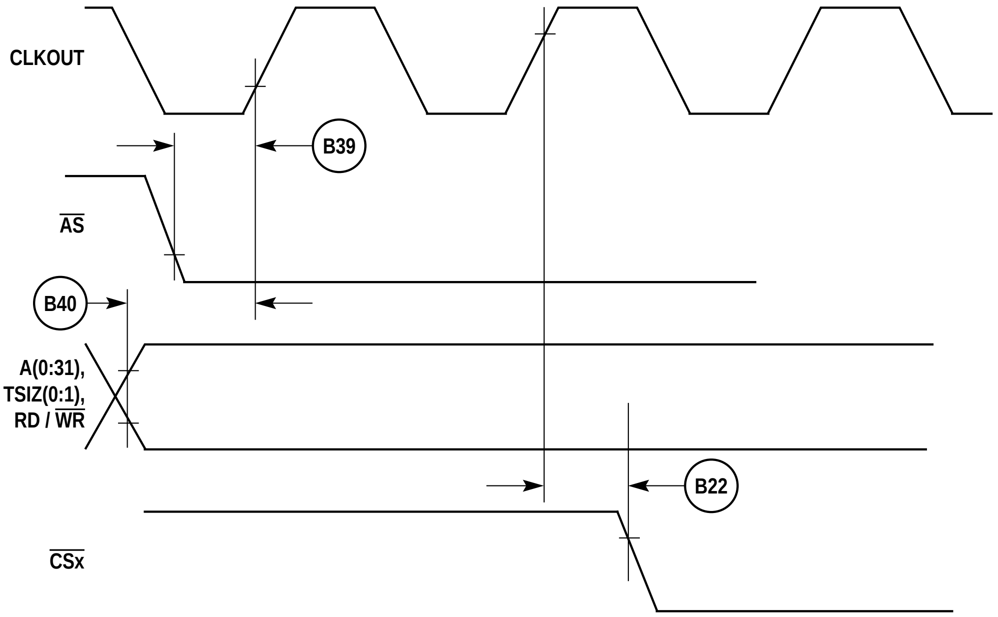


**Figure 21-19. Asynchronous External Master Memory Access Timing Diagram (GPCM Controlled–ACS = ’00’)**

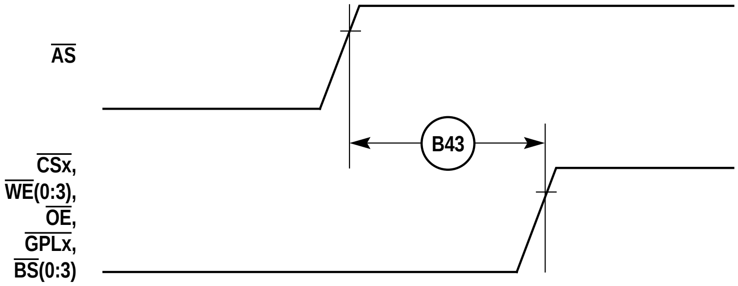


Figure 21-20. Asynchronous External Master Timing Diagram (Control Signals Negation Time)

Table 21-2. Interrupt Timing

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| I39 | IRQx Valid to CLKOUT Rising Edge (Setup Time) [1] | | 6 | — | 6 | — | ns |
| I40 | IRQx Hold Time After CLKOUT [1] | | 2 | — | 2 | — | ns |
| I41 | IRQx Pulse Width Low | | 3 | — | 3 | — | ns |
| I42 | IRQx Pulse Width High | | 3 | — | 3 | — | ns |
| I43 | IRQx Edge to Edge Time | 4*TC | 160 | — | 80 | — | ns |

NOTES: 1. The timings I39 and I40 describe the testing conditions under which the IRQ lines are tested when being defined as level sensitive. The IRQ lines are synchronized internally and do not have to be asserted or negated with reference to the CLKOUT.

2. The timings 141, 142, and 143are specified to allow the correct function of the IRQ lines detection circuitry, and has no direct relation with the total system interrupt latency that the MPC821 is able to support.

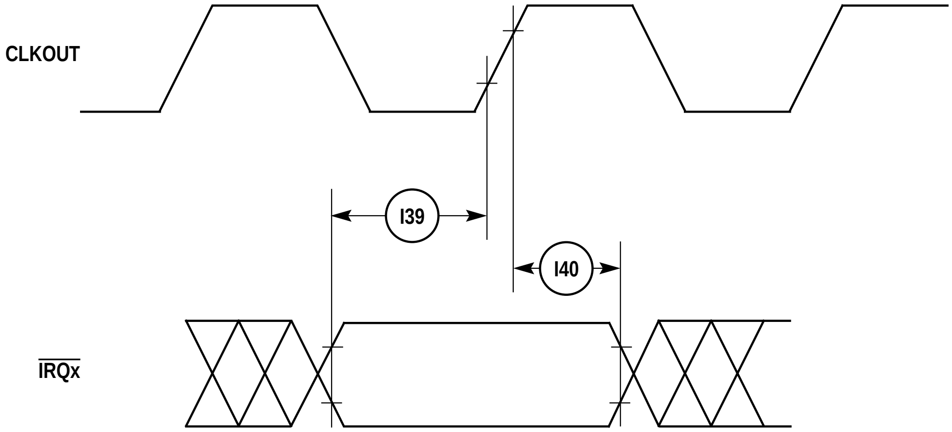


Figure 21-21. Interrupt Detection Timing Diagram
for External Level-Sensitive Lines

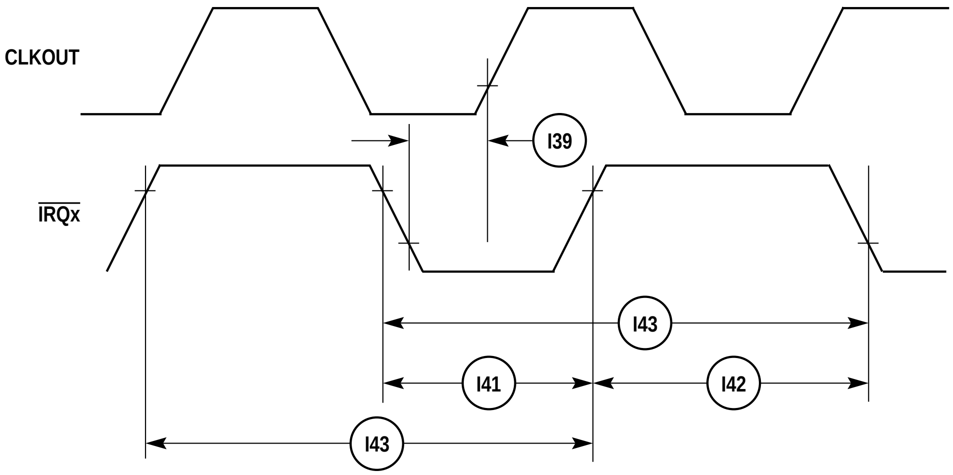


Figure 21-22. Interrupt Detection Timing Diagram
for External Edge-Sensitive Lines

## Table 21-3. PCMCIA Timing

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| P44 | A(0:31), $\overline{\text{REG}}$ Valid to PCMCIA Strobe Asserted [1] | | 28 | — | 13 | — | ns |
| P45 | A(0:31), $\overline{\text{REG}}$ Valid to ALE Negation [1] | | 38 | — | 18 | — | ns |
| P46 | CLKOUT to $\overline{\text{REG}}$ Valid | | — | 19 | 5 | 13 | ns |
| P47 | CLKOUT to $\overline{\text{REG}}$ Invalid | | 11 | — | 6 | — | ns |
| P48 | CLKOUT to $\overline{\text{CE1}}$, $\overline{\text{CE2}}$ Asserted | | 10 | 19 | 5 | 13 | — |
| P49 | CLKOUT to $\overline{\text{CE1}}$, $\overline{\text{CE2}}$ Negated | | 10 | 19 | 5 | 13 | ns |
| P50 | CLKOUT to $\overline{\text{PCOE}}$, $\overline{\text{IORD}}$, $\overline{\text{PCWE}}$, $\overline{\text{IOWR}}$ Assert Time | | — | 12 | — | 11 | ns |
| P51 | CLKOUT to $\overline{\text{PCOE}}$, $\overline{\text{IORD}}$, $\overline{\text{PCWE}}$, $\overline{\text{IOWR}}$ Negate Time | | 3 | 12 | 2 | 11 | ns |
| P52 | CLKOUT to ALE Assert Time | | 10 | 19 | 5 | 13 | ns |
| P53 | CLKOUT to ALE Negate Time | | — | 19 | — | 13 | ns |
| P54 | $\overline{\text{PCWE}}$, $\overline{\text{IOWR}}$ Negated to D(0:31)Invalid [2] | | 8 | — | 3 | — | ns |
| P55 | $\overline{\text{WAITA}}$ and $\overline{\text{WAITB}}$ Valid to CLKOUT Rising Edge [3] | | 8 | — | 8 | — | ns |
| P56 | CLKOUT Rising Edge to $\overline{\text{WAITA}}$ and $\overline{\text{WAITB}}$ Invalid [3] | | 2 | — | 2 | — | ns |

NOTES:
1. PSST = 1. Otherwise add PSST times cycle time.
2. PSHT = 0. Otherwise add PSHT times cycle time.
3. These synchronous timings define when the $\overline{\text{WAITx}}$ signals are detected in order to freeze (or relieve) the PCMCIA current cycle. The $\overline{\text{WAITx}}$ assertion will be effective only if it is detected 2 cycles before the PSL timer expiration. Refer to **Section 12 PCMCIA Interface**.

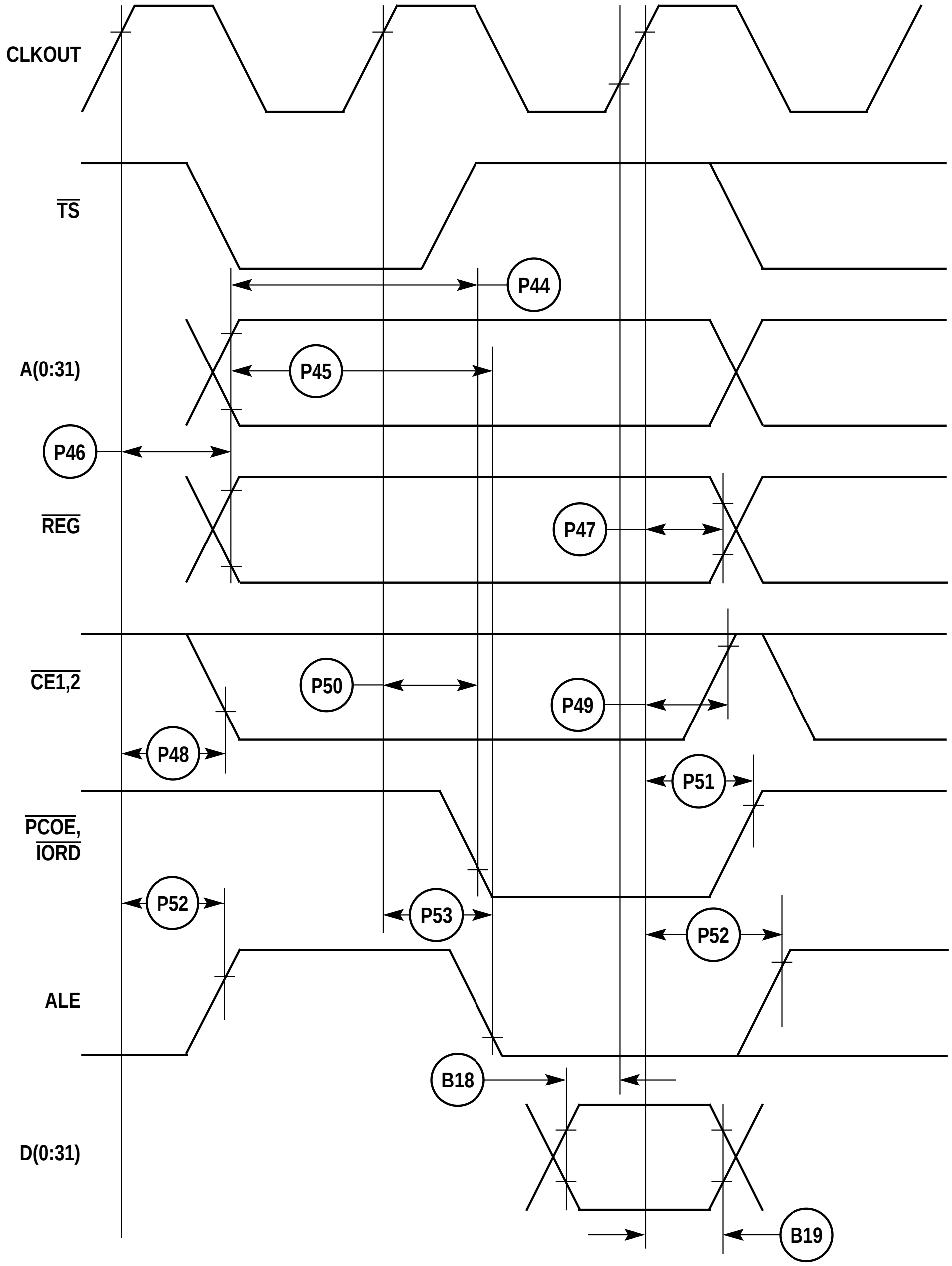


**Figure 21-23. PCMCIA Access Cycles Timing Diagram (External Bus Read)**

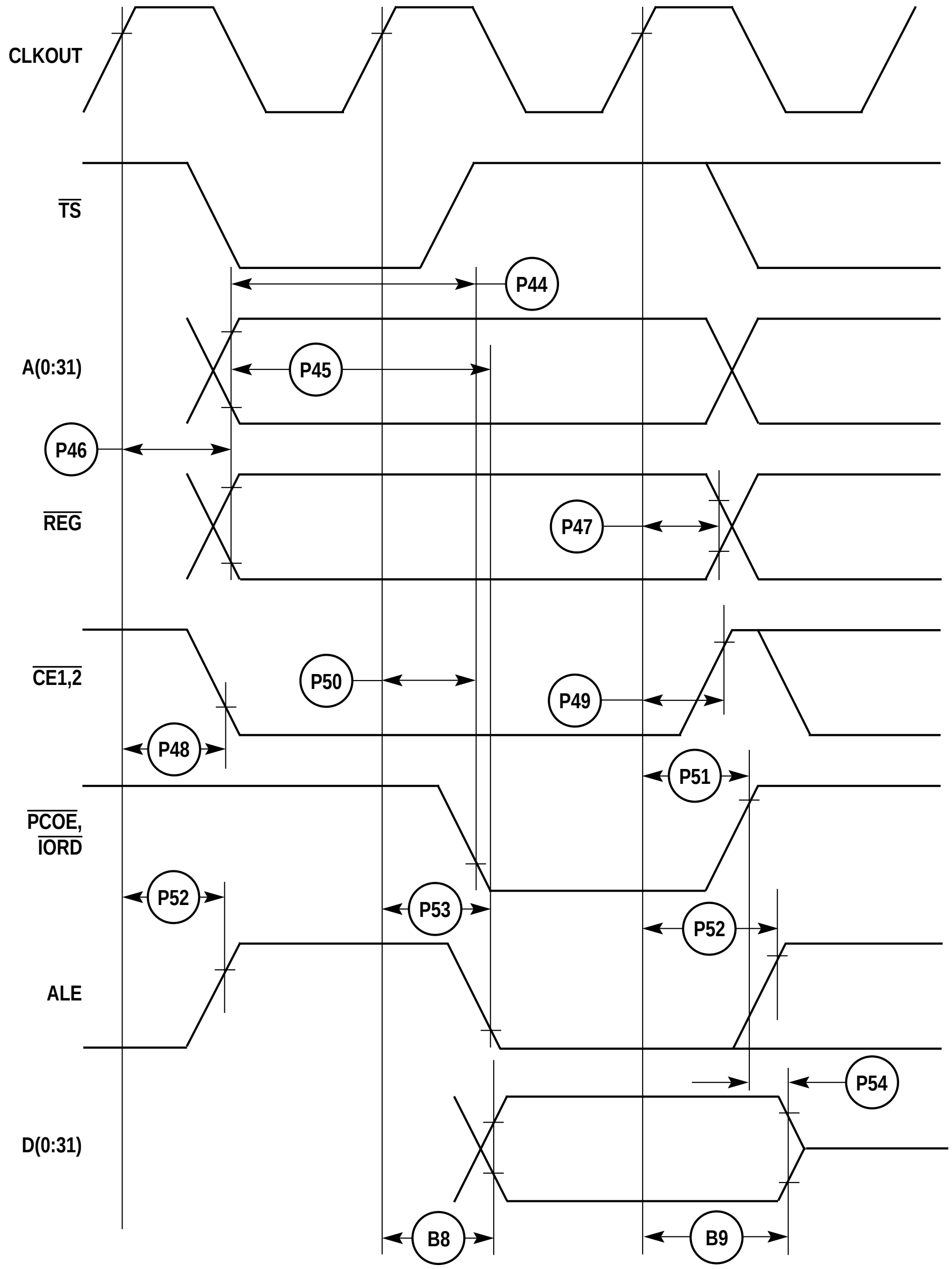


**Figure 21-24. PCMCIA Access Cycles Timing Diagram (External Bus Write)**

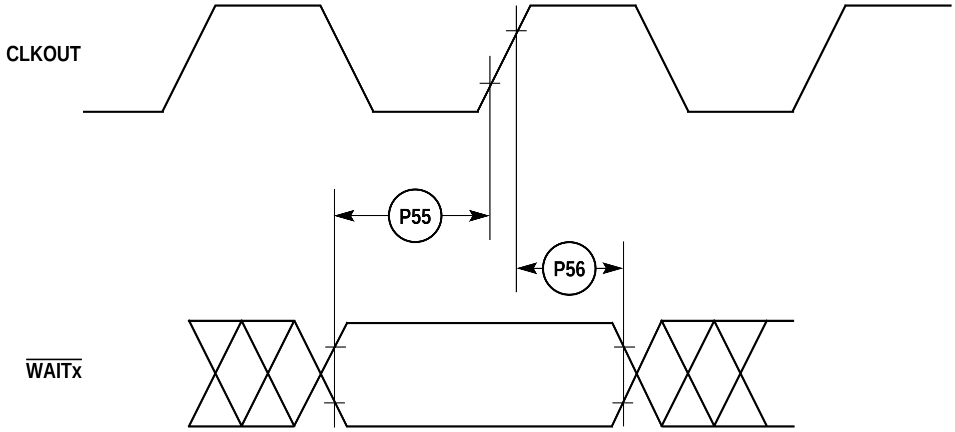


**Figure 21-25. PCMCIA Wait Signals Detection Timing Diagram**

**Table 21-4. PCMCIA Port Timing**

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ |  | 50 MHZ |  | UNIT |
|---|---|---|---|---|---|---|---|
|  |  |  | MIN | MAX | MIN | MAX |  |
| P57 | CLKOUT to OPx Valid |  | — | 25 | — | 19 | ns |
| P58 | HRESET Negated to OPx Drive * |  | 30 | — | 18 | — | ns |
| P59 | IP_Xx Valid to CLKOUT Rising Edge |  | 6 | — | 5 | — | ns |
| P60 | CLKOUT Rising Edge to IP_Xx Invalid |  | 2 | — | 1 | — | ns |

NOTE: * OP2 and OP3 only.

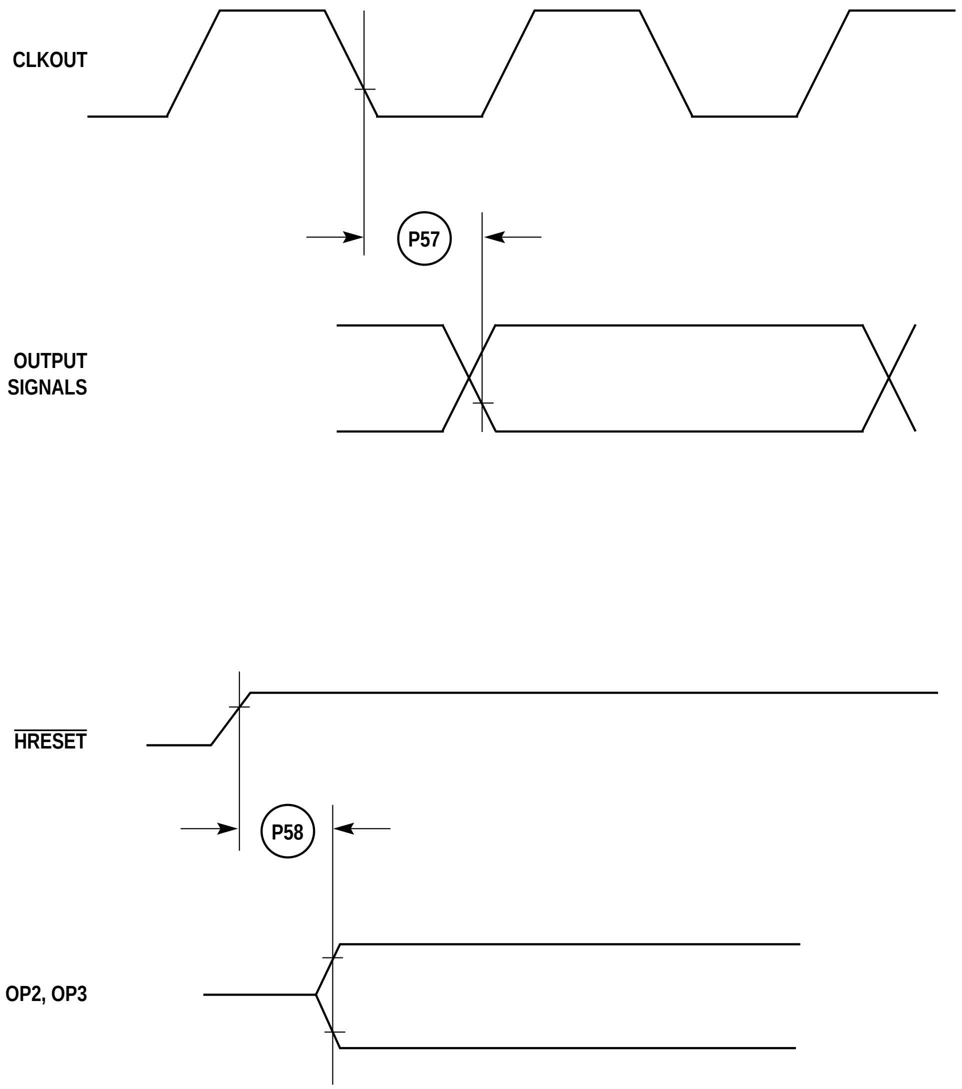


Figure 21-26. PCMCIA Output Port Timing Diagram

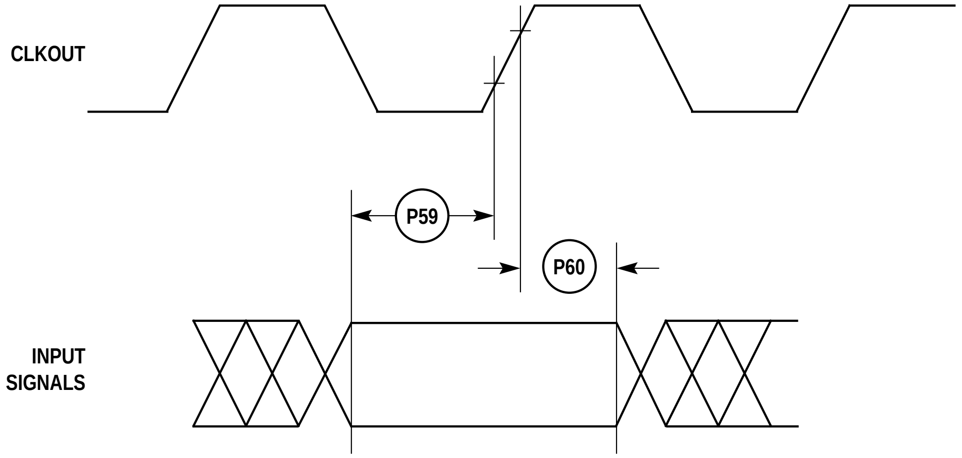


Figure 21-27. PCMCIA Input Port Timing Diagram

Table 21-5. Debug Port Timing

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| D61 | DSCK Cycle Time | | 120 | — | 60 | — | ns |
| D62 | DSCK Clock Pulse Width | | 50 | — | 25 | — | ns |
| D63 | DSCK Rise and Fall Times | | 0 | 3 | 0 | 3 | ns |
| D64 | DSDI Input Data Setup Time | | TBD | — | TBD | — | ns |
| D65 | DSDI Data Hold Time | | TBD | — | TBD | — | ns |
| D66 | DSCK High to DSDO Data Valid | | 0 | TBD | 0 | TBD | ns |
| D67 | DSCK High to DSDO Invalid | | 0 | TBD | 0 | TBD | ns |

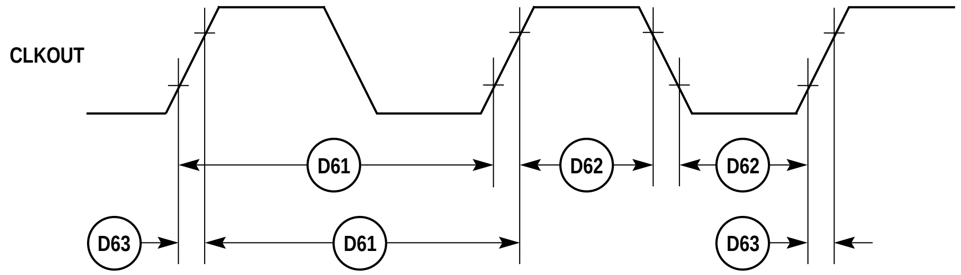


Figure 21-28. Debug Port Clock Input Timing Diagram

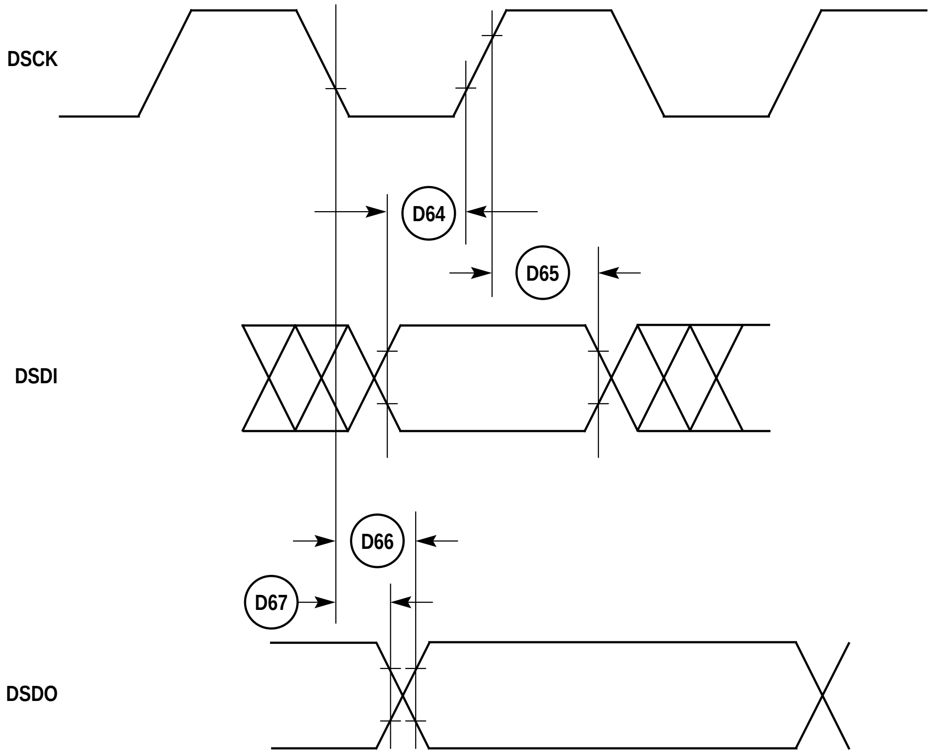


Figure 21-29. Debug Port Timing Diagram

## Table 21-6. RESET Timing

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| R68 | $\overline{\text{HRESET}}$ Setup Time | | TBD | — | TBD | — | ns |
| R69 | CLKOUT to $\overline{\text{HRESET}}$ High Impedance | | — | 20 | — | 20 | ns |
| R70 | CLKOUT to $\overline{\text{SRESET}}$ High Impedance | | — | 20 | — | 20 | ns |
| R71 | $\overline{\text{RSTCONF}}$ Pulse Width | 17*TC | 680 | — | 425 | — | ns |
| R72 | $\overline{\text{RSTCONF}}$ Setup Time | | TBD | — | TBD | — | ns |
| R73 | Configuration Data to $\overline{\text{HRESET}}$ Rising Edge Setup Time | 15*TC+TCC | 650 | — | 425 | — | ns |
| R74 | Configuration Data to $\overline{\text{RSTCONF}}$ Rising Edge Setup Time | | 650 | — | 425 | — | ns |
| R75 | Configuration Data Hold Time After $\overline{\text{RSTCONF}}$ Negation | | 0 | — | 0 | — | ns |
| R76 | Configuration Data Hold Time After $\overline{\text{HRESET}}$ Negation | | 0 | — | 0 | — | ns |
| R77 | $\overline{\text{HRESET}}$ and $\overline{\text{RSTCONF}}$ Asserted to Data Out Drive | | — | 25 | — | 25 | ns |
| R78 | $\overline{\text{RSTCONF}}$ Negated to Data Out High Impedance | | — | 25 | — | 25 | ns |
| R79 | CLKOUT of Last Rising Edge Before Chip Three-States $\overline{\text{HRESET}}$ to Data Out High Impedance | | — | 25 | — | 25 | ns |
| R80 | DSDI, DSCK Setup | 3 TCC | 120 | — | 75 | — | ns |
| R81 | DSDI, DSCK Hold Time | | 0 | — | 0 | — | ns |
| R82 | $\overline{\text{SRESET}}$ Negated to CLKOUT Rising Edge for DSDI and DSCK Sample | 8TCC | 320 | — | 200 | — | ns |

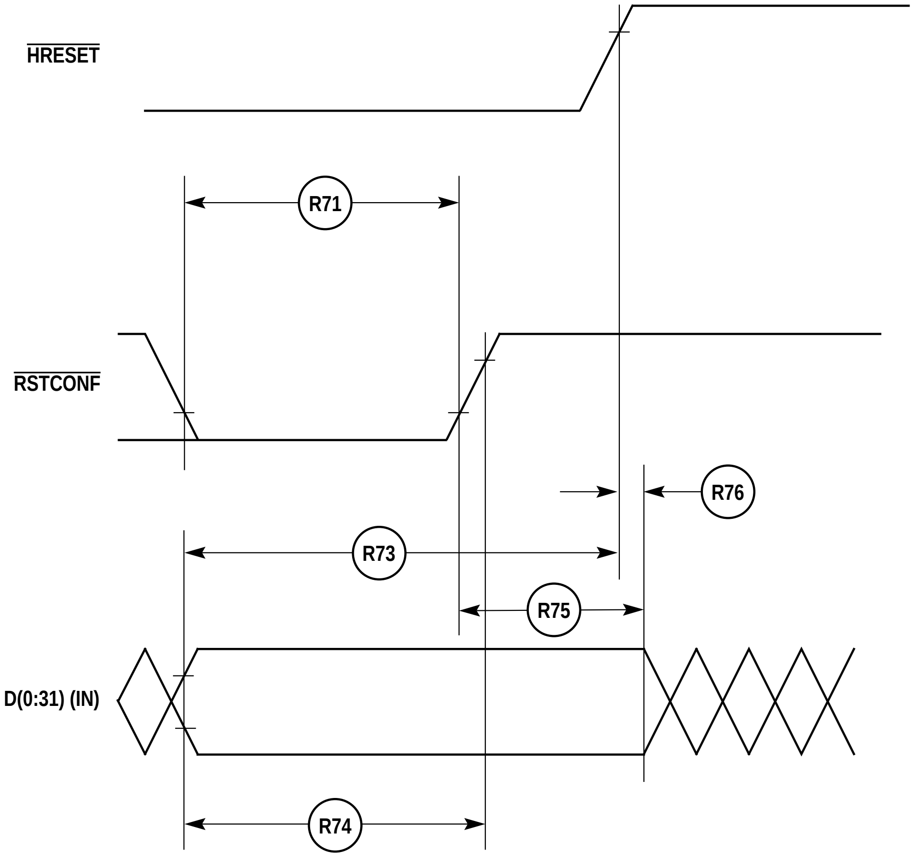


Figure 21-30. Reset Timing Diagram (Configuration from Data Bus)

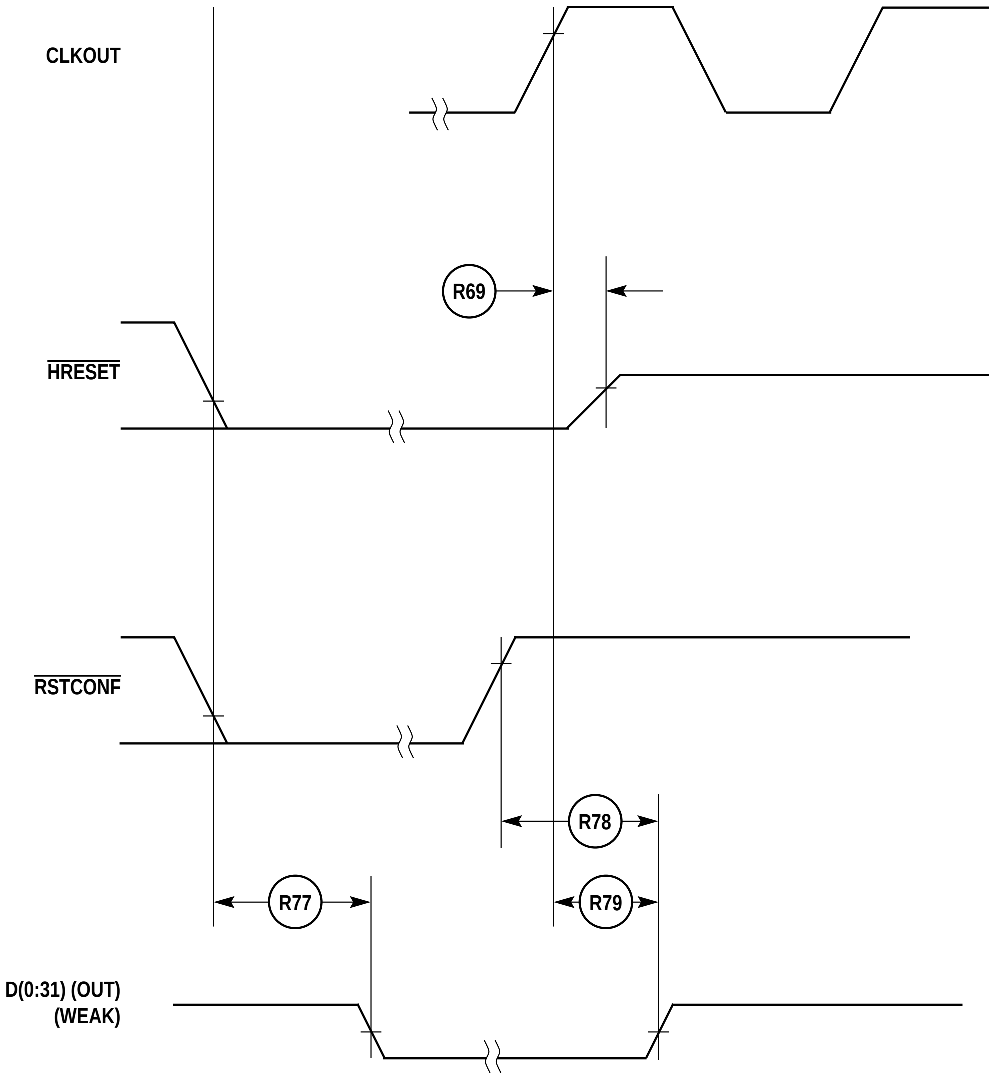


Figure 21-31. Reset Timing Diagram–MPC821 Data Bus Weak Drive During Configuration

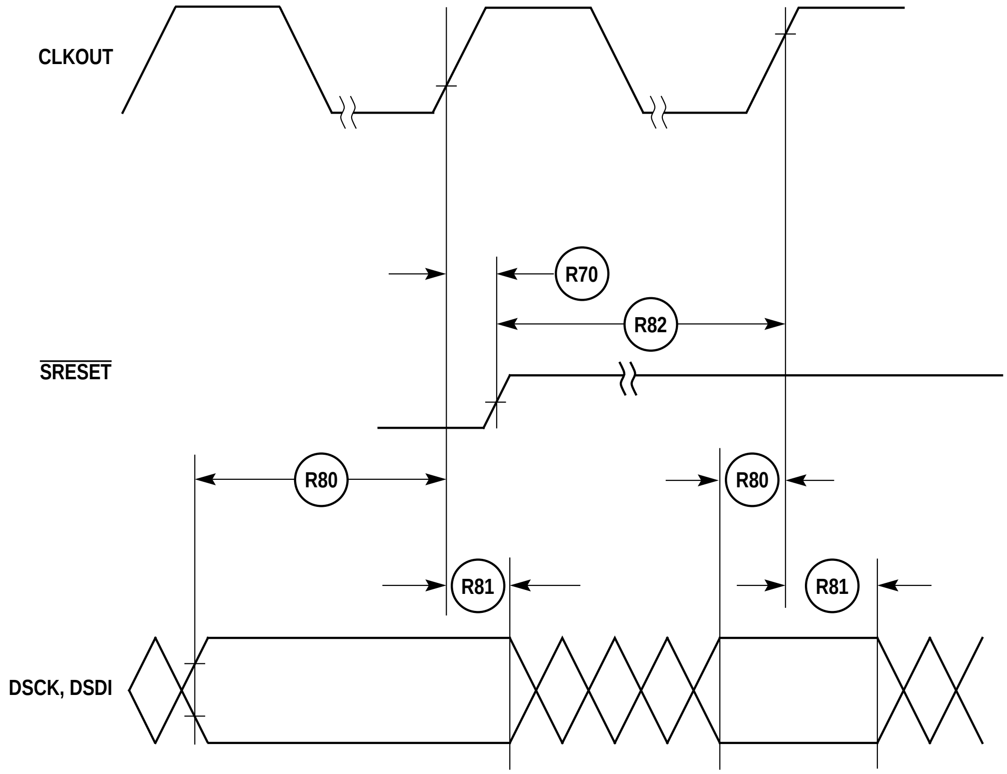


Figure 21-32. Reset Timing Diagram–Debug Port Configuration

## 21.6 IEEE 1149.1 ELECTRICAL SPECIFICATIONS

### Table 21-7. JTAG Timing

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| J82 | TCK Cycle Time | | 100 | — | 100 | — | ns |
| J83 | TCK Clock Pulse Width Measured at 1.5 V | | 40 | — | 40 | — | ns |
| J84 | TCK Rise and Fall Times | | 0 | 10 | 0 | 10 | ns |
| J85 | TMS, TDI Data Setup Time | | 5 | — | 5 | — | ns |
| J86 | TMS, TDI Data Hold Time | | 25 | — | 25 | — | ns |
| J87 | TCK Low to TDO Data Valid | | — | 20 | — | 20 | ns |
| J88 | TCK Low to TDO Data Invalid | | 0 | — | 0 | — | ns |
| J89 | TCK Low to TDO High Impedance | | — | 20 | — | 20 | ns |
| J90 | TRST Assert Time | | 100 | — | 100 | — | ns |
| J91 | TRST Setup Time to TCK Low | | 40 | — | 40 | — | ns |
| J92 | TCK Falling Edge to Output Valid | | — | 50 | — | 50 | ns |
| J93 | TCK Falling Edge to Output Valid Out of High Impedance | | — | 50 | — | 50 | ns |
| J94 | TCK Falling Edge to Output High Impedance | | — | 50 | — | 50 | ns |
| J95 | Boundary Scan Input Valid to TCK Rising Edge | | 50 | — | 50 | — | ns |
| J96 | TCK Rising Edge to Boundary Scan Input Invalid | | 50 | — | 50 | — | ns |

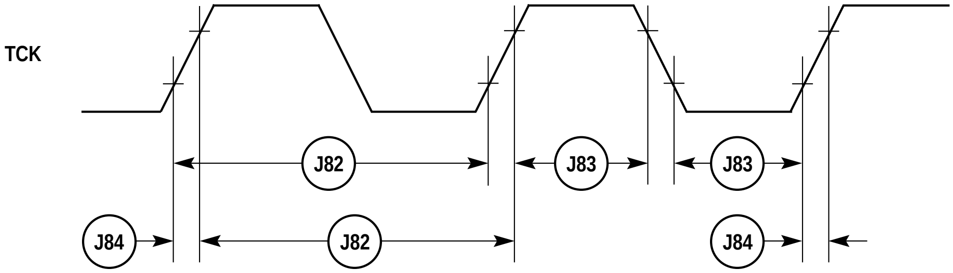


Figure 21-33. JTAG Test Clock Input Timing Diagram

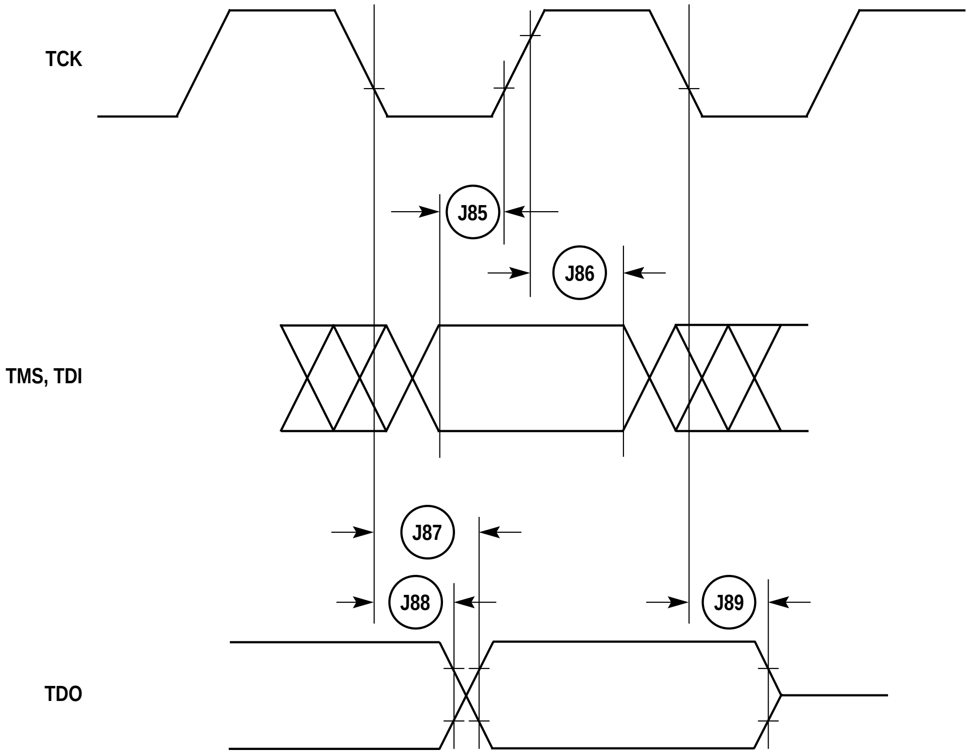


Figure 21-34. JTAG–Test Access Port Timing Diagram

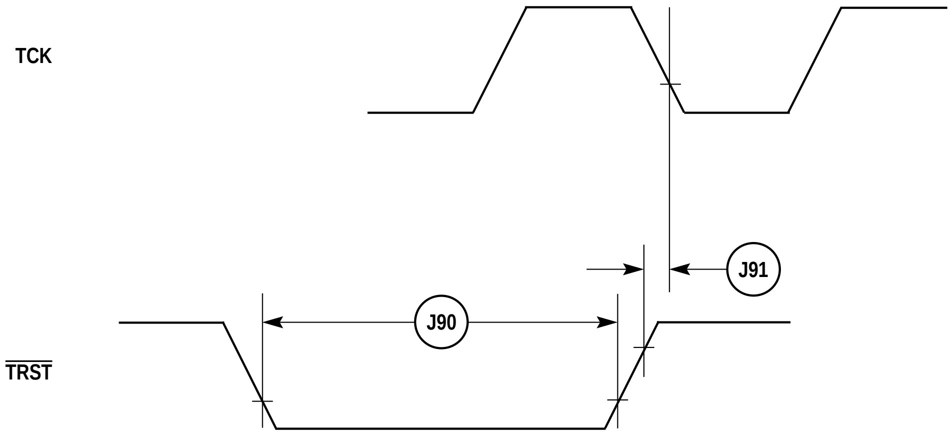


Figure 21-35. JTAG–$\overline{\text{TRST}}$ Timing Diagram

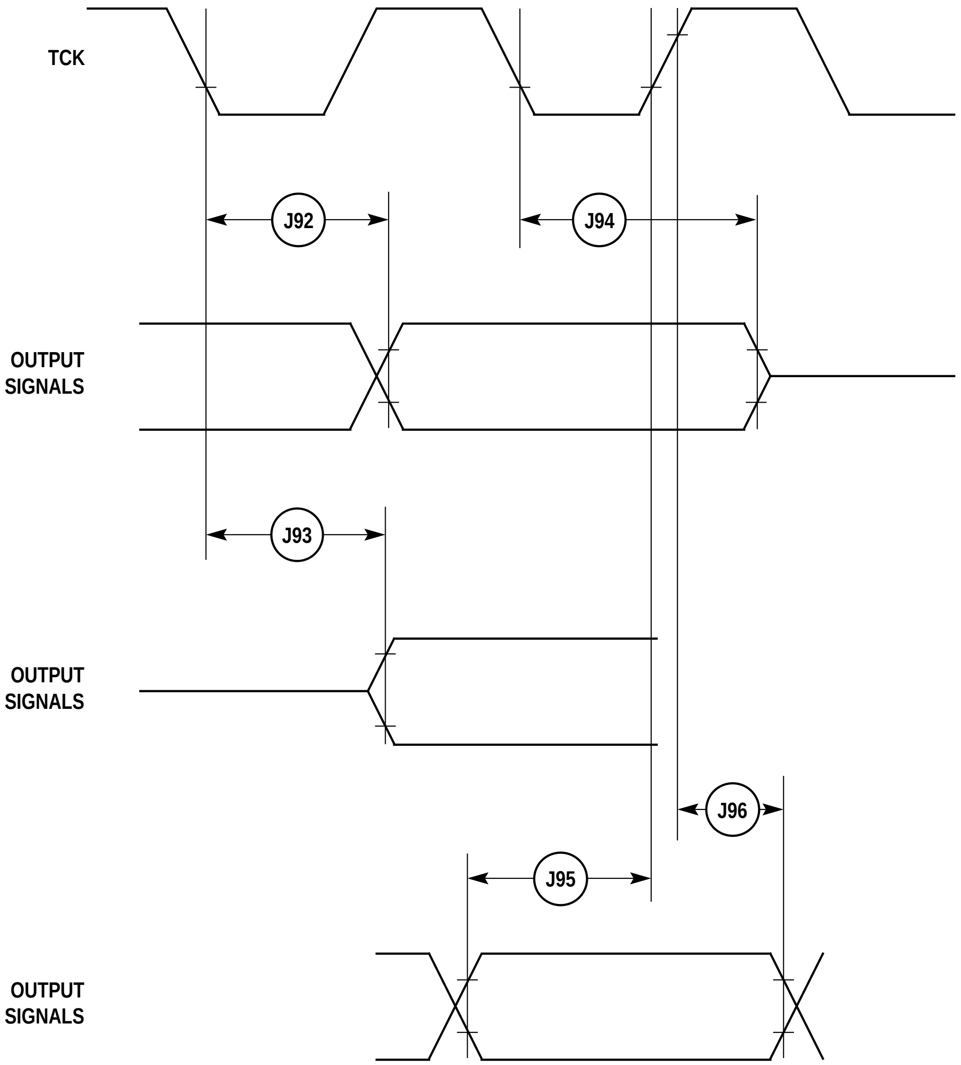


**Figure 21-36. Boundary Scan (JTAG) Timing Diagram**

# SECTION 22
# CPM ELECTRICAL CHARACTERISTICS

## 22.1 PIP/PIO AC ELECTRICAL SPECIFICATIONS

**Table 22-1. PIP/PIO Timing**

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 21 | Data-In Setup Time to STBI Low | | 0 | — | 0 | — | ns |
| 22 | Data-In Hold Time to STBI High | | 2.5 – t3 | — | 2.5 – t3 | — | clk |
| 23 | STBI Pulse Width | | 1.5 | — | 1.5 | — | clk |
| 24 | STBO Pulse Width | | 1 clk - 5ns | — | 1 clk - 5ns | — | ns |
| 25 | Data-Out Setup Time to STBO Low | | 2 | — | 2 | — | clk |
| 26 | Data-Out Hold Time from STBO High | | 5 | — | 5 | — | clk |
| 27 | STBI Low to STBO Low (Rx Interlock) | | — | 2 | — | 2 | clk |
| 28 | STBI Low to STBO High (Tx Interlock) | | 2 | — | 2 | — | clk |
| 29 | Data-In Setup Time to Clock Low | | 20 | — | 15 | — | ns |
| 30 | Data-In Hold Time from Clock Low | | 10 | — | 7.5 | — | ns |
| 31 | Clock High to Data-Out Valid (CPU Writes Data, Control, or Direction) | | — | 25 | — | 25 | ns |
| NOTE: t3 = Specification 23 | | | | | | | |

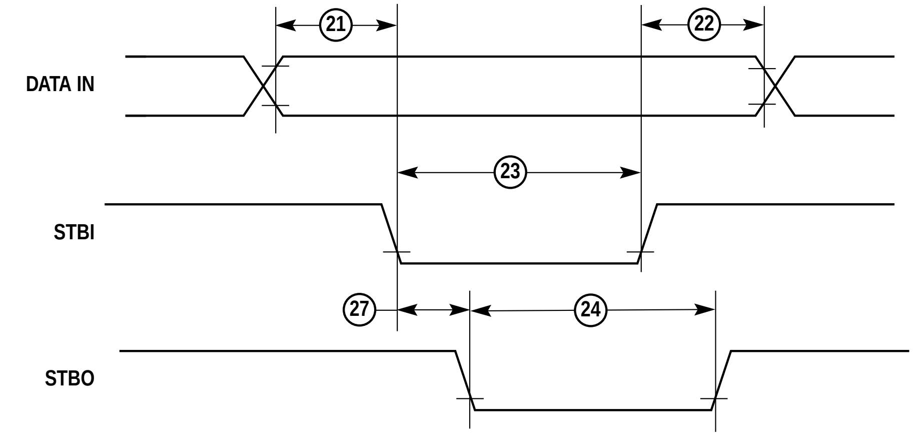


Figure 22-1. PIP RX (Interlock Mode) Timing Diagram

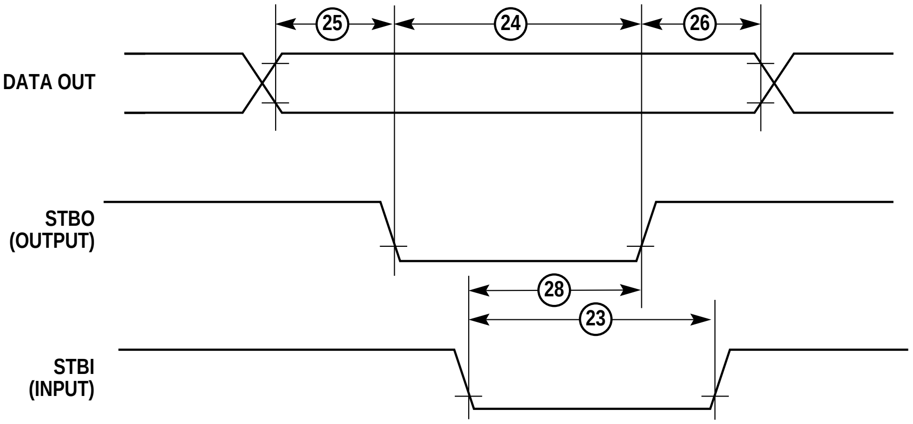


Figure 22-2. PIP TX (Interlock Mode) Timing Diagram

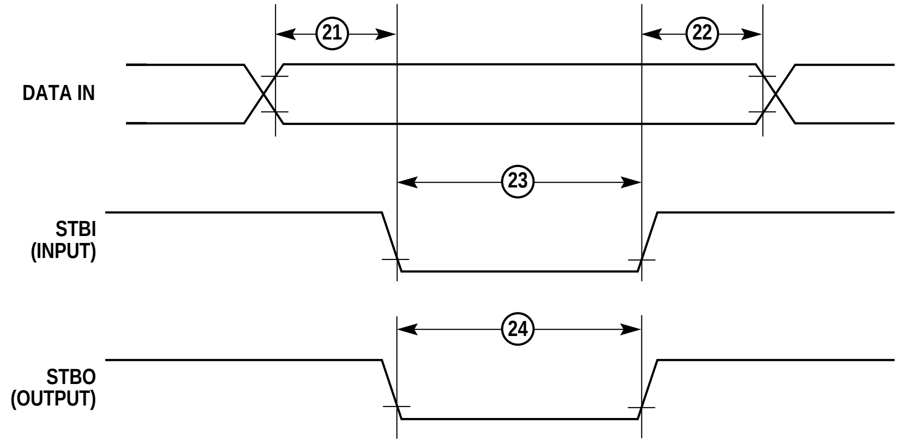


Figure 22-3. PIP RX (Pulse Mode) Timing Diagram

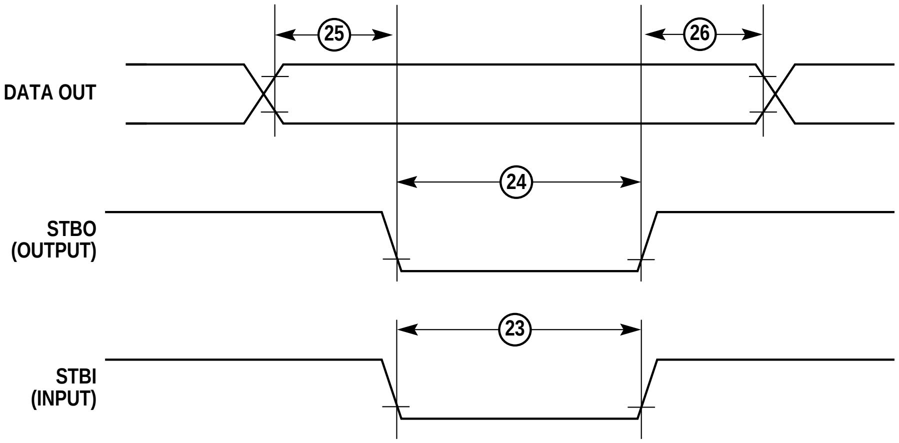


Figure 22-4. PIP TX (Pulse Mode) Timing Diagram

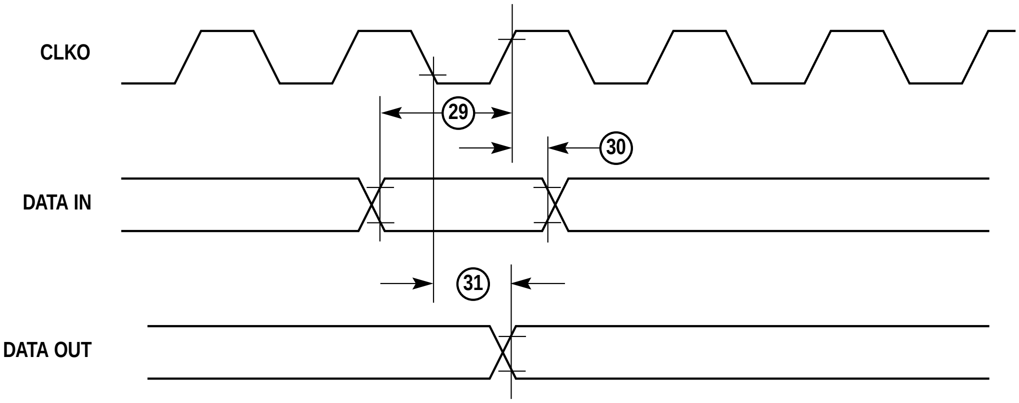


**Figure 22-5. Parallel I/O Data-In/Data-Out Timing Diagram**

## 22.2 IDMA CONTROLLER AC ELECTRICAL SPECIFICATIONS

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 40 | DREQ Setup Time to Clock High | | 12 | — | 7 | — | ns |
| 41 | DREQ Hold Time from Clock High | | 5 | — | 3 | — | ns |
| 42 | SDACK Assertion Delay from Clock High | | — | 20 | — | 12 | ns |
| 43 | SDACK Negation Delay from Clock Low | | — | 20 | — | 12 | ns |
| 44 | SDACK Negation Delay from $\overline{\text{TA}}$ Low | | — | 25 | — | 20 | ns |
| 45 | SDACK Negation Delay from Clock High | | — | 20 | — | 15 | ns |
| 46 | $\overline{\text{TA}}$ Assertion to Falling Edge of the Clock Setup Time * | | 12 | — | 7 | — | ns |

NOTE: * Applies to external TA.

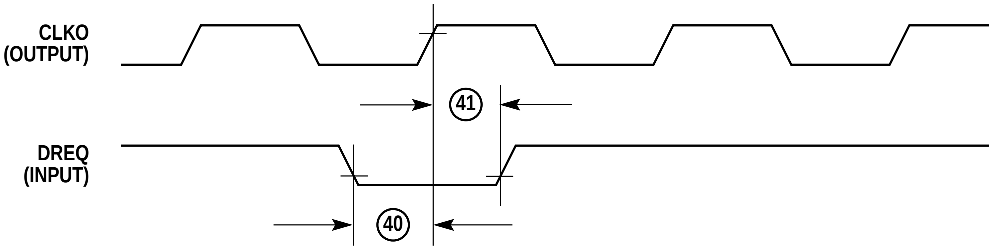


**Figure 22-6. IDMA External Requests Timing Diagram**

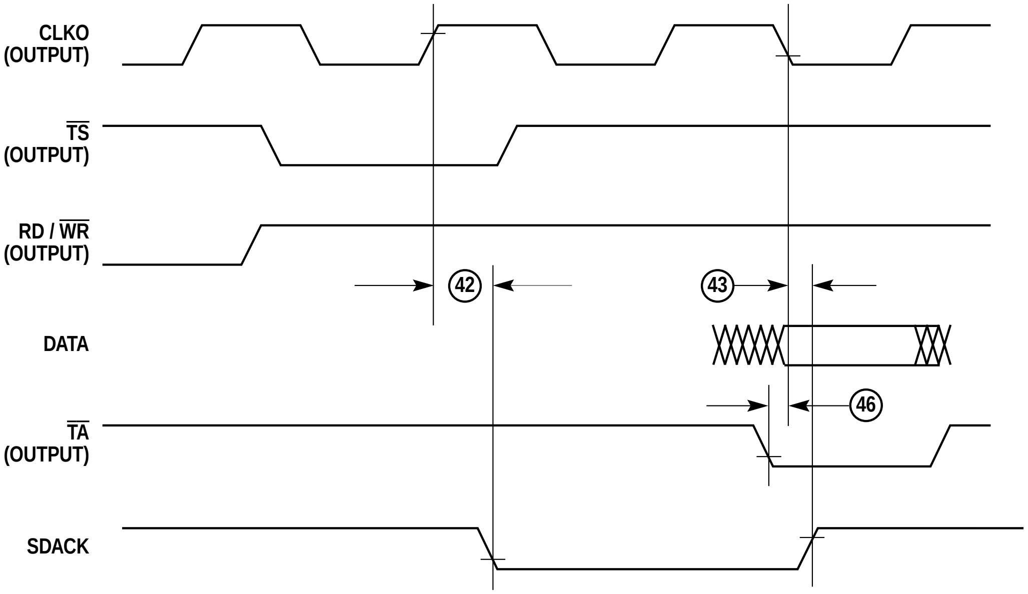


**Figure 22-7. SDACK Timing Diagram–Peripheral Write, TA Sampled Low at the Falling Edge of the Clock**

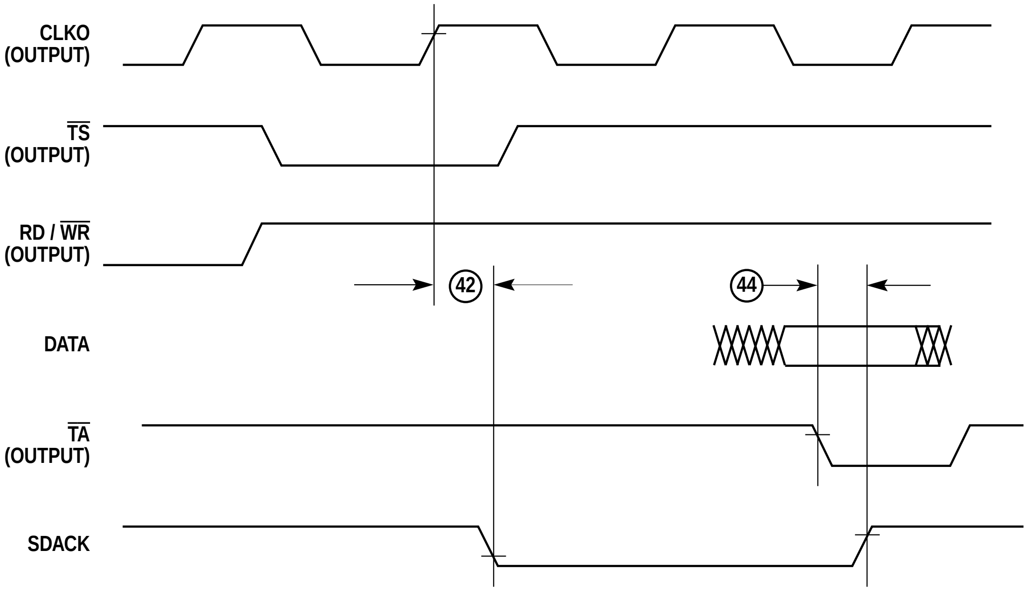


**Figure 22-8. SDACK Timing Diagram–Peripheral Write, TA Sampled High at the Falling Edge of the Clock**

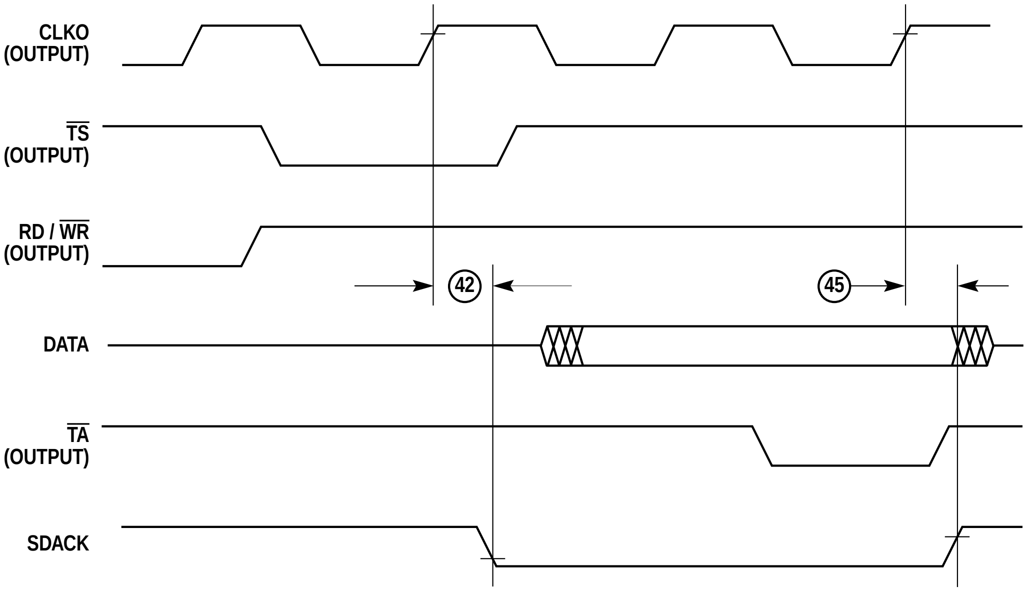


Figure 22-9. SDACK Timing Diagram–Peripheral Read

## 22.3 BAUD RATE GENERATOR AC ELECTRICAL SPECIFICATIONS

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 50 | BRGO Rise and Fall Time | | — | 10 | — | 10 | ns |
| 51 | BRGO Duty Cycle | | 40 | 60 | 40 | 60 | % |
| 52 | BRGO Cycle | | 40 | — | 40 | — | ns |

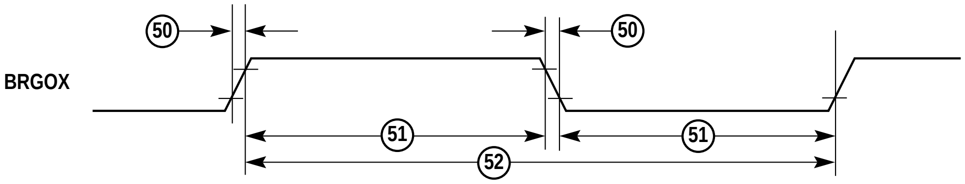


Figure 22-10. Baud Rate Generator Timing Diagram

## 22.4 TIMER AC ELECTRICAL SPECIFICATIONS

| NUM | CHARACTERISTIC | EXPRESSION | 25MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 61 | TIN/$\overline{\text{TGATE}}$ Rise and Fall Time | | 10 | — | 10 | — | ns |
| 62 | TIN/$\overline{\text{TGATE}}$ Low Time | | 1 | — | 1 | — | clk |
| 63 | TIN/$\overline{\text{TGATE}}$ High Time | | 2 | — | 2 | — | clk |
| 64 | TIN/$\overline{\text{TGATE}}$ Cycle Time | | 3 | — | 3 | — | clk |
| 65 | CLKO High to $\overline{\text{TOUT}}$ Valid | | 3 | 25 | 3 | 25 | ns |

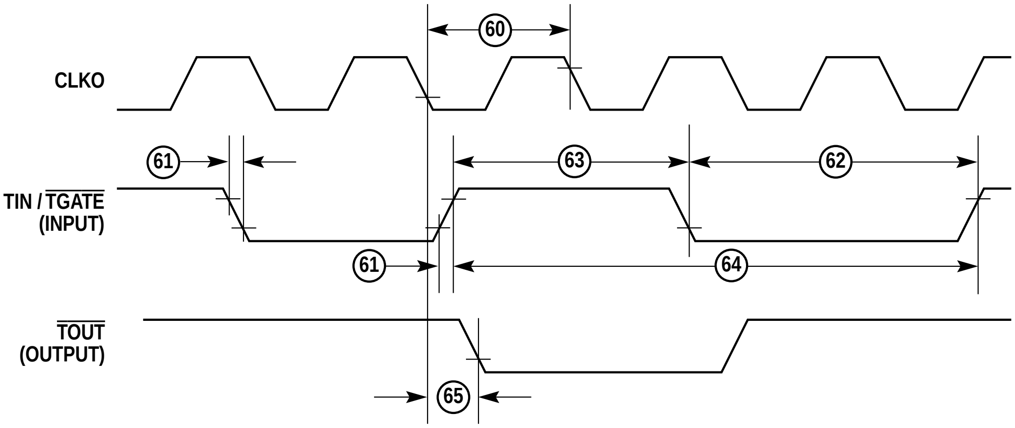


Figure 22-11. CPM General-Purpose Timers Timing Diagram

## 22.5 SERIAL INTERFACE AC ELECTRICAL SPECIFICATIONS

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 70 | L1RCLK, L1TCLK Frequency (DSC=0)[1 3] | | — | 10 | — | 10 | MHz |
| 71 | L1RCLK, L1TCLK Width Low (DSC=0)[3] | | P+10 | — | P+10 | — | ns |
| 71A | L1RCLK, L1TCLK Width High (DSC=0)[2] | | P+10 | — | P+10 | — | ns |
| 72 | L1TXD, L1ST(1–4), L1RQ, L1CLKO Rise/Fall Time | | — | 15 | — | 15 | ns |
| 73 | L1RSYNC, L1TSYNC Valid to L1CLK Edge (SYNC Setup Time) | | 20 | — | 20 | — | ns |
| 74 | L1CLK Edge to L1RSYNC, L1TSYNC Invalid (SYNC Hold Time) | | 35 | — | 35 | — | ns |
| 75 | L1RSYNC, L1TSYNC Rise/Fall Time | | — | 15 | — | 15 | ns |
| 76 | L1RXD Valid to L1CLK Edge (L1RXD Setup Time) | | 42 | — | 42 | — | ns |
| 77 | L1CLK Edge to L1RXD Invalid (L1RXD Hold Time) | | 35 | — | 35 | — | ns |
| 78 | L1CLK Edge to L1ST(1–4) Valid | | 10 | 45 | 10 | 45 | ns |
| 78A | L1SYNC Valid to L1ST(1–4) Valid [4] | | 10 | 45 | 10 | 45 | ns |
| 79 | L1CLK Edge to L1ST(1–4) Invalid | | 10 | 45 | 10 | 45 | ns |
| 80 | L1CLK Edge to L1TXD Valid | | 10 | 65 | 10 | 65 | ns |
| 80A | L1TSYNC Valid to L1TXD Valid [4] | | 10 | 65 | 10 | 65 | ns |
| 81 | L1CLK Edge to L1TXD High Impedance | | 0 | 42 | 0 | 42 | ns |
| 82 | L1RCLK, L1TCLK Frequency (DSC=1) | | — | 12.5 | — | 16 | MHz |
| 83 | L1RCLK, L1TCLK Width Low (DSC=1) | | P+10 | — | P+10 | — | ns |
| 83A | L1RCLK, L1TCLK Width High (DSC=1)[2] | | P+10 | — | P+10 | — | ns |
| 84 | L1CLK Edge to L1CLKO Valid (DSC=1) | | — | 30 | — | 30 | ns |
| 85 | L1RQ Valid Before Falling Edge of L1TSYNC [4] | | 1 | — | 1 | — | L1TCLK |
| 86 | L1GR Setup Time (See Note 3) | | 42 | — | 42 | — | ns |

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 87 | L1GR Hold Time | | 42 | — | 42 | — | ns |
| 88 | L1CLK Edge to L1SYNC Valid<br>(FSD = 00, CNT = 0000, BYT = 0, DSC=0) | | — | 0 | — | 0 | ns |

NOTES:
1. The ratio SyncCLK/L1RCLK must be greater than 2.5/1.
2. Where P=1/CLKO1. Thus for a 25 MHz CLKO1 rate, P= 40 ns.
3. These specs are valid for IDL mode only.
4. The strobes and Txd on the first bit of the frame becomes valid after L1CLK edge or L1SYNC, whichever is later.

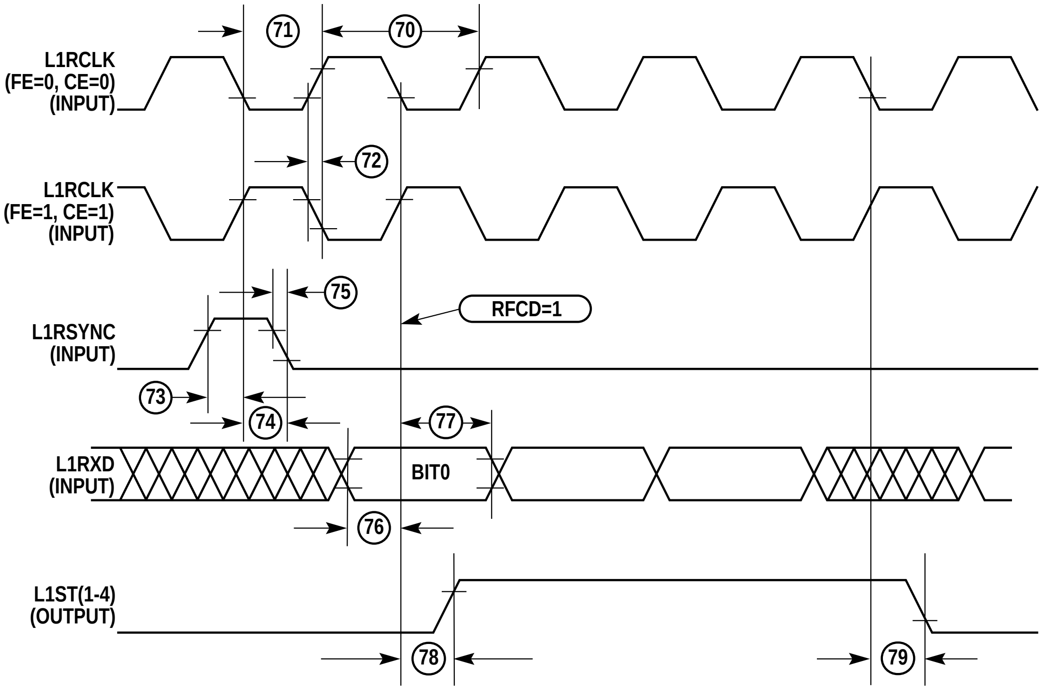


**Figure 22-12. SI Receive Timing Diagram With Normal Clocking (DSC =0)**

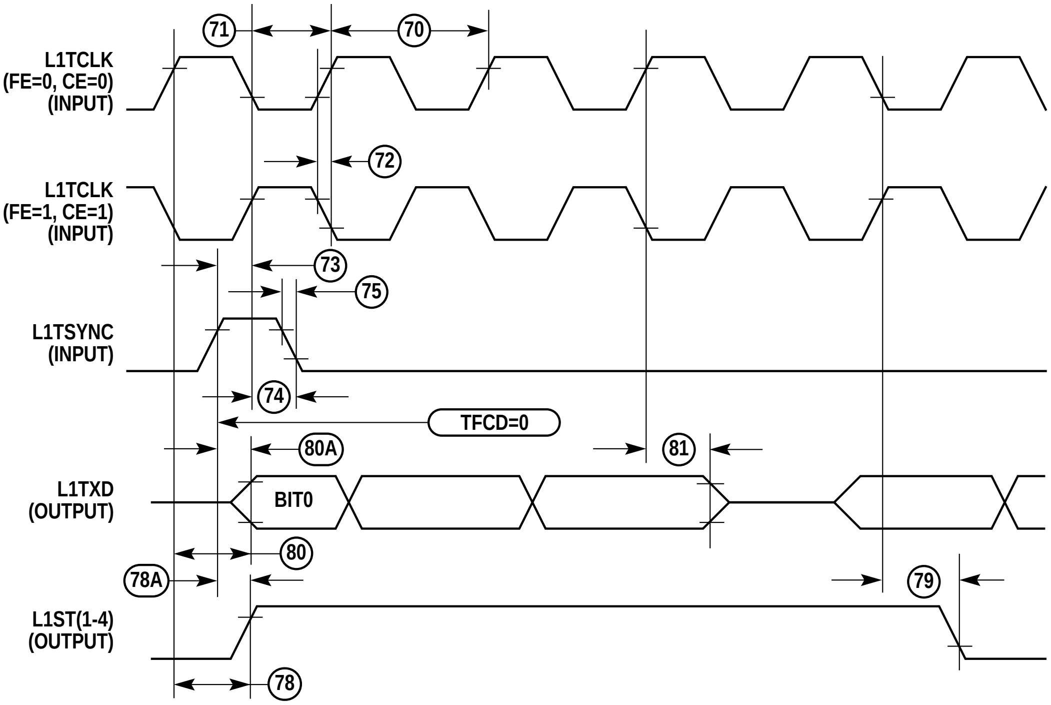


Figure 22-13. SI Transmit Timing Diagram

## 22.6 SCC IN NMSI MODE–EXTERNAL CLOCK ELECTRICAL SPECIFICATIONS

The electrical specifications in this document are preliminary.

**Table 22-2. NMSI External Clock Timing**

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 100 | RCLK1 and TCLK1 Width High[1] | | CLKO | — | CLKO<br>25 | — | ns |
| 101 | RCLK1 and TCLK1 Width Low | | CLKO<br>+5ns | — | CLKO<br>+5ns<br>30 | — | ns |
| 102 | RCLK1 and TCLK1 Rise/Fall Time | | — | 15 | — | 15 | ns |
| 103 | TXD1 Active Delay<br>(From TCLK1 Falling Edge) | | 0 | 50 | 0 | 50 | ns |
| 104 | $\overline{\text{RTS1}}$ Active/Inactive Delay<br>(From TCLK1 Falling Edge) | | 0 | 50 | 0 | 50 | ns |
| 105 | $\overline{\text{CTS1}}$ Setup Time to TCLK1 Rising Edge | | 5 | — | 5 | — | ns |
| 106 | RXD1 Setup Time to RCLK1 Rising Edge | | 5 | — | 5 | — | ns |
| 107 | RXD1 Hold Time from RCLK1 Rising Edge [2] | | 5 | — | 5 | — | ns |
| 108 | CD1 Setup Time to RCLK1 Rising Edge | | 5 | — | 5 | — | ns |

NOTES:
1. The ratio SyncCLK/RCLK1 and SyncCLK/TCLK1 must be greater or equal to 2.25/1.
2. Also applies to CD and $\overline{\text{CTS}}$ hold time when they are used as an external sync signals.

## 22.7 SCC IN NMSI MODE–INTERNAL CLOCK ELECTRICAL SPECIFICATIONS

The electrical specifications in this document are preliminary.

**Table 22-3. NMSI Internal Clock Timing**

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 100 | RCLK1 and TCLK1 Frequency [1] | | 0 | 8.3 | 0 | 13 | MHz |
| 102 | RCLK1 and TCLK1 Rise/Fall Time | | — | — | — | — | ns |
| 103 | TXD1 Active Delay<br>(From TCLK1 Falling Edge) | | 0 | 30 | 0 | 30 | ns |
| 104 | $\overline{\text{RTS1}}$ Active/Inactive Delay<br>(From TCLK1 Falling Edge) | | 0 | 30 | 0 | 30 | ns |
| 105 | $\overline{\text{CTS1}}$ Setup Time to TCLK1 Rising Edge | | 40 | — | 40 | — | ns |
| 106 | RXD1 Setup Time to RCLK1 Rising Edge | | 40 | — | 40 | — | ns |
| 107 | RXD1 Hold Time from RCLK1 Rising Edge [2] | | 0 | — | 0 | — | ns |
| 108 | CD1 Setup Time to RCLK1 Rising Edge | | 40 | — | 40 | — | ns |

NOTES: 1. The ratio SyncCLK/RCLK1 and SyncCLK/TCLK1 must be greater or equal to 3/1.

2. Also applies to CD and $\overline{\text{CTS}}$ hold time when they are used as an external sync signals.

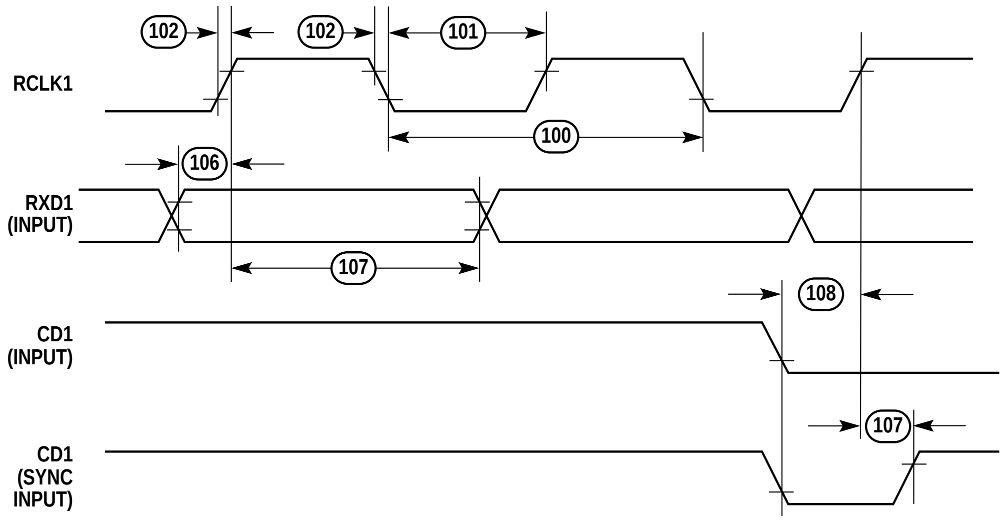


**Figure 22-14. SCC NMSI Receive Timing Diagram**

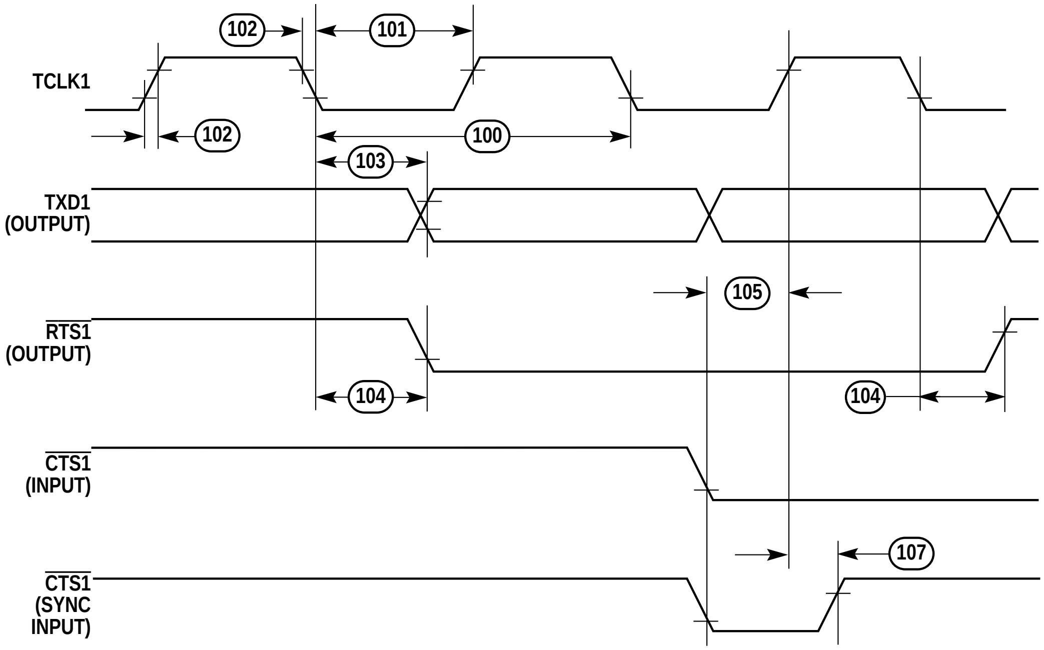


**Figure 22-15. SCC NMSI Transmit Timing Diagram**

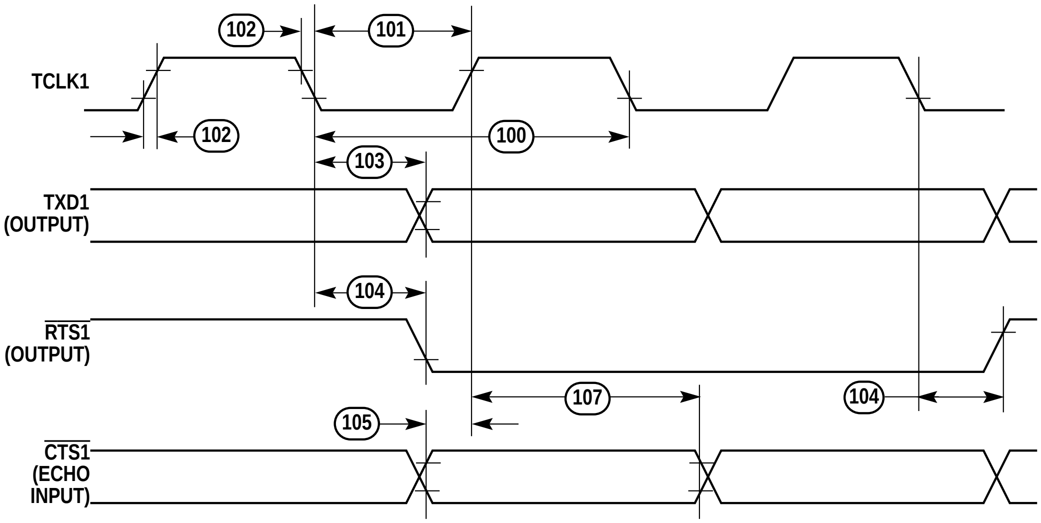


**Figure 22-16. HDLC Bus Timing Diagram**

## 22.8 ETHERNET ELECTRICAL SPECIFICATIONS

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 120 | CLSN Width High | | 40 | — | 40 | — | ns |
| 121 | RCLK1 Rise/Fall Time | | — | 15 | — | 15 | ns |
| 122 | RCLK1 Width Low | | 40 | — | 40 | — | ns |
| 123 | RCLK1 Clock Period [1] | | 80 | 120 | 80 | 120 | ns |
| 124 | RXD1 Setup Time | | 20 | — | 20 | — | ns |
| 125 | RXD1 Hold Time | | 5 | — | 5 | — | ns |
| 126 | RENA Active Delay (From RCLK1 Rising Edge of the Last Data Bit) | | 10 | — | 10 | — | ns |
| 127 | RENA Width Low | | 100 | — | 100 | — | ns |
| 128 | TCLK1 Rise/Fall Time | | — | 15 | — | 15 | ns |
| 129 | TCLK1 Width Low | | 40 | — | 40 | — | ns |
| 130 | TCLK1 Clock Period [1] | | 99 | 101 | 99 | 101 | ns |
| 131 | TXD1 Active Delay (From TCLK1 Rising Edge) | | 10 | 50 | 10 | 50 | ns |
| 132 | TXD1 Inactive Delay (From TCLK1 Rising Edge) | | 10 | 50 | 10 | 50 | ns |
| 133 | TENA Active Delay (From TCLK1 Rising Edge) | | 10 | 50 | 10 | 50 | ns |
| 134 | TENA Inactive Delay (From TCLK1 Rising Edge) | | 10 | 50 | 10 | 50 | ns |
| 135 | $\overline{\text{RSTRT}}$ Active Delay (From TCLK1 Falling Edge) | | 10 | 50 | 10 | 50 | ns |
| 136 | $\overline{\text{RSTRT}}$ Inactive Delay (From TCLK1 Falling Edge) | | 10 | 50 | 10 | 50 | ns |
| 137 | REJECT Width Low | | 1 | — | 1 | — | CLK |
| 138 | CLKO1 Low to SDACK Asserted [2] | | — | 20 | — | 20 | ns |
| 139 | CLKO1 Low to SDACK Negated [2] | | — | 20 | — | 20 | ns |

NOTES:
1. The ratio SyncCLK/RCLK1 and SyncCLK/TCLK1 must be greater or equal to 2/1.
2. SDACK is asserted whenever the SDMA writes the incoming frame DA into memory.

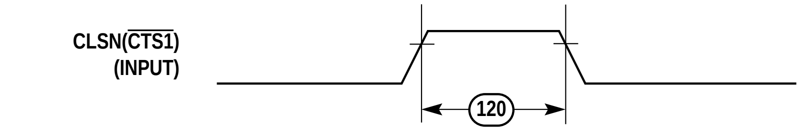


Figure 22-17. Ethernet Collision Timing Diagram

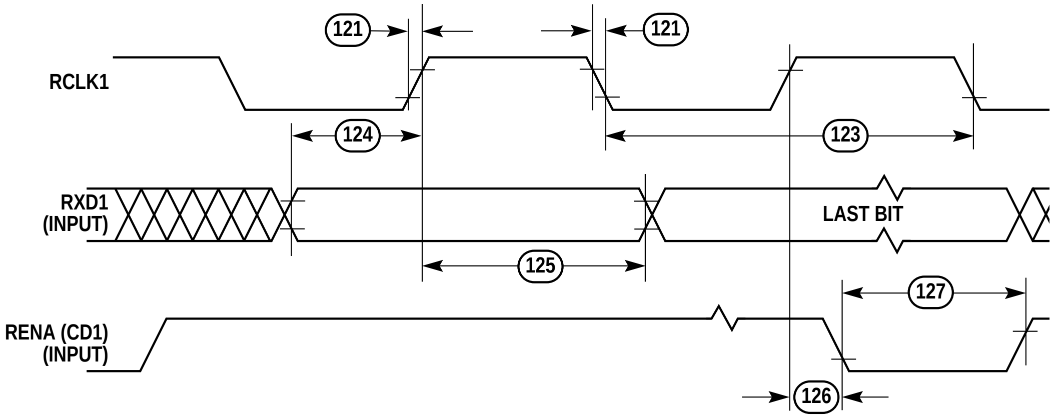
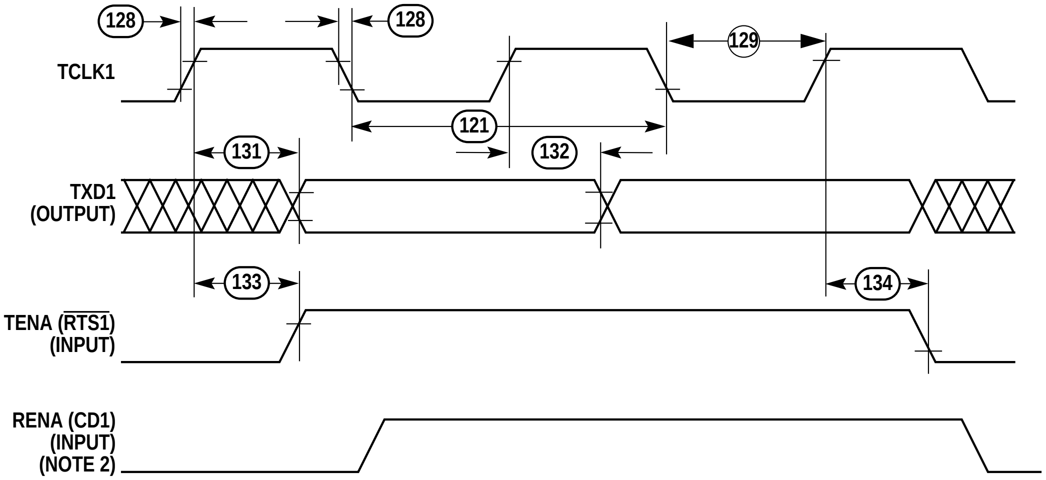


Figure 22-18. Ethernet Receive Timing Diagram



NOTES: 1. TRANSMIT CLOCK INVERT (TCI) BIT IN GSMR IS SET.
2. IF RENA IS DEASSERTED BEFORE TENA, OR RENA IS NOT ASSERTED AT ALL DURING TRANSMIT, THEN THE CSL BIT IS SET IN THE BUFFER DESCRIPTOR AT THE END OF THE FRAME TRANSMISSION.

Figure 22-19. Ethernet Transmit Timing Diagram

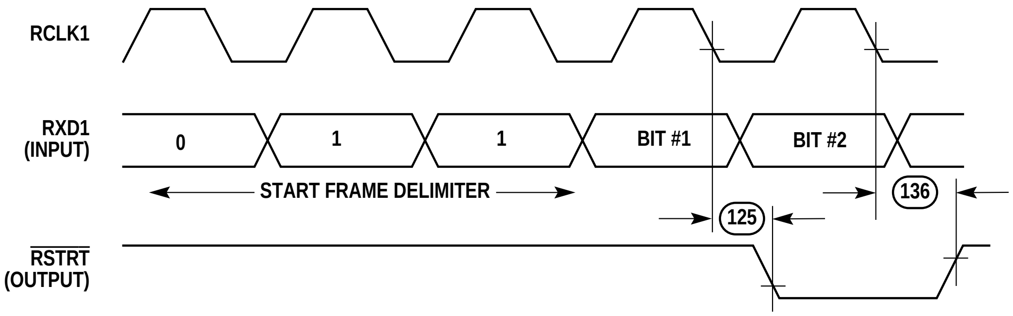


Figure 22-20. CAM Interface Receive Start Timing Diagram

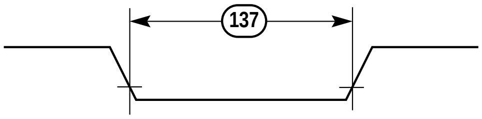
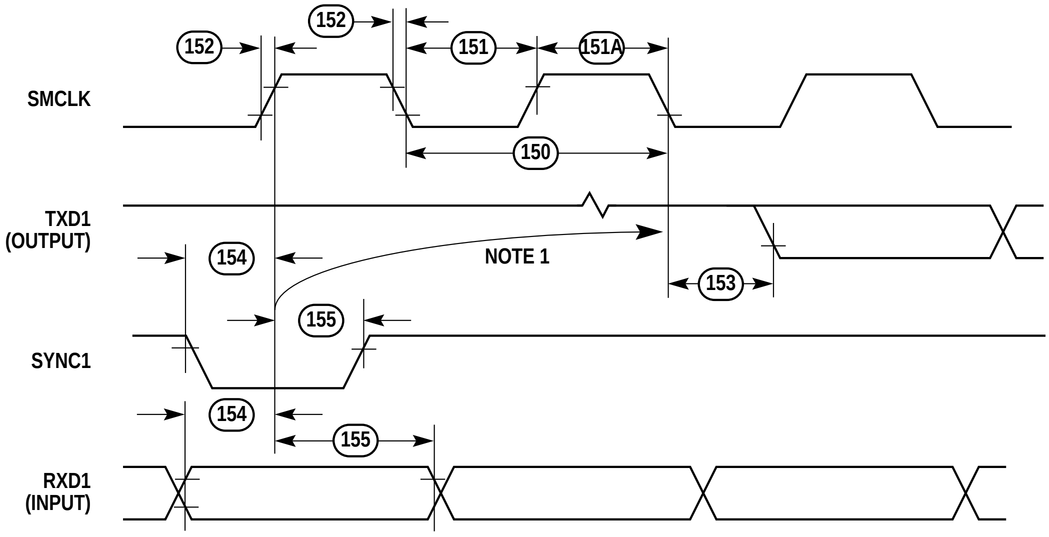


Figure 22-21. CAM Interface Reject Timing Diagram

## 22.9 I$^2$C AC ELECTRICAL SPECIFICATIONS–SCL < 100 KHz

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 150 | CLK1 Clock Period * | | 100 | — | 100 | — | ns |
| 151 | CLK1 Width Low | | 50 | — | 50 | — | ns |
| 151A | CLK1 Width High | | 50 | — | 50 | — | ns |
| 152 | CLK1 Rise/Fall Time | | — | 15 | — | 15 | ns |
| 153 | TXD1 Active Delay (From CLK1 Falling Edge) | | 10 | 50 | 10 | 50 | ns |
| 154 | RXD1/SYNC1 Setup Time | | 20 | — | 20 | — | ns |
| 155 | RXD1/SYNC1 Hold Time | | 5 | — | 5 | — | ns |

NOTE: * The ratio SyncCLK/SMCLK must be greater or equal to 2/1.



NOTE: 1. THIS DELAY IS EQUAL TO AN INTEGER NUMBER OF “CHARACTER LENGTH” CLOCKS

**Figure 22-22. SMC Transparent Timing Diagram**

## 22.10 SPI MASTER AC ELECTRICAL SPECIFICATIONS

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 160 | Master Cycle Time | | 4 | 1024 | 4 | 1024 | tcyc |
| 161 | Master Clock (SCK) High or Low Time | | 2 | 512 | 2 | 512 | tcyc |
| 162 | Master Data Setup Time (Inputs) | | 50 | — | 50 | — | ns |
| 163 | Master Data Hold Time (Inputs) | | 0 | — | 0 | — | ns |
| 164 | Master Data Valid (After SCK Edge) | | — | 20 | — | 20 | ns |
| 165 | Master Data Hold Time (Outputs) | | 0 | — | 0 | — | ns |
| 166 | Rise Time Output | | — | 15 | — | 15 | ns |
| 167 | Fall Time Output | | — | 15 | — | 15 | ns |

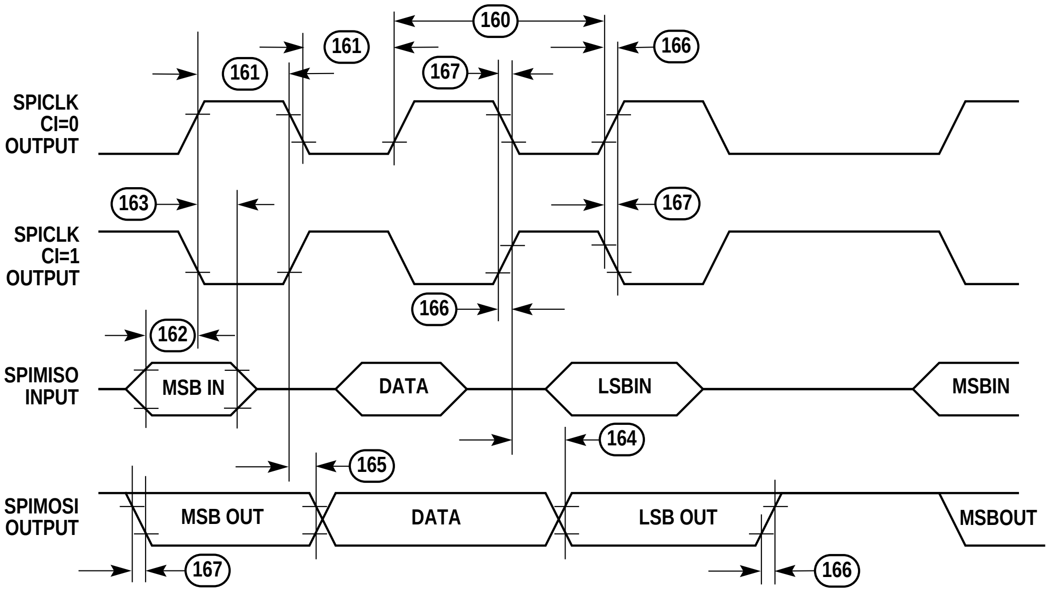


Figure 22-23. SPI Master (CP=0) Timing Diagram

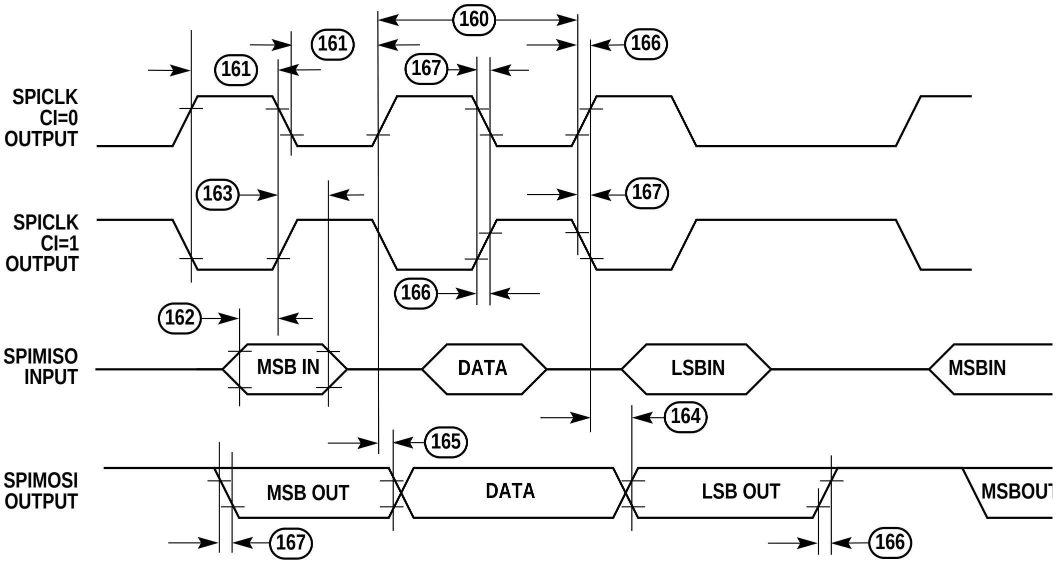


**Figure 22-24. SPI Master (CP=1) Timing Diagram**

## 22.11 SPI SLAVE AC ELECTRICAL SPECIFICATIONS

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 170 | Slave Cycle Time | | 2 | — | 2 | — | tcyc |
| 171 | Slave Enable Lead Time | | 15 | — | 15 | — | ns |
| 172 | Slave Enable Lag Time | | 15 | — | 15 | — | ns |
| 173 | Slave Clock (SPICLK) High or Low Time | | 1 | — | 1 | — | tcyc |
| 174 | Slave Sequential Transfer Delay (Does Not Require Deselect) | | 1 | — | 1 | — | tcyc |
| 175 | Slave Data Setup Time (Inputs) | | 20 | — | 20 | — | ns |
| 176 | Slave Data Hold Time (Inputs) | | 20 | — | 20 | — | ns |
| 177 | Slave Access Time | | — | 50 | — | 50 | ns |

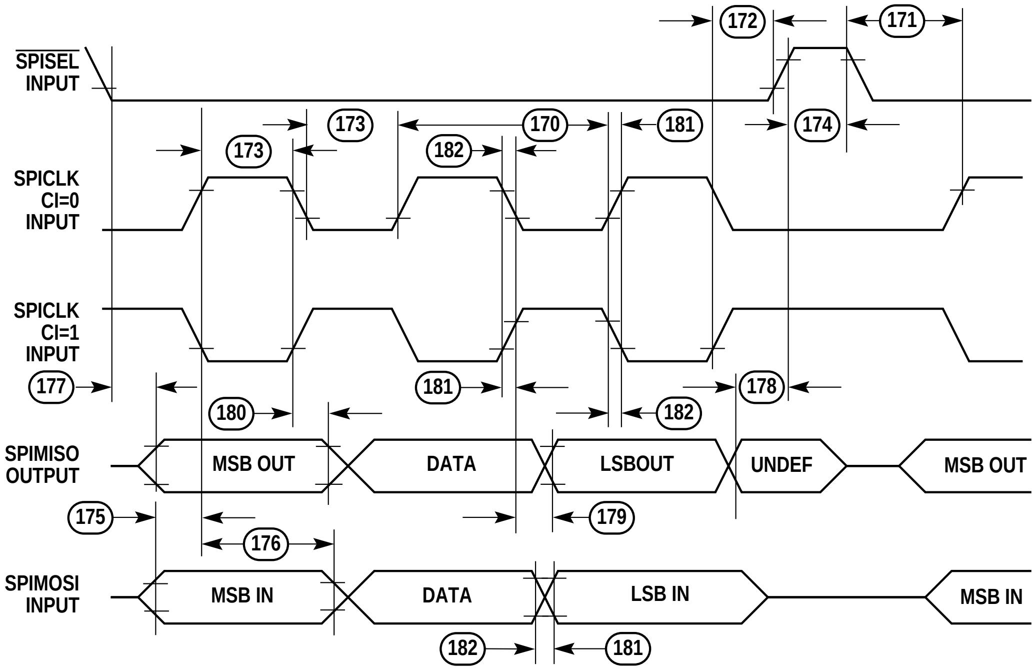


**Figure 22-25. SPI Slave (CP=0) Timing Diagram**

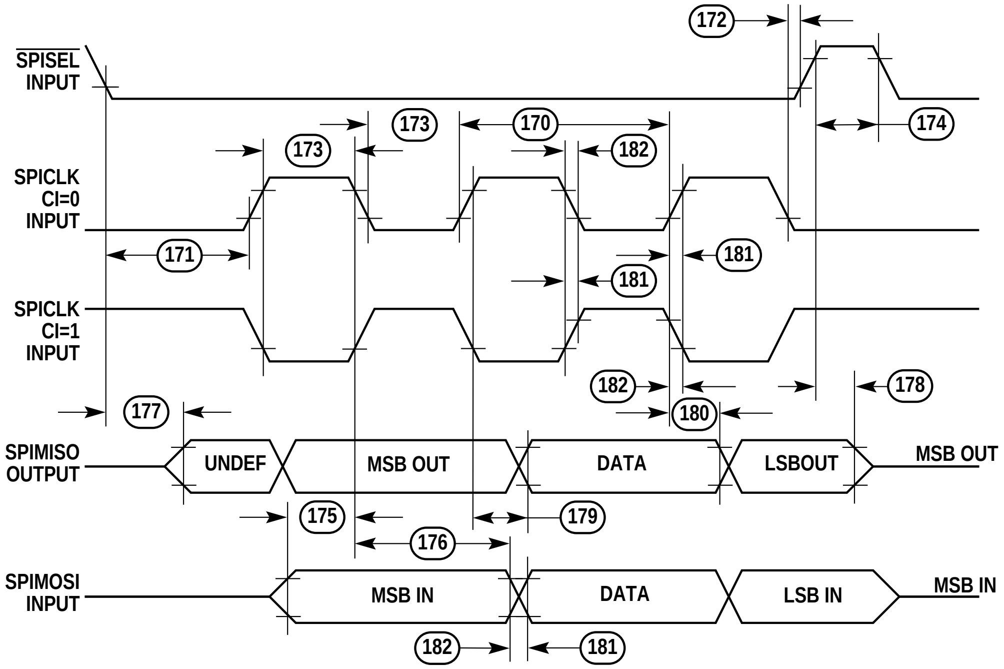


**Figure 22-26. SPI Slave (CP=1) Timing Diagram**

## 22.12 I$^2$C AC ELECTRICAL SPECIFICATIONS–SCL < 100 KHz

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 200 | SCL Clock Frequency (SLAVE) | | 0 | 100 | 0 | 100 | KHz |
| 200 | SCL Clock Frequency (MASTER) * | | 1.5 | 100 | 1.5 | 100 | KHz |
| 202 | Bus Free Time Between Transmissions | | 4.7 | — | 4.7 | — | µs |
| 203 | LOW Period of SCL | | 4.7 | — | 4.7 | — | µs |
| 204 | HIGH Period of SCL | | 4.0 | — | 4.0 | — | µs |
| 205 | START Condition Setup Time | | 4.7 | — | 4.7 | — | µs |
| 206 | START Condition Hold Time | | 4.0 | — | 4.0 | — | µs |
| 207 | DATA Hold Time | | 0 | — | 0 | — | µs |
| 208 | DATA Setup Time | | 250 | — | 250 | — | ns |
| 209 | SDL/SCL Rise Time | | — | 1 | — | 1 | µs |
| 210 | SDL/SCL Fall Time | | — | 300 | — | 300 | ns |
| 211 | STOP Condition Setup Time | | 4.7 | — | 4.7 | — | µs |

NOTE: * SCL frequency is given by SCL = BRGCLK_frequency / ((BRG register + 3) * pre_scaler * 2). The ratio SyncClk/(BRGCLK/pre_scaler) must be greater or equal to 4/1.

## 22.13 I²C AC ELECTRICAL SPECIFICATIONS–SCL > 100 KHz

| NUM | CHARACTERISTIC | EXPRESSION | MIN | MAX | UNIT |
|---|---|---|---|---|---|
| 200 | SCL Clock Frequency (SLAVE) | fSCL | 0 | BRGCLK/48 | Hz |
| 200 | SCL Clock Frequency (MASTER) * | fSCL | BRGCLK/16512 | BRGCLK/48 | Hz |
| 202 | Bus Free Time Between Transmissions | | 1/(2.2 * fSCL) | — | s |
| 203 | LOW Period of SCL | | 1/(2.2 * fSCL) | — | s |
| 204 | HIGH Period of SCL | | 1/(2.2 * fSCL) | — | s |
| 205 | START Condition Setup Time | | 1/(2.2 * fSCL) | — | s |
| 206 | START Condition Hold Time | | 1/(2.2 * fSCL) | — | s |
| 207 | DATA Hold Time | | 0 | — | s |
| 208 | DATA Setup Time | | 1/(40 * fSCL) | — | s |
| 209 | SDL/SCL Rise Time | | — | 1/(10 * fSCL) | s |
| 210 | SDL/SCL Fall Time | | — | 1/(33 * fSCL) | s |
| 211 | STOP Condition Setup Time | | 1/(2.2 * fSCL) | — | s |

NOTE: * SCL frequency is given by SCL = BrgClk_frequency / ((BRG register + 3) * pre_scaler * 2). The ratio SyncClk/(Brg_Clk/pre_scaler) must be greater or equal to 4/1.

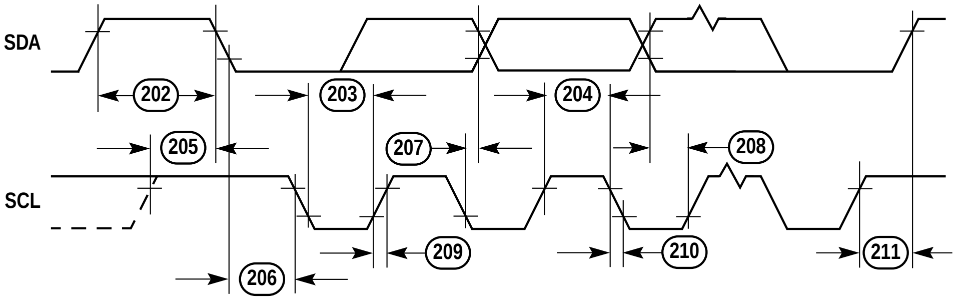


Figure 22-27. I²C Bus Timing Diagram

## 22.14 LCD CONTROLLER AC ELECTRICAL SPECIFICATIONS

| NUM | CHARACTERISTIC | EXPRESSION | 25 MHZ | | 50 MHZ | | UNIT |
|---|---|---|---|---|---|---|---|
| | | | MIN | MAX | MIN | MAX | |
| 220 | Shift Clock Cycle Time | | 40 | — | 40 | — | ns |
| 221 | Shift Clock High Time | | 20 | — | 20 | — | ns |
| 223 | Clock/Hsync/Vsync/$\overline{\text{OE}}$ Rise/Fall Time | | — | 10 | — | 10 | ns |
| 224 | Data Valid Delay from Shift Clock High | | — | 15 | — | 15 | ns |
| 225 | Vsync to Hsync Setup Time [1] | | 5 | — | 5 | — | T |
| 226 | Vsync Hold Time | | 1 | — | 1 | — | T |
| 227 | Hsync Pulse Width | | 4 | — | 4 | — | T |
| 228 | Time from Clock Falling Edge to Hsync Rising Edge | | 4.5 | — | 4.5 | — | T |
| 229 | Time from Hsync Falling Edge to Clock Rising Edge [2] | | 4 | — | 4 | — | T |
| 230 | AC Active Delay | | — | 25 | — | 25 | ns |
| 231 | Vsync Pulse Width (TFT) | | 1 | 16 | 1 | 16 | Line |
| 232 | Hsync to $\overline{\text{OE}}$ Delay [3] | | 4 | — | 4 | — | T |
| 233 | $\overline{\text{OE}}$ to Hsync Delay | | 4 | — | 4 | — | T |
| 234 | Vsync to $\overline{\text{OE}}$ Delay (TFT) | | 0 | 1,023 | 0 | 1,023 | T |
| 235 | Vsync/Hsync/$\overline{\text{OE}}$ Active Delay (TFT) | | — | 15 | — | 15 | ns |
| 236 | Wait between Frames [4] | | WBF | — | WBF | — | Line |

NOTES:
1. T= shift clock cycle.
2. This number is given for wbl (wait between lines) ≤ 2. For wbl=n {n>2} the timing will be (n+2)T.
3. This number is given for wbl (wait between lines) ≤ 2. For wbl=n {n>2} the timing will be (n+2)T.
4. Wait between frames (WBF) is programmable parameter.

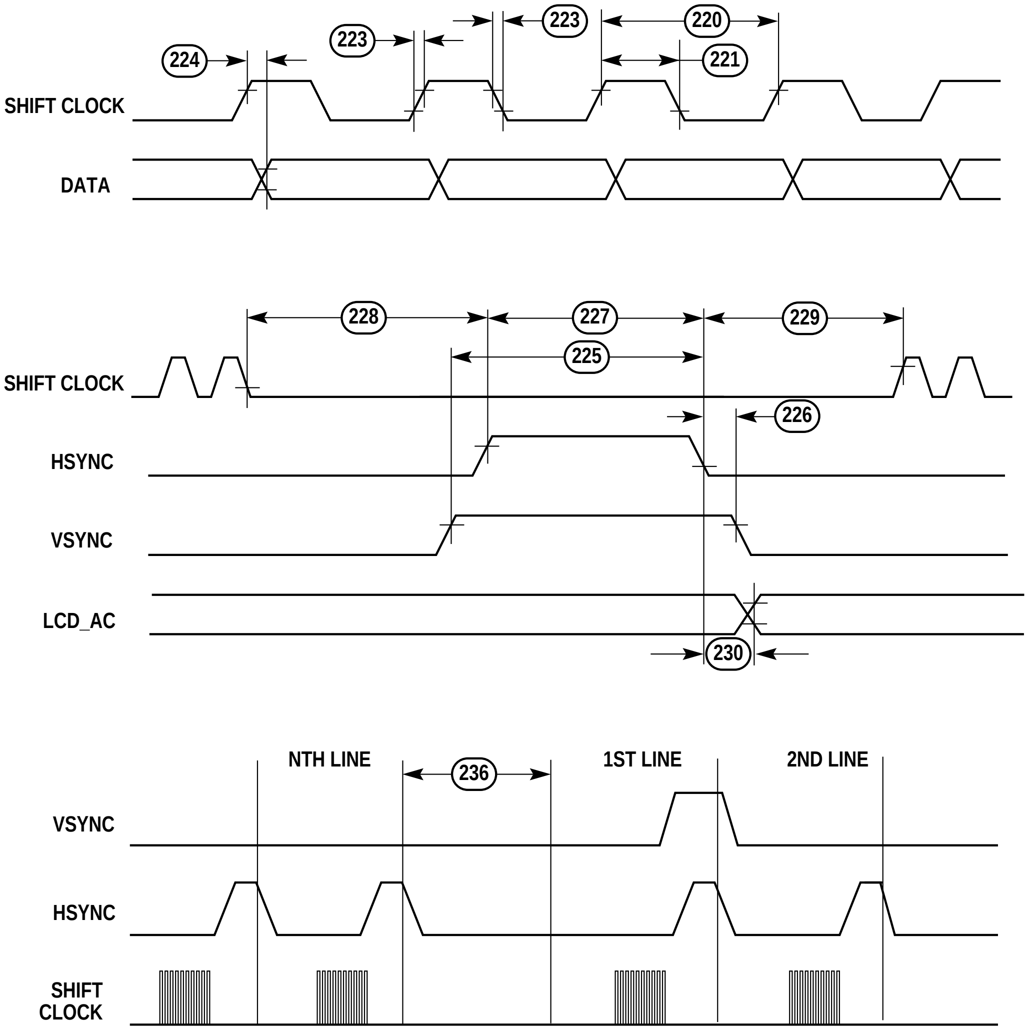


**Figure 22-28. Passive Panel Timing Diagram**

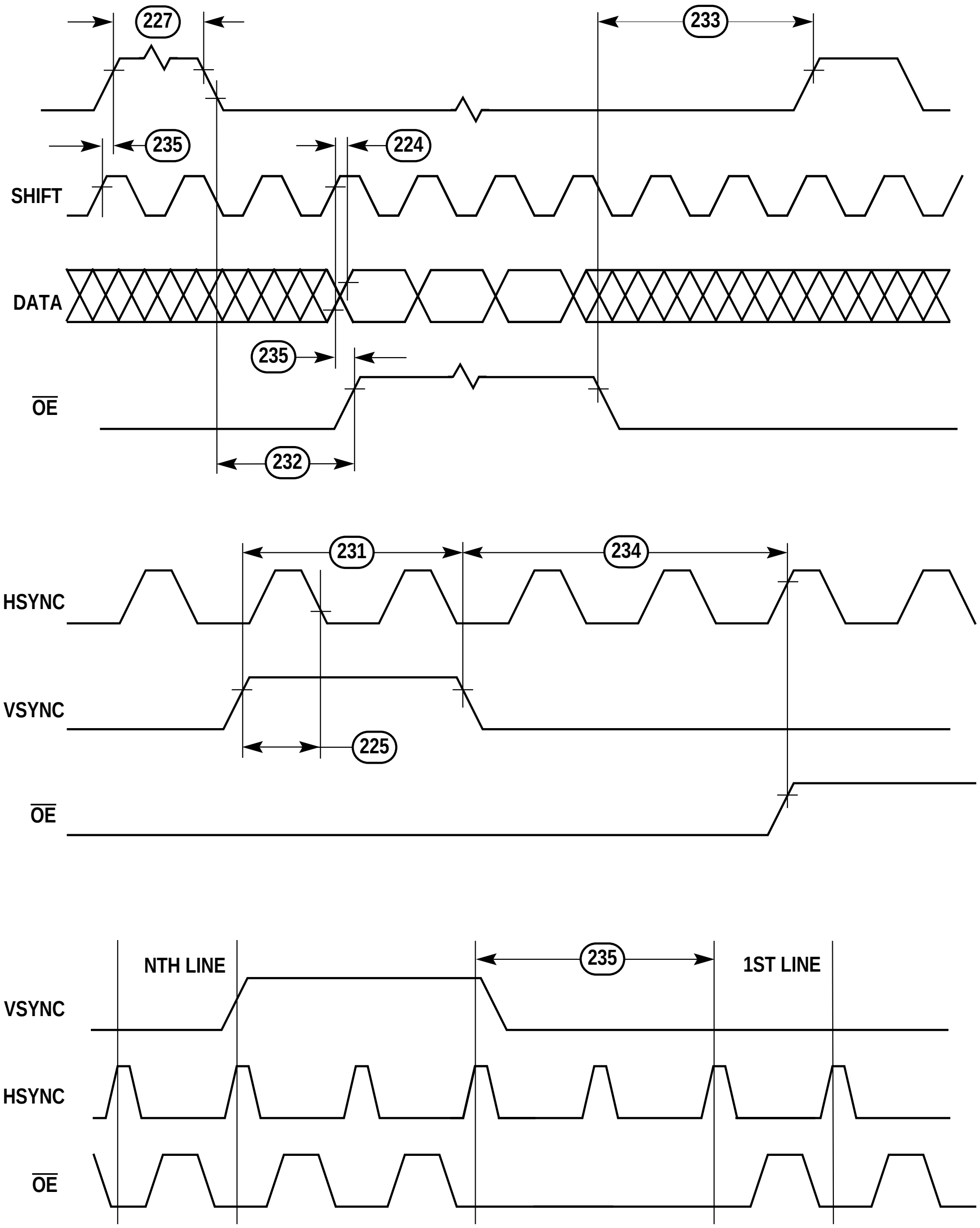


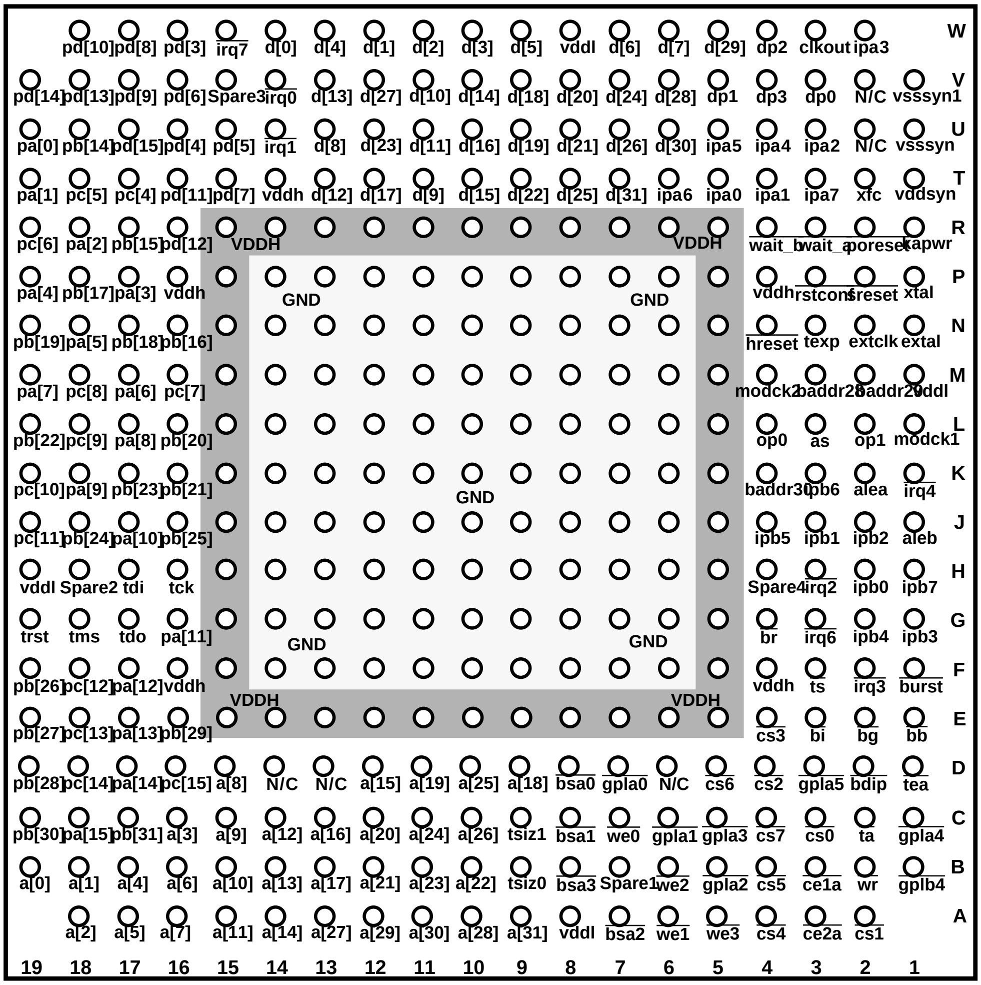
**Figure 22-29. TFT Panel Timing Diagram**

# SECTION 23
# MECHANICAL DATA AND ORDERING INFORMATION

## 23.1 ORDERING INFORMATION

| PACKAGE TYPE | FREQUENCY (MHZ) | TEMPERATURE | ORDER NUMBER |
|---|---|---|---|
| Plastic Ball Grid Array (ZP Suffix) | 25(IMP), 60(DSP) | 0° to 70°C | MC68356ZP25 |

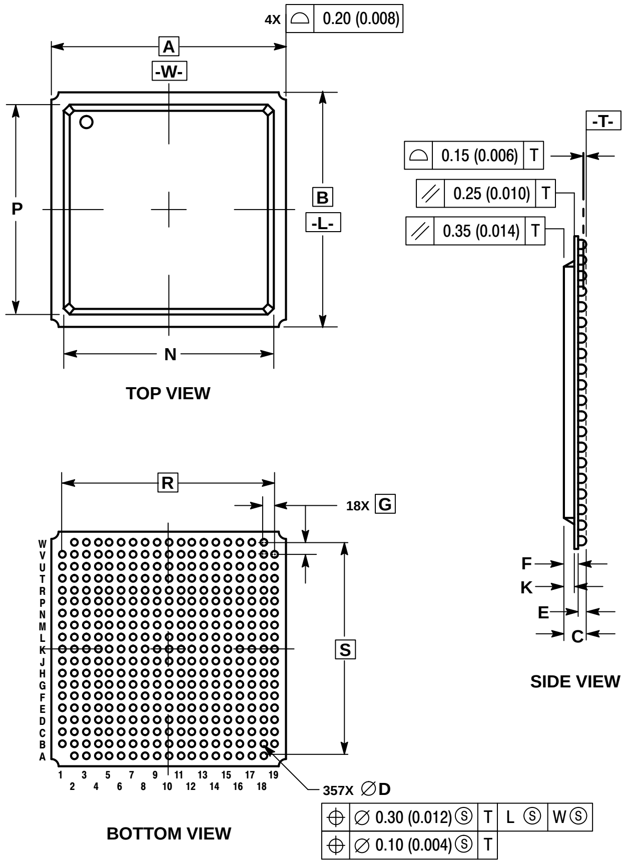
## 23.2 PIN ASSIGNMENTS – PBGA-TOP VIEW



| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | pd[10] | pd[8] | pd[3] | irq7 | d[0] | d[4] | d[1] | d[2] | d[3] | d[5] | vddl | d[6] | d[7] | d[29] | dp2 | clkout | ipa3 | | W |
| pd[14] | pd[13] | pd[9] | pd[6] | Spare3 | irq0 | d[13] | d[27] | d[10] | d[14] | d[18] | d[20] | d[24] | d[28] | dp1 | dp3 | dp0 | N/C | vsssyn1 | V |
| pa[0] | pb[14] | pd[15] | pd[4] | pd[5] | irq1 | d[8] | d[23] | d[11] | d[16] | d[19] | d[21] | d[26] | d[30] | ipa5 | ipa4 | ipa2 | N/C | vsssyn | U |
| pa[1] | pc[5] | pc[4] | pd[11] | pd[7] | vddh | d[12] | d[17] | d[9] | d[15] | d[22] | d[25] | d[31] | ipa6 | ipa0 | ipa1 | ipa7 | xfc | vddsyn | T |
| pc[6] | pa[2] | pb[15] | pd[12] | | | | | | | | | | | | wait_b | wait_a | poreset | kapwr | R |
| pa[4] | pb[17] | pa[3] | vddh | | | | | | | | | | | | vddh | rstconf | sreset | xtal | P |
| pb[19] | pa[5] | pb[18] | pb[16] | | | | | | | | | | | | hreset | texp | extclk | extal | N |
| pa[7] | pc[8] | pa[6] | pc[7] | | | | | | | | | | | | modck2 | baddr28 | baddr29 | vddl | M |
| pb[22] | pc[9] | pa[8] | pb[20] | | | | | | | | | | | | op0 | as | op1 | modck1 | L |
| pc[10] | pa[9] | pb[23] | pb[21] | | | | | | | | | | | | baddr30 | ipb6 | alea | irq4 | K |
| pc[11] | pb[24] | pa[10] | pb[25] | | | | | | | | | | | | ipb5 | ipb1 | ipb2 | aleb | J |
| vddl | Spare2 | tdi | tck | | | | | | | | | | | | Spare4 | irq2 | ipb0 | ipb7 | H |
| trst | tms | tdo | pa[11] | | | | | | | | | | | | br | irq6 | ipb4 | ipb3 | G |
| pb[26] | pc[12] | pa[12] | vddh | | | | | | | | | | | | vddh | ts | irq3 | burst | F |
| pb[27] | pc[13] | pa[13] | pb[29] | | | | | | | | | | | | cs3 | bi | bg | bb | E |
| pb[28] | pc[14] | pa[14] | pc[15] | a[8] | N/C | N/C | a[15] | a[19] | a[25] | a[18] | bsa0 | gpla0 | N/C | cs6 | cs2 | gpla5 | bdip | tea | D |
| pb[30] | pa[15] | pb[31] | a[3] | a[9] | a[12] | a[16] | a[20] | a[24] | a[26] | tsiz1 | bsa1 | we0 | gpla1 | gpla3 | cs7 | cs0 | ta | gpla4 | C |
| a[0] | a[1] | a[4] | a[6] | a[10] | a[13] | a[17] | a[21] | a[23] | a[22] | tsiz0 | bsa3 | Spare1 | we2 | gpla2 | cs5 | ce1a | wr | gplb4 | B |
| | a[2] | a[5] | a[7] | a[11] | a[14] | a[27] | a[29] | a[30] | a[28] | a[31] | vddl | bsa2 | we1 | we3 | cs4 | ce2a | cs1 | | A |
| 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | |

## 23.3 PACKAGE DIMENSIONS–PLASTIC BALL GRID ARRAY (PBGA)

For more information on the printed circuit board layout of the PBGA package, including thermal via design and suggested pad layout, please refer to AN-1231/D, *Plastic Ball Grid Array Application Note* available from your local Motorola sales office.



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.

| | MILLIMETERS | | INCHES | |
|---|---|---|---|---|
| DIM | MIN | MAX | MIN | MAX |
| A | 25.00 BSC | | 0.984 BSC | |
| B | 25.00 BSC | | 0.984 BSC | |
| C | --- | 2.05 | --- | 0.081 |
| D | 0.60 | 0.90 | 0.024 | 0.035 |
| E | 0.50 | 0.70 | 0.020 | 0.028 |
| F | 0.95 | 1.35 | 0.037 | 0.053 |
| G | 1.27 BSC | | 0.50 BSC | |
| K | 0.70 | 0.90 | 0.028 | 0.035 |
| N | 22.40 | 22.60 | 0.882 | 0.890 |
| P | 22.40 | 22.60 | 0.882 | 0.890 |
| R | 22.86 BSC | | 0.900 BSC | |
| S | 22.86 BSC | | 0.900 BSC | |

# SECTION 24
# TERMINOLOGY GLOSSARY

This section provides the definitions to some of the key terms used in this document.

**address demunging**

Reversal of a munge operation on an address. This has the effect of restoring the original address.

**address munging**

Address modification in a way that the low-order three bits of the address are exclusive-OR'ed with a three-bit value that depends on the length of the operand (refer to the *PowerPC™ Microprocessor Family: The Programming Environments (MPGFPE/AD)*.

**atomic cycle**

If multiple bus transactions by a bus master occur in a sequence where the master retains ownership of the bus during the duration of the sequence, thus preventing other master(s) from transferring in the middle of the sequence, the sequence is considered atomic.

**autobaud**

The process of determining a serial data rate by timing the width of a single bit.

**big-endian**

An ordering of the bytes within a word where the least-significant byte is at the highest address and vice versa. For example, a 32-bit wide data bus with big-endian, the least-significant byte is located on data bus bits 24-31 and the most-significant byte is on bits 0-7.

**blockage**

The interval from the time an instruction begins execution until its execution unit is available for a subsequent instruction (AND has 1 clock latency and 1 clock blockage).

**boundedly undefined**

Results of a given (not defined) instruction are boundedly undefined if they could have been achieved by executing an arbitrary sequence of defined instructions in valid form starting in the state the machine was before the attempt was made to execute the given instruction.

**breakpoint**

An event that forces the machine to branch into a breakpoint exception routine.

**bubble**

A number inside a circle that is used to identify specific terms in AC timing diagrams.

**burst**

A bus transfer that has more than one piece of data associated with it.

**burst length**

The number of data associated with a burst cycle. For example, a burst length of four has four data pieces (four beats) associated with it.

**bus park**

Keeps the bus granted to a bus master although it has completed the bus cycle. This allows the same master to make the next transfer without having to rearbitrate for the bus.

**copyback**

Updates to external memory are delayed until forced by the user program or a transfer of bus control to an external master. At the time of forced update or relinquishment of the bus, all changes to the cache are written to external memory. Until that time, cache and external memory are not coherent.

**critical-data first**

This feature allows the data transferred during the burst cycle to be organized where the word or data needed first is the first one to transfer within the burst-data block. The order of transferring can be sequential and usually wraps back to the word (or data) zero. For example, 1→ 2 → 3 → 0 for a sequence of four data with data 1 as the critical data.

**datastream**

A sequence of information to be processed by the CPU.

**early termination**

Some burst protocols specify the burst length at the beginning of the transfer. Early termination allows the burst to be terminated before all data beats are transferred.

**exception**

An error, unusual condition, or external signal that can set a status bit. It may or may not cause an interrupt, depending on whether or not the corresponding interrupt is enabled.

**execution serialization**

Instruction issue is halted until all instructions that are currently in progress complete execution (all internal pipeline stages and instruction buffers have emptied and all outstanding memory transactions are completed).

**execution stream**

The combination of instructions and data on which the CPU operates.

**fetch serialization**

Instruction fetch is halted until all instructions currently in the processor have completed execution (all issued instructions as well as the prefetched instructions waiting to be issued). The machine after fetch serialization is said to be completely synchronized.

**fixed transaction**

A bus transaction that combines the address and data phase of the bus cycle into a single event.

**flow control instruction**

One of: B BR BCR BCC RFI SC and sometimes ISYNC.

**half-word**

A half-word consists of 2 bytes or 16 bits.

**instruction done**

Execution finished and results written back.

**instruction execution time**

All the time between taken and done.

**instruction fetch**

Reading the instruction data received from the instruction memory (I-Cache, Flash).

**instruction issue**

Driving valid instruction bits inside the core.

**instruction retire**

The instruction and all preceding instructions in the program finished execution with no errors. Retired instructions are said to be architecturally executed.

**instruction stream**

A sequence of operands to be executed by the CPU.

**instruction taken**

All resources to perform the instruction are ready and the core begins executing it.

**internal bus**

The bus connecting the CPU and SIU.

**interrupt**

The act of changing the machine state register and other parts of the machine state in response to an exception.

**latency**

The interval from the time an instruction begins execution until it produces a result that is available for use by a subsequent instruction.

**little-endian**

Byte ordering that assigns the lowest address to the lowest-order 8 bits of the scalar.

**master**

A device on the bus that requests bus ownership and initiates the bus cycles.

**memory controller**

A functional logic section of the MPC821. It's primary function is to provide the controls for the external bus memories and I/O devices.

**no operation (NOP)**

An instruction whose sole function is to increment the Program Counter, but which affects no changes to any registers or memory.

**pace control**

Controls the rate of the data flow between the master and slave. The MPC821 burst mechanisms allow this to be controlled by the slave and is useful in slowing down the data transfer rate. Slave delay can be used in place of pace control. It means the data pace can be slowed down by the slave.

**pipeline**

The act of initiating a bus cycle while another bus cycle is in progress. Thus, the bus can have multiple bus cycles pending at a time.

**scan chain**

The peripheral buffers of a device, linked in JTAG test mode, that are addressed in a shift register fashion.

**scoreboard**

A register tracking system that ensures that values are not pulled from a register before they are updated by a previous instruction.

**sequential instruction**

Any instruction that is not a flow control instruction and not ISYNC.

**slave**

A device that responds to the master's address. A slave receives data on a write cycle and gives data to the master on a read cycle.

**snoop**

The act of monitoring external bus activity by alternate bus masters. By snooping these external accesses, a CPU can identify accesses to memory locations that contain dirty data and possibly halt activity to supply correct data.

**swap**

Four byte lanes, reversing (lane 0 to lane 3, lane 1 to lane 2, lane 2 to lane 1 and lane 3 to lane 0).

**tablewalk**

An index value is used to identify an entry point in a tree structure that is traversed until a pointer is found. The system 'walks' through a table of pointers to it's end.

**transaction**

A bus transaction consists of an address transfer (address phase) and data transfers (data phase).

**time-division multiplex (TDM)**

Any serial channel that is divided into channels separated by time.

**watchpoint**

An event that is reported, but does not change the timing of the machine.

**word**

A word consists of 4 bytes or 32 bits.

**writethrough**

Continuous updates, as they occur, of external memory so that cache and memory maintain coherency at all times.

# APPENDIX A
# SERIAL COMMUNICATION PERFORMANCE

The MPC821 operating at 25 MHz is designed to support unrestricted operation of the high-level data link control (HDLC) or transparent protocol running simultaneously on two serial communication controllers (SCCs) at 2.048 Mbps. The MPC821 can also support one Ethernet channel at 10 Mbps and one HDLC or transparent channel at 1 Mbps. The physical clocking limit of the SCCs is higher than the sustained serial bit rate. This limit is given as a 1:2.25 ratio between the sync clock (a clock generated in the clock synthesizer that can be as fast as the 25-MHz system clock) and the serial clock. For example, with a sync clock of 25 MHz, the SCCs can be clocked at 11.1 MHz. This clocking scheme allows the SCCs to handle high-speed bursts of data bits for short periods of time subject to the FIFO sizes.

When the SCCs are connected to a time-division multiplexed channel using the time-slot assigner present on the MPC821, the SCC physical clocking limit is a 1:2.5 ratio between the sync clock and serial clock. Therefore, the SCCs can be connected to a 10.0-MHz time-division multiplexed channel with a 25-MHz MPC821. This clocking scheme allows the SCCs to handle high-speed bursts of data bits for short periods of time subject to the FIFO sizes. Other devices that offer a higher HDLC performance than the MPC821 are the Motorola MC68605 1984 CCITT X.25 LAPB controller and MC68606 CCITT Q.921 multilink LAPD controller. The MC68605 and MC68606 perform the full data-link layer protocol and support various transparent modes within HDLC-framed operation at speeds of at least 10 Mbps. The performance figures listed in Table A-1 are for a 25-MHz system clock only. Notice that, in general, performance scales linearly with frequency so that a combination of protocols over the MPC821’s performance limitation at 25 MHz could be possible at 50 MHz.

## A.1 CHANNEL COMBINATIONS

When operating the multiple HDLC/transparent channel protocol, most of the processing load falls on the CPM RISC. Protocols other than the multiple HDLC/transparent protocol rely on the hardware support built into the serial channels. Combining the SCC and the RISC results in the ability to attain throughput rates of up to 8 Mbps on HDLC and transparent mode.

The multiple channel protocol does not rely on the SCC and operates transparently to the user performing serial to parallel conversion and vice versa. All bit manipulation is performed in the RISC software or hardware, causing the CPM to experience a heavier load when operating in multiple channel mode. The heavier load occurs even if all time-slots are concatenated to one logical channel.

The following table and equation guides the user in calculating the load. Notice that CPM load estimation is not a completely linear equation. There is a gray area near the maximum limit and it is recommended that the user test his operations on target hardware to determine the exact load.

Table A-1. MPC821/MPC821EN Performance Table at 25 MHz

| PROTOCOL | SPEED[2] |
|---|---|
| Transparent | 8 Mbps FD |
| HDLC | 8 Mbps FD |
| UART | 2.4 Mbps FD |
| Ethernet | 22 Mbps HD |
| SMC transparent | 1.5 Mbps FD |
| SMC UART | 220 kbps FD |
| Bisync | 1.5 Mbps FD |
| IDMA Mem to Mem | 5.7MBYTE/s[3] |
| ““ with burst aligned source/dest Addr | 10.4MBYTE/s[3] |
| IDMA Dual Addr Periph to Mem | 2.2MBYTE/s[3] |
| IDMA Dual Addr Mem to Periph | 1.6MBYTE/s[3] |
| IDMA Single Addr Periph to Mem | 5MBYTE/s[3] |
| IDMA Single Addr Mem to Periph | 5MBYTE/s[3] |
| Async HDLC | 3 Mbps |
| $I^2C$ | 520Khz |
| SPI-16bit mode | 3.125Mbps |
| SPI-8bit mode[1] | 500kbps |
| PIP | 625kBYTE/s |

NOTE: 1. 500Kbits/sec is the maximum throughput when no other peripherals (SCCs, SMCs) are being used. Load on those peripherals will further reduce the maximum data rate through the SPI.

2. Performance scales linearly with device operating frequency

3. IDMA transfer rates are independent of bus cycle length

## Example #1

MPC821 operating 1 × 10 Mbit Ethernet in half duplex, 1 × 2 Mbit HDLC, 1 × 19.2 kbit SMC UART and 1 × 38 kbit SMC UART. The following equation applies:

$$\frac{10}{22} + \frac{2}{8} + \frac{0.0192}{0.22} + \frac{0.038}{0.22} \quad = \quad \mathbf{0.96 < 1}$$

If the results of a calculation is greater than one, this will not work. Some configurations are scalable in frequency. Multiply the resultant by the ratio of ‘25/f’, where f = the Frequency in MHz and if that resultant is less than one, then that channel combination will most likely perform satisfactorily. The equations are scalable to frequency with the exception of the Ethernet protocol which is somewhat nonlinear and is difficult to take into account. Notice that any calculation with a result close to one and above is considered to be in a gray area and should be run as a test case in a lab set up before implemention.

### Example #2

A block of 512 kbytes transferred by IDMA in single address mode from memory to a peripheral, one asynchronous HDLC at 1Mbps, and one SMC UART at 38.4 Kbit.

$$\frac{0.512}{5} + \frac{1}{3} + \frac{0.038}{0.22} = \mathbf{0.69}$$

**NOTE**

In the case of IDMA, this process calculates the peak CPM utilization, not the sustained rate. By nature, IDMA transfers occur in random intervals and are not consistent bit rates when compared to the serial channel operation.

### Example #3

Two ethernet channels at 10 Mb/s and one SMC UART at 115.2 K baud running at 40 MHz.

$$\left[\frac{2*10}{22} + \frac{0.1152}{0.22}\right] = 1.43 \text{ X } \frac{25}{40} = \mathbf{0.90}$$

# APPENDIX B
# QUICK REFERENCE GUIDE TO MPC821 REGISTERS

## B.1 CORE CONTROL REGISTERS

The following are CPU and SIU control registers implemented within the MPC821. Access to these special registers occurs via the mtspr and mfspr instructions. Special purpose registers must access memory indirectly via the general-purpose registers, which can be written as ‘rN’ or simply as ‘N.’ An assembler directive for accessing these registers is as follows:

```
mtspr 8, r0      /* MOVE TO THE LINK REGISTER */
mfspr 3, 638     /* MOVES FROM THE IMMR REGISTER TO REGISTER 3 */
```

**NOTE**

The IMMR (special purpose register #638) setting determines the base address for all on-chip 821 modules that are not dedicated to core operation. During HRESET, the internal base address is initially set to one of four addresses determined by the ISB field in the HARD RESET CONFIGURATION WORD. If, during HRESET, the RSTCONF pin is not driven low and the HARD RESET CONFIGURATION WORD is not placed on the data bus, the initial base address is $00000000. After the reset process is finished, the internal base address can be moved to any value by writing to the IMMR.

### Table B-1. Standard Special Purpose Registers

| SPR | | | NAME | DESCRIPTION |
|---|---|---|---|---|
| DECIMAL | SPR $_{5:9}$ | SPR $_{0:4}$ | | |
| 1 | 00000 | 00001 | XER | Fixd Pt Exception Cause |
| 8 | 00000 | 01000 | LR | Link Register |
| 9 | 00000 | 01001 | CTR | Count Register |
| 18 | 00000 | 10010 | DSISR | Data Storage Int. Status |
| 19 | 00000 | 10011 | DAR | Data Address Register |
| 22 | 00000 | 10110 | DEC | Decrementer |
| 26 | 00000 | 11010 | SRR0 | Machine Status Save/Rest |
| 27 | 00000 | 11011 | SRR1 | Machine Status Save/Rest |
| 272 | 01000 | 10000 | SPRG0 | Address of except. handler |
| 273 | 01000 | 10001 | SPRG1 | Exception Handler Scratch |
| 274 | 01000 | 10010 | SPRG2 | Scratch Register 2 |
| 275 | 01000 | 10011 | SPRG3 | Scratch Register 3 |
| 287 | 01000 | 11111 | PVR | Processor Version |

### Table B-2. Standard Timebase Register Mapping

| SPR | | | NAME | DESCRIPTION |
|---|---|---|---|---|
| DECIMAL | SPR $_{5:9}$ | SPR $_{0:4}$ | | |
| 268 | 01000 | 01100 | TB read [2] | Read Lower Timebase |
| 269 | 01000 | 01101 | TBU read [2] | Read Upper Timebase |
| 284 | 01000 | 11100 | TB write [3] | Write Lower Timebase |
| 285 | 01000 | 11101 | TBU write [3] | Write Upper Timebase |

NOTE: 1. Extended opcode for mftb, 371 rather then 339.

2. Any write (mtspr) to this address, results in an implementation dependent software emulation interrupt.

3. Any write (mftb) to this address, results in an implementation dependent software emulation interrupt.

### Table B-3. Added Special Purpose Registers

| SPR | | | NAME | DESCRIPTION |
|---|---|---|---|---|
| DECIMAL | SPR $_{5:9}$ | SPR $_{0:4}$ | | |
| 80 | 00010 | 10000 | EIE* | External Interrupt Enable |
| 81 | 00010 | 10001 | EID | External Insterrupt Disable |
| 82 | 00010 | 10010 | NRI | Non-Recoverable Int. |
| 144 | 00100 | 10000 | CMPA | Compare A Value |
| 145 | 00100 | 10001 | CMPB | Compare B Value |
| 146 | 00100 | 10010 | CMPC | Compare C Value |
| 147 | 00100 | 10011 | CMPD | Compare D Value |
| 148 | 00100 | 10100 | ICR | Interrupt Cause |
| 149 | 00100 | 10101 | DER | Debug Enable |
| 150 | 00100 | 10110 | COUNTA | Instr/Load Watchpt Count |
| 151 | 00100 | 10111 | COUNTB | Instr/Load Watchpt Count |
| 152 | 00100 | 11000 | CMPE | Compare E Value |
| 153 | 00100 | 11001 | CMPF | Compare F Value |
| 154 | 00100 | 11010 | CMPG | Compare G Value |
| 155 | 00100 | 11011 | CMPH | Compare H Value |
| 156 | 00100 | 11100 | LCTRL1 | Load/Store Compare Cntl |
| 157 | 00100 | 11101 | LCTRL2 | Load/Store AND-OR |
| 158 | 00100 | 11110 | ICTRL | Instr. Support Cntl |
| 159 | 00100 | 11111 | BAR | Breakpt Address |
| 630 | 10011 | 10110 | DPDR | Develop. Port Data |
| 631 | 10011 | 10111 | DPIR | Develop. Port Instr. |
| 638 | 10011 | 11110 | IMMR | Internal Memory Map |
| 560 | 10001 | 10000 | IC_CST | Instr. Cache Cntl/Stat |
| 561 | 10001 | 10001 | IC_ADR | Instr. Cache Address |
| 562 | 10001 | 10010 | IC_DAT | Instr. Cache Data |
| 568 | 10001 | 11000 | DC_CST | Data Cache Cntl/Stat |
| 569 | 10001 | 11001 | DC_ADR | Data Cache Address |
| 570 | 10001 | 11010 | DC_DAT | Data Cache Data |
| 784 | 11000 | 10000 | MI_CTR | Instruction MMU Cntl |
| 786 | 11000 | 10010 | MI_AP | Instr. MMU Access Perm. |
| 787 | 11000 | 10011 | MI_EPN | Instr. MMU Effect. Pg Num |
| 789 | 11000 | 10101 | MI_TWC<br>(MI_L1DL2P) | Instr. MMU Tblewalk Cntl |

Table B-3. Added Special Purpose Registers (Continued)

| SPR | | | NAME | DESCRIPTION |
|---|---|---|---|---|
| DECIMAL | SPR $_{5:9}$ | SPR $_{0:4}$ | | |
| 790 | 11000 | 10110 | MI_RPN | Instr. MMU Real Pg Num |
| 816 | 11001 | 10000 | MI_DBCAM | Data MMU CAM Read |
| 817 | 11001 | 10001 | MI_DBRAM0 | Data MMU CAM Read 0 |
| 818 | 11001 | 10010 | MI_DBRAM1 | Data MMU CAM Read 1 |
| 792 | 11000 | 11000 | MD_CTR | Data MMU Cntl |
| 793 | 11000 | 11001 | M_CASID | CASID |
| 794 | 11000 | 11010 | MD_AP | Data MMU Access Priv |
| 795 | 11000 | 11011 | MD_EPN | Data MMU Eff. Page Num |
| 796 | 11000 | 11100 | M_TWB<br>(MD_L1P) | MMU TableWalk Base |
| 797 | 11000 | 11101 | MD_TWC<br>(MD_L1DL2P) | Data MMU TbleWalk Cntl |
| 798 | 11000 | 11110 | MD_RPN | Data MMU Real Pg Num |
| 799 | 11000 | 11111 | M_TW<br>(M_SAVE) | MMU TableWalk Special |
| 824 | 11001 | 11000 | MD_DBCAM | Data MMU CAM Read |
| 825 | 11001 | 11001 | MD_DBRAM0 | Data MMU CAM Read0 |
| 826 | 11001 | 11010 | MD_DBRAM1 | Data MMU CAM Read1 |

Table B-4. Other Control Registers

| DESCRIPTION | NAME |
|---|---|
| Machine State Register | MSR |
| Condition Register | CR |

## B.2 INTERNALLY MAPPED REGISTERS

The MPC821 internal memory resources are mapped within a contiguous block of storage. The size of the internal space in the MPC821 is 16 Kbytes. The location of this block within the global 4 Gigabyte real storage space can be mapped on a 64 Kbytes resolution through an implementation specific special register, and the internal memory map register (IMMR). The following table defines the internal memory map of the MPC821.

To address the registers below, the number in the internal address column is added as an offset to the IMMR (special purpose register number 638). For example, if the most-significant bits of the IMMR are set to $FF00, then to initialize the SDMA Configuration Register (SDCR) the user should write to location $FF000030. To access the dual port RAM, the user should address locations $FF002000 through $FF004000.

### Table B-5. MPC821 Internal Memory Map

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| General SIU | | | |
| 000 | SIUMCR | SIU Module configuration | 32 |
| 004 | SYPCR | System Protection Control | 32 |
| 008 | SWT | SW Watchdog Timer Current Value | 32 |
| 00E | SWSR | Software Service | 16 |
| 010 | SIPEND | Interrupt Pend Register | 32 |
| 014 | SIMASK | Interrupt Mask | 32 |
| 018 | SIEL | Interrupt Edge Level Mask | 32 |
| 01C | SIVEC | Interrupt Vector | 32 |
| 020 | TESR | Transfer Error Status | 32 |
| 024 to 02F | Reserved | | |
| 030 | SDCR | SDMA Configuration Register | 32 |
| 034 to 07f | Reserved | | |
| PCMCIA | | | |
| 080 | PBR0 | PCMCIA Base | 32 |
| 084 | POR0 | PCMCIA Option | 32 |
| 088 | PBR1 | PCMCIA Base | 32 |
| 08c | POR1 | PCMCIA Option | 32 |
| 090 | PBR2 | PCMCIA Base | 32 |
| 094 | POR2 | PCMCIA Option | 32 |
| 098 | PBR3 | PCMCIA Base | 32 |
| 09c | POR3 | PCMCIA Option | 32 |
| 0a0 | PBR4 | PCMCIA Base | 32 |
| 0a4 | POR4 | PCMCIA Option | 32 |
| 0a8 | PBR5 | PCMCIA Base | 32 |
| 0ac | POR5 | PCMCIA Option | 32 |
| 0b0 | PBR6 | PCMCIA Base | 32 |
| 0b4 | POR6 | PCMCIA Option | 32 |
| 0b8 | PBR7 | PCMCIA Base | 32 |
| 0bc | POR7 | PCMCIA Option | 32 |
| 0c0 to 0df | Reserved | | |

## Table B-5. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 0e0 | PGCRA | PCMCIA Slot A Control | 32 |
| 0e4 | PGCRB | PCMCIA Slot B Control | 32 |
| 0e8 | PSCR | PCMCIA Status | 32 |
| 0ec to 0ef | Reserved | | |
| 0f0 | PIPR | PCMCIA Pins Value | 32 |
| 0f4 to 0f7 | Reserved | | |
| 0f8 | PER | PCMCIA Enable | 32 |
| 0fc to 0ff | Reserved | | |
| MEMC | | | |
| 100 | BR0 | Base Register | 32 |
| 104 | OR0 | Option Register | 32 |
| 108 | BR1 | Base Register | 32 |
| 10c | OR1 | Option Register | 32 |
| 110 | BR2 | Base Register | 32 |
| 114 | OR2 | Option Register | 32 |
| 118 | BR3 | Base Regster | 32 |
| 11c | OR3 | Option Register | 32 |
| 120 | BR4 | Base Register | 32 |
| 124 | OR4 | Option Register | 32 |
| 128 | BR5 | Base Register | 32 |
| 12c | OR5 | Option Register | 32 |
| 130 | BR6 | Base Register | 32 |
| 134 | OR6 | Option Register | 32 |
| 138 | BR7 | Base Register | 32 |
| 13c | OR7 | Option Register | 32 |
| 140 to 163 | Reserved | | |
| 164 | MAR | Memory Address | 32 |
| 168 | MCR | Memory Command | 32 |
| 16C to 16F | Reserved | | |
| 170 | MAMR | Machine A Mode | 32 |
| 174 | MBMR | Machine B Mode | 32 |
| 178 | MSTAT | Memory Status | 16 |

## Table B-5. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 17A | MPTPR | Memory Periodic Timer Prescaler | 16 |
| 17C | MDR | Memory Data | 32 |
| 180 to 1FF | Reserved | | |
| SYSTEM INTEGRATION TIMERS | | | |
| 200 | TBSCR | Timebase Status and Control | 16 |
| 204 | TBREFF0 | Timebase Reference 0 | 32 |
| 208 | TBREFF1 | Timebase Reference 1 | 32 |
| 20c to 21f | Reserved | | |
| 220 | RTCSC | Real-Time Clock Status and Control | 16 |
| 224 | RTC | Real-Time Clock | 32 |
| 228 | RTSEC | Real-Time Alarm Seconds | 32 |
| 22c | RTCAL | Real-Time Alarm | 32 |
| 230 to 23f | Reserved | | |
| 240 | PISCR | PIT Status and Control | 16 |
| 244 | PITC | PIT Count | 32 |
| 248 | PITR | PIT | 32 |
| 24c to 27f | Reserved | | |
| CLOCKS AND RESETS | | | |
| 280 | SCCR | System Clock Control | 32 |
| 284 | PLPRCR | PLL, Low Power and Reset Control Register | 32 |
| 288 | RSR | Reset Status Register | 32 |
| 28c to 2ff | Reserved | | |
| SYSTEM INTEGRATION TIMERS KEYS | | | |
| 300 | TBSCRK | Timebase Status and Control Key | 32 |
| 304 | TBREFF0K | Timebase Reference 0 Key | 32 |
| 308 | TBREFF1K | Timebase Reference 1 Key | 32 |
| 30C | TBK | Timebase and Decrementer Key | 32 |
| 310 to 31f | Reserved | | |
| 320 | RTCSCK | Real-Time Clock Status and Control Key | 32 |
| 324 | RTCK | Real-Time Clock Key | 32 |
| 328 | RTSECK | Real-Time Alarm Seconds Key | 32 |
| 32c | RTCALK | Real-Time Alarm Key | 32 |

## Table B-5. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 330 to 33f | Reserved | | 8x32 |
| 340 | PISCRK | PIT Status and Control Key | 32 |
| 344 | PITCK | PIT Count Key | 32 |
| 348 to 37f | Reserved | | |
| CLOCKS AND RESET KEYS | | | |
| 380 | SCCRK | System Clock Control Key | 32 |
| 384 | PLPRCRK | PLL, Low Power and Reset Control Register Key | 32 |
| 388 | RSRK | Reset Status Register Key | 32 |
| 38c to 3ff | Reserved | | |
| 400 to 7ff | Reserved | | |
| 800 to 85F | Reserved | | |
| I2C | | | |
| 860 | I2MOD | I2C Mode Register | 8 |
| 864 | I2ADD | I2C Address Register | 8 |
| 868 | I2BRG | I2C BRG Register | 8 |
| 86C | I2COM | I2C Command Register | 8 |
| 870 | I2CER | I2C Event Register | 8 |
| 874 | I2CMR | I2C Mask Register | 8 |
| DMA | | | |
| 900 -903 | Reserved | | |
| 904 | SDAR | SDMA Address Register | 32 |
| 908 | SDSR | SDMA Status Register | 8 |
| 909-90b | Reserved | | |
| 90c | SDMR | SDMA Mask Register | 8 |
| 90d-90f | Reserved | | |
| 910 | IDSR1 | IDMA1 Status Register | 8 |
| 911-913 | Reserved | | |
| 914 | IDMR1 | IDMA1 Mask Register | 8 |
| 915-917 | Reserved | | |
| 918 | IDSR2 | IDMA2 Status Register | 8 |
| 919-91b | Reserved | | |
| 91c | IDMR2 | IDMA2 Mask Register | 8 |

Table B-5. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 91d-92F | Reserved | | |
| CPM IntERRUPT CONTROL | | | |
| 930 | CIVR | CP Interrupt Vector Register | 16 |
| 932 to 93F | Reserved | | |
| 940 | CICR | CP Interrupt Configuration Register | 32 |
| 944 | CIPR | CP Interrupt Pending Register | 32 |
| 948 | CIMR | CP Interrupt Mask Register | 32 |
| 94C | CISR | CP In-Service Register | 32 |
| INPUT/OUTPUT PORT | | | |
| 950 | PADIR | Port A Data Direction Register | 16 |
| 952 | PAPAR | Port A Pin Assignment Register | 16 |
| 954 | PAODR | Port A Open Drain Register | 16 |
| 956 | PADAT | Port A Data Register | 16 |
| 958 to 95f | Reserved | | |
| 960 | PCDIR | Port C Data Direction Register | 16 |
| 962 | PCPAR | Port C Pin Assignment Register | 16 |
| 964 | PCSO | Port C Special Options | 16 |
| 966 | PCDAT | Port C Data Register | 16 |
| 968 | PCINT | Port C Interrupt Control Register | 16 |
| 96a to 96f | Reserved | | |
| 970 | PDDIR | Port D Data Direction Register | 16 |
| 972 | PDPAR | Port D Pin Assignment Register | 16 |
| 974 | Reserved | Reserved | 16 |
| 976 | PDDAT | Port D Data Register | 16 |
| 978 to 97f | Reserved | | |
| CPM TIMERS | | | |
| 980 | TGCR | Timer Global Configuration Register | 16 |
| 982 to 98f | Reserved | | |
| 990 | TMR1 | Timer1 Mode Register | 16 |
| 992 | TMR2 | Timer2 Mode Register | 16 |
| 994 | TRR1 | Timer1 Reference Register | 16 |
| 996 | TRR2 | Timer2 Reference Register | 16 |

Table B-5. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 998 | TCR1 | Timer1 Capture Register | 16 |
| 99A | TCR2 | Timer2 Capture Register | 16 |
| 99C | TCN1 | Timer1 Counter | 16 |
| 99E | TCN2 | Timer2 Counter | 16 |
| 9A0 | TMR3 | Timer3 Mode Register | 16 |
| 9A2 | TMR4 | Timer4 Mode Register | 16 |
| 9A4 | TRR3 | Timer3 Reference Register | 16 |
| 9A6 | TRR4 | Timer4 Reference Register | 16 |
| 9A8 | TCR3 | Timer3 Capture Register | 16 |
| 9AA | TCR4 | Timer4 Capture Register | 16 |
| 9AC | TCN3 | Timer3 Counter | 16 |
| 9AE | TCN4 | Timer4 Counter | 16 |
| 9B0 | TER1 | Timer1 Event Register | 16 |
| 9B2 | TER2 | Timer2 Event Register | 16 |
| 9B4 | TER3 | Timer3 Event Register | 16 |
| 9B6 | TER4 | Timer4 Event Register | 16 |
| 9b8 to 9bf | Reserved | | |
| COMMUNICATION PROCESSOR | | | |
| 9CO | CPCR | Communications Processor Command Register | 16 |
| 9C4 | RCCR | RISC Configuration Register | 16 |
| 9C6 | RES | Reserved | 8 |
| 9C7 | RES | Reserved | 8 |
| 9C8 | RES | Reserved | 32 |
| 9CC | CPMCR1 | Comm. Processor Module Control Register 1 | 16 |
| 9CE | CPMCR2 | Comm. Processor Module Control Register 2 | 16 |
| 9D0 | CPMCR3 | Comm. Processor Module Control Registerr 3 | 16 |
| 9D2 | CPMCR4 | Comm. Processor Module Control Register 4 | 16 |
| 9D6 | RTER | RISC Timers Event Register | 16 |
| 9DA | RTMR | RISC Timers Mask Register | 16 |
| 9dc to 9ef | Reserved | | |
| BRGs | | | |
| 9F0 | BRGC1 | BRG1 Configuration Register | 32 |

Table B-5. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| 9F4 | BRGC2 | BRG2 Configuration Register | 32 |
| 9F8 | BRGC3 | BRG3 Configuration Register | 32 |
| 9FC | BRGC4 | BRG4 Configuration Register | 32 |
| SCC1 | | | |
| a00 | GSMR_L1 | SCC1 General Mode Register | 32 |
| a04 | GSMR_H1 | SCC1 General Mode Register | 32 |
| a08 | PSMR1 | SCC1 Protocol Specific Mode Register | 16 |
| a0c | TODR1 | SCC1 Transmit-On-Demand | 16 |
| a0e | DSR1 | SCC1 Data Sync. Register | 16 |
| a10 | SCCE1 | SCC1 Event Register | 16 |
| a14 | SCCM1 | SCC1 Mask Register | 16 |
| a17 | SCCS1 | SCC1 Status Register | 8 |
| a18 to a1f | Reserved | | |
| SCC2 | | | |
| a20 | GSMR_L2 | SCC2 General ModeRegister | 32 |
| a24 | GSMR_H2 | SCC2 General Mode Register | 32 |
| a28 | PSMR2 | SCC2 Protocol Specific Mode Register | 16 |
| a2c | TODR2 | SCC2 Transmit-On-Demand | 16 |
| a2e | DSR2 | SCC2 Data Sync. Register | 16 |
| a30 | SCCE2 | SCC2 Event Register | 16 |
| a34 | SCCM2 | SCC2 Mask Register | 16 |
| a37 | SCCS2 | SCC2 Status Register | 8 |
| a38 to a3f | Reserved | | |
| SCC3 | | | |
| a40 | GSMR_L3 | SCC3 General Mode Register | 32 |
| a44 | GSMR_H3 | SCC3 General Mode Register | 32 |
| a48 | PSMR3 | SCC3 Protocol Specific Mode Register | 16 |
| a4c | TODR3 | SCC3 Transmit-On-Demand | 16 |
| a4e | DSR3 | SCC3 Data Sync. Register | 16 |
| a50 | SCCE3 | SCC3 Event Register | 16 |
| a54 | SCCM3 | SCC3 Mask Register | 16 |
| a57 | SCCS3 | SCC3 Status Register | 8 |

### Table B-5. MPC821 Internal Memory Map (Continued)

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| a58 to a5f | Reserved | | |
| SCC4 | | | |
| a60 | GSMR_L4 | SCC4 General Mode Register | 32 |
| a64 | GSMR_H4 | SCC4 General Mode Register | 32 |
| a68 | PSMR4 | SCC4 Protocol Specific Mode Register | 16 |
| a6c | TODR4 | SCC4 Transmit-On- Demand | 16 |
| a6e | DSR4 | SCC4 Data Sync. Register | 16 |
| a70 | SCCE4 | SCC4 Event Register | 16 |
| a74 | SCCM4 | SCC4 Mask Register | 16 |
| a77 | SCCS4 | SCC4 Status Register | 8 |
| a78 to a81 | Reserved | | |
| SMC1 | | | |
| a82 | SMCMR1 | SMC1 Mode Register | 16 |
| a86 | SMCE1 | SMC1 Event Register | 8 |
| a8a | SMCM1 | SMC1 Mask Register | 8 |
| a8C | Reserved | | |
| SMC2 | | | |
| a92 | SMCMR2 | SMC2 Mode Register | 16 |
| a96 | SMCE2 | SMC2 or PIP Event Register | 8 |
| a9a | SMCM2 | SMC2 Mask Register | 8 |
| a9C | Reserved | | |
| SPI | | | |
| aA0 | SPMODE | SPI Mode Register | 16 |
| aA6 | SPIE | SPI Event Register | 8 |
| aAA | SPIM | SPI Mask Register | 8 |
| aAD | SPCOM | SPI Command Register | 8 |
| PIP | | | |
| aB2 | PIPC | PIP Configuration Register | 16 |
| aB6 | PTPR | PIP Timing Parameters Register | 16 |
| aB8 | PBDIR | Port B Data Direction Register | 32 |
| aBC | PBPAR | Port B Pin Assignment Register | 32 |
| aC2 | PBODR | Port B Open Drain Register | 16 |

**Table B-5. MPC821 Internal Memory Map (Continued)**

| INTERNAL ADDRESS | MNEMONIC | NAME | SIZE |
|---|---|---|---|
| aC4 | PBDAT | Port B Data Register | 32 |
| ac8 to adf | Reserved | | |
| SI | | | |
| aE0 | SIMODE | SI Mode Register | 32 |
| aE4 | SIGMR | SI Global Mode Register | 8 |
| aE6 | SISTR | SI Status Register | 8 |
| aE7 | SICMR | SI Command Register | 8 |
| aE8 | RES | Reserved | |
| aEC | SICR | SI Clock Route | 32 |
| aF0 | SIRP | SI RAM Pointers | 32 |
| aF4 to bFF | Reserved | | |
| c00 to Dff | SIRAM | SI Routing RAM | 512 bytes |
| E00 to FFF | LCDCOLR | LCD Color RAM Table | |
| 1000 to 2000 | Reserved | | |
| 2000 to 4000 | DPRAM | | |

## B.3 PARAMETER RAM ADDRESSING

To address protocol specific areas within the parameter RAM, the offset listed in Table B-6 would be added to the base location of the dual port RAM. Assume that the 15 most-significant bits of the IMMR are set to $FF00. The dual port RAM base address is therefore located at $FF002000, the base address of SCC1’s protocol specific registers at $FF003C00, and SCC2 protocol specific registers at $FF003D00. To initialize the UART parameter MAX_IDL for SCC1, write to location $FF003C38.

Table B-6. Parameter RAM Locations

| PAGE | ADDRESES | PERIPHERAL |
|---|---|---|
| 1 | DPRAM_Base+ $1c00 | SCC1 |
| | DPRAM_Base+ $1c7f | |
| | DPRAM_Base+ $1c80 | I$^2$C |
| | DPRAM_Base+ $1caf | |
| | DPRAM_Base+ $1cb0 | MISC |
| | DPRAM_Base+ $1cbf | |
| | DPRAM_Base+ $1cc0 | IDMA1 |
| | DPRAM_Base+ $1cff | |
| 2 | DPRAM_Base+ $1d00 | SCC2 |
| | DPRAM_Base+ $1d7f | |
| | DPRAM_Base+ $1d80 | SPI |
| | DPRAM_Base+ $1daf | |
| | DPRAM_Base+ $1db0 | Timers |
| | DPRAM_Base+ $1dbf | |
| | DPRAM_Base+ $1dc0 | IDMA2 |
| | DPRAM_Base+ $1dff | |
| 3 | DPRAM_Base+ $1e00 | Reserved |
| | DPRAM_Base+ $1e7f | |
| | DPRAM_Base+ $1e80 | SMC1 |
| | DPRAM_Base+ $1ebf | |
| | DPRAM_Base+ $1ec0 | DSP1 |
| | DPRAM_Base+ $1eff | |
| 4 | DPRAM_Base+ $1f00 | Reserved |
| | DPRAM_Base+ $1f7f | |
| | DPRAM_Base+ $1f80 | SMC2 |
| | DPRAM_Base+ $1fbf | |
| | DPRAM_Base+ $1fc0 | DSP2 |
| | DPRAM_Base+ $1fff | |

### B.3.1 UPM RAM Locations

To access the UPM RAM entries, first write the MDR with a UPM word that is the pattern that controls a single clock cycle of an external memory access. The MDR is at address IMMR+$17C. For a burst read cycle, a valid UPM pattern is $0x0fffcc24. An example of a write to this location is (assuming r3 = $0FFFCC24) mtspr (638)$17C, r3.

| BITS | MNEMONIC | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 0-31 | MD(0:31) | **MEMORY DATA**.This is the data to be written into the RAM array when a WRITE command is supplied to the MCR. This is the Ddata read from the array when a READ command is supplied to the MCR. | |

Next, the MAR should be written with the cycle-type offset within the UPM RAM. For example, if setting up the first cycle of a burst read, the user should write to location IMMR+$168 (MCR) being sure that bits 0:1 are equal to $00 and that bits 26-31 are equal to $08.

MCR Register

| | | | |
|---|---|---|---|
| 0-1 | OP(0:1) | **COMMAND OPCODE**. This field determines which command will be executed by the machine specified in the UM field. | 00 = WRITE. Write the contents of the MDR into the RAM location pointed to by MAD into the UPM specified by UM.<br>01 = READ. Read the contents of the RAM location pointed to by MAD into the UPM specified in UM into the MDR.<br>10 = RUN. Run the pattern written in the RAM array of the UPM specified in UM servicing the memory bank specified in MB. The pattern run starts at the location pointed by MAD and continues until the LAST bit in the RAM is set.<br>11 = Reserved. |

BITS 2:25 not relevant in this reference discussion.

| | | | |
|---|---|---|---|
| 26-31 | MAD(0:5) | MACHINE ADDRESS. This field is the RAM address pointer of the executed command. | |

# INDEX