MOTOROLA

PowerPC™ MPC801

Integrated Microprocessor for Embedded Systems User's Manual

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Section 7
PowerPC Architecture Compliance

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Section 13
External Bus Interface

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Section 16
Serial Communication Modules

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Section 19
IEEE 1149.1 Test Access Port

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

SECTION 1
INTRODUCTION

The MPC801 PowerPC™ Quad Integrated Communications Controller (PowerQUICC) is a versatile one-chip integrated microprocessor and peripheral combination that can be used in a variety of controller applications. It is a low-cost version of the MPC860 that provides an effective price/performance solution across a wide range of applications. The MPC801, like the MPC860, combines a high-performance PowerPC core with a multifaceted system integration package. Unless otherwise specified, the PowerQUICC unit will be referred to as the MPC801 in this manual.

The MPC801 is a PowerPC-based derivative of Motorola's MC68360 Quad Integrated Communications Controller (QUICC™). The CPU on the MPC801 is a 32-bit PowerPC implementation that incorporates memory management units and instruction and data caches. The memory controller has been enhanced, thus enabling the MPC801 to support any type of memory, including high performance memories and newer dynamic random access memories (DRAMs).

The purpose of this manual is to describe the operation of all the MPC801 functionality with concentration on the I/O functions. Additional details on the MPC801 can be found in the PowerPC architectural specifications.

1.1 FEATURES

The following list summarizes the main features of the MPC801:

- PowerPC Single-Issue Integer Core Performs Branch Folding and Prediction with Conditional Prefetch, but Without Conditional Execution
- 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 @40MHz, 3.3V, 1.3W Total Power)
- Low Power (3.3V Operation with 5V TTL Compatibility)
- MPC8XX PowerPC System Interface, Including a Periodic Interrupt Timer, a Bus Monitor, and Real-Time Clocks
- Fully Static Design (0-40MHz Operation)

- Four Major Power-Saving Modes
  - Full on, doze, sleep, and deep sleep
  - Gear mode
- 256-Pin Ball Grid Array Packaging
- 26-Bit Address Bus and 32-bit Data 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 5V inputs and provides 3.3V outputs
- Flexible Memory Management
  - 8-entry, fully associative instruction translation lookaside buffers (TLBs)
  - 8-entry, fully associative data translation lookaside buffers
  - 4K, 16K, 512K, or 8M page size support
  - 1K protection granularity
  - Support for multiple protection groups and tasks
  - Attribute support for trapping, writethrough, cache inhibit, and memory-mapped I/O
  - Supports software tablewalk
- 1K Physical Address, Two-Way, Set-Associative Data Cache
  - Single-cycle access on hit
  - 4-word line size, burst fill, least recently used (LRU) replacement
  - Cache lockable on line granularity
  - Read capability of all tags and attributes provided for debugging purposes
- 2K 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 on line 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
- Memory Controller with Eight Banks
  - Glueless interface to SRAM, DRAM, EPROM, FLASH and other peripherals
  - Byte write enables and selectable parity generation
  - 32-bit address decodes with bit masks
  - Each bank can be a chip-select or $\overline{\text{RAS}}$ to support a DRAM bank
  - Maximum of 30 programmable wait states per bank
  - Programmable DRAM controller supports most size and speed memory interfaces
  - Four $\overline{\text{CAS}}$ lines, four $\overline{\text{WE}}$ lines, and one $\overline{\text{OE}}$ line
  - Boot chip-select available at reset (optional 8-, 16-, or 32-bit memory)
  - Variable block sizes (32K to 256M)
  - Selectable write protection

- System Interface Unit
  - Clock synthesizer
  - Power management
  - Reset controller
  - PowerPC decrementer and timebase
  - Real-time clock register
  - Periodic interrupt timer
  - Hardware bus monitor and software watchdog timer
  - IEEE 1149.1 JTAG test access port
  - Low-power stop mode
  - On-chip bus arbitration logic
- Interrupt Support
  - Eight external interrupt request lines
  - Internal interrupt sources
- UART Support
  - Two UARTs
  - Standard baud rates of 300bps to 115.2kbps with 16× sample clock
  - External 1× clock for high-speed synchronous communication
  - Flexible 5-wire serial interface
  - Direct support of IrDA physical layer protocol
  - 8-byte FIFOs for transmit and receive
  - Programmable data format:
  - Programmable channel modes (normal and local loopback)
  - Parity, framing, and overrun error detection
  - Generation and detection of break
  - Robust receiver data sampling with noise filtering
  - Eight maskable interrupts
  - Low-power idle mode
- I^2C® Support
  - Two-wire interface (SDA and SCL)
  - Full-duplex operation
  - Master or slave I^2C mode support
  - Multimaster environment support
  - Clock rate support at a maximum of 520KHz, assuming a 25MHz system clock
  - Local loopback capability for testing
- Serial Peripheral Interface Support
  - Four-wire interface (SPIMOSI, SPIMISO, SPICLK, and $\overline{\text{SPISEL}}$)
  - Full-duplex operation
  - 8- and 16-bit data character operation
  - Back-to-back character transmission and reception support
  - Master or slave serial peripheral interface mode support
  - Multimaster environment support

— Clock rates support at a maximum of 6.25MHz in master mode and 12.5MHz in slave mode (assuming a 25MHz system clock)
— Independent programmable baud rate generator
— Programmable clock phase and polarity
— Open-drain output pins support multimaster configuration
— Local loopback capability for testing

- Debug Interface Support
  — Eight comparators
  — Supports =, ≠, <, and > conditions
  — Each watchpoint can generate a breakpoint internally

1.2 MPC801 ARCHITECTURE

The MPC801 is a combination of the embedded PowerPC core and integrated peripherals brought together to meet the demands of various communications and networking products. The MPC801 is comprised of six modules that all use the 32-bit internal bus:

- An embedded PowerPC core
- A system interface unit
- Two UARTs
- An I^2C controller
- A serial peripheral interface

The MPC801 block diagram is illustrated in Figure 1-1.

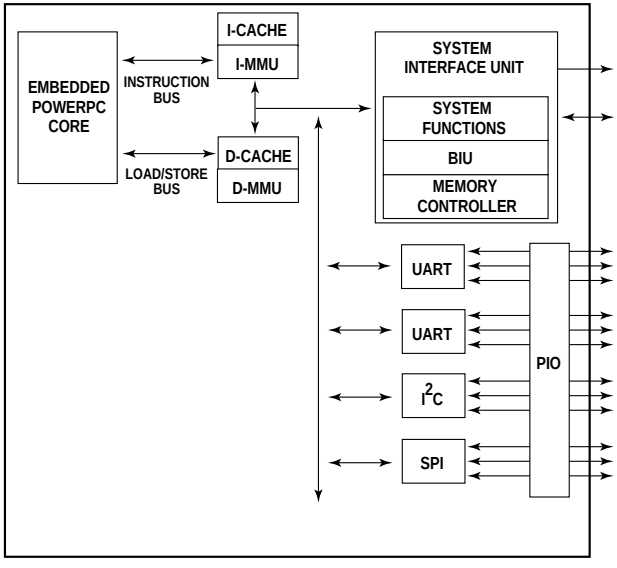


Figure 1-1. MPC801 Block Diagram

1.2.1 The Embedded PowerPC Core

The embedded PowerPC core complies with the specifications discussed in the *PowerPC Family: The Programming Environment (MPCFPE/D)* manual that is available from Motorola. The core has a fully static design that consists of two functional blocks—the integer block and load/store block. It executes all integer and load/store operations directly on the hardware. The core supports integer operations on a 32-bit internal data path with 32-bit arithmetic hardware. The interface to the internal and external buses is 32 bits.

The core uses a 2-instruction load/store queue, a 4-instruction prefetch queue, and a 6-instruction history buffer. It does branch folding and prediction with conditional prefetch, but does not support conditional execution. The core can operate on 32-bit external operands with one bus cycle. The PowerPC integer block supports 32×32-bit fixed-point general-purpose registers and it can execute one integer instruction per clock cycle.

The embedded PowerPC core is integrated with the memory management unit as well with the instruction and data caches. Each memory management unit (MMU) provides an 8-entry, fully associative instruction and data TLB, with 4K, 16K, 512K, and 8M page sizes. It supports 16 virtual address spaces with 16 protection groups. Three special registers are available as scratch registers to support software tablewalk and update.

The instruction cache is 2K, two way, set associative with physical addressing. It allows single-cycle access on hit with no added latency for miss. It has four words per line supporting burst line fill using least recently used replacement. The cache can be locked on a per line basis for application critical routines. The cache inhibit mode can be programmed on a per MMU page basis.

The data cache is 1K, two way, set associative with physical addressing. It allows single-cycle access on hit with one added clock latency for miss. It has four words per line, supporting burst line fill using LRU replacement. The cache can be locked on a per line basis for application critical data. The data cache can be programmed to support copyback or writethrough via the memory management unit. The cache inhibit mode can be programmed on a per MMU page basis. The PowerPC core, together with its instruction and data caches, delivers approximately 52 MIPS at 40MHz using Dhrystone 2.1.

The core contains a debug interface that provides superior debug capabilities without causing any degradation in the speed of operation. This interface supports six watchpoint pins that are used to detect software events. Internally, it has eight comparators, four of which operate on the effective address of the address bus. The remaining four comparators are split—two comparators operate on the effective address of the data address bus and two operate on the data bus. The core can compare using the =, $\neq$, <, and > conditions to generate watchpoints. Then each watchpoint can generate a breakpoint that can be programmed to trigger after a certain number of events.

1.2.2 The System Interface Unit

The system interface unit (SIU) on the MPC801 integrates general-purpose features useful in almost any 32-bit processor system, thus enhancing the performance provided by the system integration module on the MC68360 QUICC device. Multiple bus port sizes are supported and bus sizing allows 8-, 16-, and 32-bit peripherals and memory to exist in the 32-bit system bus mode. Data parity is supported using 4-bit data parity and the parity type can be odd or even. The system interface unit also contains power management functions, reset control, decrementer, timebase, and real-time clock.

The memory controller manages memories with a nonmultiplexed address bus using the SRAM interface. Using the DRAM interface, the memory controller also manages memories with a multiplexed address bus (DRAM, SRDRAM, and EDO). Both submodules support glueless interface to 8-, 16-, and 32-bit wide memories. The memory controller supports up to eight memory banks and can use address type matching to qualify the memory bank accesses. Each bank can use either the SRAM or DRAM interface. The memory controller provides four byte enable signals, one output enable signal, and one boot chip-select available at reset.

The SRAM interface provides block sizes that vary between 32K to 64M. Each bank of memory has 0 to 30 wait states (zero wait state means a two-clock access to external SRAM). The DRAM interface supports 256K, 512K, 1M, 2M, 4M, 8M, 16M, 32M, or 64M memory bank depths for all port sizes. The memory depth can be defined as 64K and 128K for 8-bit memory or 128M and 256M for 32-bit memory. The DRAM controller supports page mode access for successive transfers within bursts.

The MPC801 supports a glueless interface to one bank of DRAM, but external buffers are required for additional memory banks. The refresh unit provides $\overline{CAS}$ before $\overline{RAS}$, a programmable refresh timer, disable refresh mode, and stacking for seven refresh cycles. The DRAM interface uses a programmable state machine to support most memory interfaces.

1.2.3 The UART Controller

Each communication channel has a full-duplex universal asynchronous receiver/transmitter (UART). The operating frequency for each receiver and transmitter can be selected independently from the baud rate generator, counter/timer, or external clock. The transmitter accepts parallel data from the core, converts it to a serial bitstream, inserts the appropriate START, STOP, or optional PARITY bits, and then outputs a composite serial stream of data on the TxD output pin. The receiver accepts the serial data on the RxD pin, converts it to parallel format, checks for a START bit, STOP bit, PARITY bit (if any), or break condition, and transfers an assembled character to the core during read operations.

1.2.4 The I2C® Controller

The I^2C controller is a synchronous, multimaster bus that is used to connect several integrated circuits on a board. It uses two wires to carry information between the integrated circuits that are connected to the bus. The I^2C 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 that is derived from the I^2C controller baud rate generator in master mode and generated externally in slave mode.

1.2.5 The Serial Peripheral Interface Controller

The serial peripheral interface (SPI) is a full-duplex, synchronous, character-oriented channel that supports a four-wire interface (receive, transmit, clock and slave select). The serial peripheral interface 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 that is derived from the SPI baud rate generator in master mode and generated externally in slave mode. During an serial peripheral interface transfer, data is simultaneously transmitted and received.

1.3 POWER MANAGEMENT

The MPC801 supports a wide range of power management features, including full-on, doze, sleep, deep sleep, and low-power stop. In full-on mode, the MPC801 processor is fully powered with all internal units operating at full speed. A gear mode is provided that allows the operating system to reduce the operational frequency of the processor. Doze mode disables core functional units, except for the timebase, decrementer, phase locked loop, memory controller, and real-time clock. Sleep mode disables everything, except the real-time clock and periodic interrupt timer, thus leaving the phase locked loop active for quick wake-up. The deep sleep mode disables the phase locked loop for low-power, but at the expense of a slower wake-up. Low-power stop disables all logic in the processor (except the minimum logic required to restart the device) and lowers the power consumption, but it also requires the longest wake-up time.

1.4 MPC801 APPLICATIONS

The MPC801 device is specifically designed to be a general-purpose, low-cost entry point to the embedded PowerPC Family at Motorola. The device excels in applications that require the performance of a single-issue PowerPC core with moderate amounts of data and instruction cache. It can support alternate bus masters in addition to providing all the basic features of glueless memory connections, but does not provide much in the way of serial connectivity. Instead, it supplies simple UART serial channels as well as I^2C and serial peripheral interface channels for onboard communication to other peripheral chips.

The MPC801 excels in low-power and portable applications because of its expansive power-down modes. In addition, the normal operation current is low. The MPC801 is ideal for applications where a significant portion of your added value is in peripherals or ASICs and a low-cost general-purpose core is required. The programmable flexibility of the memory controller ensures that the board design can accommodate future memory types without hardware changes, thus enabling the ASIC to concentrate on other system goals.

1.5 DIFFERENCES BETWEEN THE MPC801 AND MPC860

The MPC801 can be considered a subset of the standard MPC860 device. The following modifications were made to the MPC860 to create the MPC801:

- The 4K instruction cache was reduced to 2K
- The 4K data cache was reduced to 1K
- The 32-bit instruction and data memory management units were reduced to eight TLB entries each
- The 32-bit address bus was reduced to 26 bits
- PCMCIA support was eliminated
- All existing serial interface logic and their respective DMAs was replaced with two UARTs, one I^2C, and a serial peripheral interface (all non-DMA based). The UARTs are modeled after the MC68328 DragonBall™ device.

1.6 MPC801 GLUELESS SYSTEM DESIGN

A fundamental design goal of the MPC801 was to have a flexible interface to other system components. Figure 1-2 illustrates a system configuration that offers one flash EPROM and supports DRAM SIMM and one SRAM. Depending on the capacitance on the system bus, external buffers may be required. From a logic standpoint, however, a glueless system is maintained.

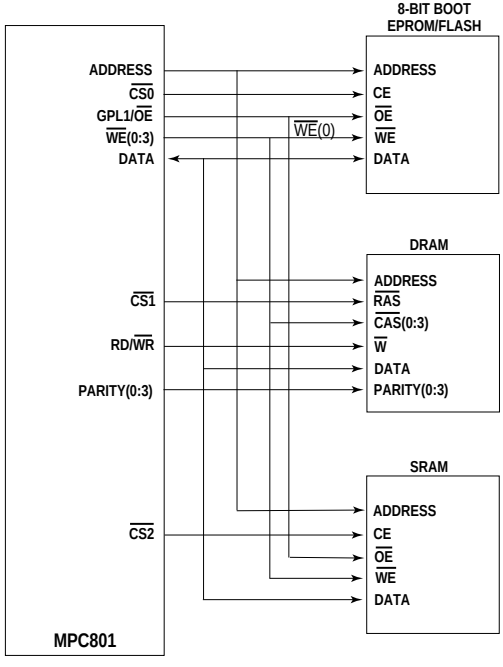


Figure 1-2. MPC801 System Configuration

SECTION 2
EXTERNAL SIGNALS

This section contains brief descriptions of the MPC801 input and output signals in their functional groups as illustrated in Figure 2-1.

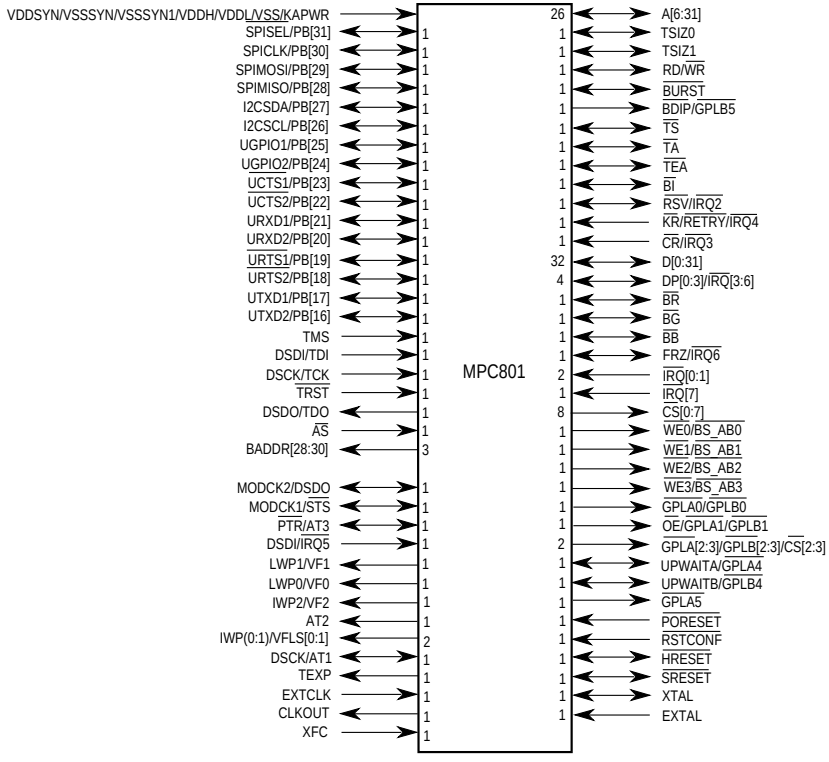


Figure 2-1. MPC801 External Signals

2.1 THE SYSTEM BUS SIGNALS

The MPC801 system bus signals consist of all the lines that interface with the external bus. Many of these lines perform different functions, depending on how you assign them. The following input and output signals are identified by their mnemonic name and each signal's pin number can be found in Figure 2-1.

Table 2-1. Signal Descriptions

PIN NAME	PIN NUMBER	DESCRIPTION
A[6-31]	See Table 2-2 for pin breakout.	**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 MPC801 initiates a transaction on the external bus. The MPC801 is connected to the 26 least-significant bits on the bus. The bus is input when an external master initiates a transaction on the bus and it is sampled internally so the memory controller can control the accessed slave device.
TSIZ0	C10	**Transfer Size 0**—When accessing a slave in the external bus, this three-state signal is used (together with TSIZ1) by the bus master to indicate the number of operand bytes waiting to be transferred in the current bus cycle. This signal is input when an external master initiates a transaction on the bus and it is sampled internally so the memory controller can control the accessed slave device.
TSIZ1	B10	**Transfer Size 1—**This three-state signal is used by the bus master to indicate the number of operand bytes waiting to be transferred in the current bus cycle. This signal is driven by the MPC801 when it is the owner of the bus. It is input when an external master initiates a transaction on the bus and it is sampled internally so the memory controller can control the accessed slave device.
RD/$\overline{WR}$	C4	**Read Write**—This three-state signal is driven by the bus master to indicate the direction of the bus' data transfer. A logic one indicates a read from a slave device and a logic zero indicates a write to a slave device. This signal is driven by the MPC801 when it is the owner of the bus. It is input when an external master initiates a transaction on the bus and is sampled internally so the memory controller can control the accessed slave device.
$\overline{BURST}$	E1	**Burst Transaction**—This three-state signal is driven by the bus master to indicate that the current initiated transfer is a burst one. This signal is driven by the MPC801 when it is the owner of the bus. It is input when an external master initiates a transaction on the bus; this signal and is sampled internally so the memory controller can control the accessed slave device.
$\overline{BDIP}$ $\overline{GPLB5}$	C3	**Burst Data in Progress**—When accessing a slave device in the external bus, the master on the bus asserts this signal to indicate that the data beat in front of the current one is the one requested by the master. This signal is negated prior to the expected last data beat of the burst transfer. **General-Purpose Line B5**—This signal is used by the memory controller when the user programmable machine B (UPMB) takes control of the slave access.
$\overline{TS}$	B1	**Transfer Start—**This three-state signal is asserted by the bus master to indicate the start of a bus cycle that transfers data to or from a slave device. This signal is driven by the master only when it has gained ownership of the bus. Every master should negate this signal before the bus relinquishes. A pull-up resistor should be connected to this signal to prevent a slave device from detecting a spurious bus accessing it when no master is taking ownership of the bus. This signal is sampled by the MPC801 when it is not the owner of the external bus so the memory controller can control the accessed slave device. It indicates that an external synchronous master initiated a transaction.

Table 2-1. Signal Descriptions (Continued)

PIN NAME	PIN NUMBER	DESCRIPTION
$\overline{TA}$	B2	**Transfer Acknowledge**—This bidirectional three-state signal indicates that the slave device addressed in the current transaction has accepted the data transferred by the master (write) or has driven the data bus with valid data (read). The signal behaves as an output when the memory controller takes control of the transaction. the only exception occurs when the memory controller is controlling the slave access by means of the gpcm and the corresponding option register is instructed to wait for an external assertion of the transfer acknowledge line. every slave device should negate the ta signal after the end of the transaction and immediately three-state it to avoid contentions on the line if a new transfer is initiated addressing other slave devices. A pull-up resistor should be connected to this signal to keep a master device from detecting the assertion of this signal when no slave is addressed in a transfer or when the address detection for the addressed slave is slow.
$\overline{TEA}$	A1	**Transfer Error Acknowledge**—This open-drain signal indicates that a bus error occurred in the current transaction. It is driven asserted by the MPC801 when the bus monitor does not detect a bus cycle termination within a reasonable amount of time. The assertion of $\overline{TEA}$ causes the termination of the current bus cycle, thus ignoring the state of $\overline{TA}$.
$\overline{BI}$	D3	**Burst Inhibit**—This bidirectional three-state signal indicates that the slave device addressed in the current burst transaction is unable to support burst transfers. The signal behaves as an output when the memory controller takes control of the transaction. When the MPC801 drives out the signal for a specific transaction, it asserts or negates $\overline{BI}$ 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.
$\overline{RSV}$ $\overline{IRQ2}$	G4	**Reservation**—This three-state signal is output by the MPC801 in conjunction with the address bus to indicate that the internal core initiated a transfer as a result of a **stwcx** or **lwarx** instruction. **Interrupt Request 2**—This input is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.
$\overline{KR}/\overline{RETRY}$ $\overline{IRQ4}$	I2	**Kill Reservation**—This input is used as a part of the storage reservation protocol when the MPC801 initiated a transaction as the result of a **stwcx** instruction. **Retry**—This input signal is used by the slave device to indicate that it is unable to accept the transaction. The MPC801 must relinquish ownership of the bus and initiate the transaction again after winning the bus arbitration. **Interrupt Request 4**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{IRQ4}$) and the DP1/$\overline{IRQ4}$ (if defined to function as $\overline{IRQ4}$).
$\overline{CR}$ $\overline{IRQ3}$	D1	**Cancel Reservation**—This input signal is used as a part of the storage reservation protocol. **Interrupt Request 3**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{IRQ3}$) and the DP0/$\overline{IRQ3}$ if defined to function as $\overline{IRQ3}$.
D[0:31]	See Table 2-2 for pin breakout	**Data Bus**—This bidirectional three-state bus provides the general-purpose data path between the MPC801 and all other devices. Although the data path is a maximum of 32 bits wide, it can be dynamically sized to support 8-, 16-, or 32-bit transfers. D0 is the most-significant bit of the data bus.
DP0 $\overline{IRQ3}$	M5	**Data Parity 0**—This bidirectional three-state signal provides parity generation and checking for the data bus lane D[0:7] by transferring to a slave device initiated by the MPC801. The parity function can be defined independently for each one of the addressed memory banks (if controlled by the memory controller) and for the rest of the slaves sitting on the external bus. **Interrupt Request 3**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{IRQ3}$) and the $\overline{CR}$/$\overline{IRQ3}$ if defined to function as $\overline{IRQ3}$.

Table 2-1. Signal Descriptions (Continued)

PIN NAME	PIN NUMBER	DESCRIPTION
DP1 $\overline{\text{IRQ4}}$	O5	**Data Parity 1**—This bidirectional three-state signal provides parity generation and checking for the data bus lane D[8:15] by transferring to a slave device initiated by the MPC801. The parity function can be defined independently for each one of the addressed memory banks (if controlled by the memory controller) and for the rest of the slaves on the external bus. **Interrupt Request 4**—This input is one of the eight external lines that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{\text{IRQ4}}$) and the $\overline{\text{KR/RETRY/IRQ4}}$ if defined to function as $\overline{\text{IRQ4}}$.
DP2 $\overline{\text{IRQ5}}$	O4	**Data Parity 2**—This bidirectional three-state signal provides parity generation and checking for the data bus lane D[16:23] by transferring to a slave device initiated by the MPC801. The parity function can be defined independently for each one of the addressed memory banks (if controlled by the memory controller) and for the rest of the slaves on the external bus. **Interrupt Request 5**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{\text{IRQ6}}$) and the DSDI/$\overline{\text{IRQ5}}$ if defined to function as $\overline{\text{IRQ5}}$.
DP3 $\overline{\text{IRQ6}}$	N4	**Data Parity 3**—This bidirectional three-state signal provides parity generation and checking for the data bus lane D[24:31] by transferring to a slave device initiated by the MPC801. The parity function can be defined independently for each one of the addressed memory banks (if controlled by the memory controller) and for the rest of the slaves on the external bus. **Interrupt Request 6**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{\text{IRQ6}}$) and the FRZ/$\overline{\text{IRQ6}}$ if defined to function as $\overline{\text{IRQ6}}$.
$\overline{\text{BR}}$	E3	**Bus Request**—This bidirectional signal is asserted low when a possible master is requesting ownership of the bus. When the MPC801 is configured to operate with the internal arbiter, this signal is configured as an input. However, when the MPC801 is configured to operate with an external arbiter, this signal is configured as an output and asserted every time a new transaction is intended to be initiated and no parking on the bus is granted.
$\overline{\text{BG}}$	D2	**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 MPC801 is configured to operate with the internal arbiter, this signal is configured as an output and asserted every time the external master asserts the $\overline{\text{BR}}$ signal and its priority request is higher than any of the internal sources requiring the initiation of a bus transfer. However, when the MPC801 is configured to operate with an external arbiter, this signal is configured as an input.
$\overline{\text{BB}}$	C2	**Bus Busy**—This bidirectional signal is asserted low by a master to show that it owns the bus. The MPC801 asserts this signal after the bus arbiter grants it bus ownership and the $\overline{\text{BB}}$ signal is negated.
FRZ $\overline{\text{IRQ6}}$	E2	**Freeze**—This output signal is asserted to indicate that the internal core is in debug mode. **Interrupt Request 6**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{\text{IRQ6}}$) and the DP3/$\overline{\text{IRQ6}}$ (if defined to function as $\overline{\text{IRQ6}}$.)
IRQ0	N15	**Interrupt Request 0**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.
$\overline{\text{IRQ1}}$	N16	**Interrupt Request 1**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.
$\overline{\text{IRQ7}}$	M17	**Interrupt Request 7**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.

Table 2-1. Signal Descriptions (Continued)

PIN NAME	PIN NUMBER	DESCRIPTION
$\overline{CS}$[0:7]	See Table 2-2 for pin breakout	**Chip Select** 0–7—These output signals enable peripheral or memory devices at programmed addresses if they are appropriately defined in the memory controller. $\overline{CS0}$ can be configured to be the global chip-select for the boot device.
$\overline{WE0}$ $\overline{BS\_AB0}$	C9	**Write Enable 0**—This output signal is asserted when a write access to an external slave controlled by the GPCM in the memory controller is initiated by the MPC801. $\overline{WE0}$ is asserted if the data lane D[0:7] contains valid data to be stored by the slave device. **Byte Select 0 on UPMA or UPMB**—This output signal is asserted as required by the UPMA or UPMB in the memory controller whenever you program it. In a read or write transfer, the signal is only asserted if the data lane D[0:7] contains valid data.
$\overline{WE1}$ $\overline{BS\_AB1}$	A10	**Write Enable 1**—This output signal is asserted when the MPC801 initiates a write access to an external slave controlled by the GPCM in the memory controller. $\overline{WE1}$ is asserted if the data lane D[8:15] contains valid data to be stored by the slave device. **Byte Select 1 on UPMA or UPMB**—This output signal is asserted as required by the UPMA or UPMB in the memory controller whenever you program it. In a read or write transfer, the signal is only asserted if the data lane D[8:15] contains valid data.
$\overline{WE2}$ $\overline{BS\_AB2}$	A9	**Write Enable 2**—This output signal is asserted when the MPC801 initiates a write access to an external slave controlled by the GPCM in the memory controller. $\overline{WE2}$ is asserted if the data lane D[16:23] contains valid data to be stored by the slave device. **Byte Select 2 on UPMA or UPMB**—This output signal is asserted as required by the UPMA or UPMB in the memory controller whenever you program it. In a read or write transfer, the signal is only asserted if the data lane D[16:23] contains valid data.
$\overline{WE3}$ $\overline{BS\_B3}$	C8	**Write Enable 3**—This output signal is asserted when the MPC801 initiates a write access to an external slave controlled by the GPCM in the memory controller. $\overline{WE3}$ is asserted if the data lane D[24:31] contains valid data to be stored by the slave device. **Byte Select 3 on UPMA or UPMB**—This output signal is asserted as required by the UPMA or UPMB in the memory controller whenever you program it. In a read or write transfer, the signal is only asserted if the data lane D[24:31] contains valid data.
$\overline{GPLA0}$ $\overline{GPLB0}$	B8	**General-Purpose Line 0 on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). **General-Purpose Line 0 on UPMB**—This output signal reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB).
$\overline{OE}$ $\overline{GPLA1}$ $\overline{GPLB1}$	A7	**Output Enable**—This output signal is asserted when the MPC801 initiates a read access to an external slave controlled by the GPCM in the memory controller. **General-Purpose Line 1on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). **General-Purpose Line 1 on UPMB**—This output signal reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB).
$\overline{GPLA}$[2:3] $\overline{GPLB}$[2:3] $\overline{CS}$[2:3]	B7 and C7	**General-Purpose Lines 2 and 3 on UPMA**—These output signals 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). **General-Purpose Lines 2 and 3 on UPMB**—These output signals 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). **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 $\overline{CS2}$ and $\overline{CS3}$ is independently defined for each signal in the SIUMCR.

Table 2-1. Signal Descriptions (Continued)

PIN NAME	PIN NUMBER	DESCRIPTION
UPWAITA GPLA4	A2	**User Programmable Machine Wait A**—This input signal is sampled as you need it when an access to an external slave is controlled by the UPMA in the memory controller. **General-Purpose Line 4 on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA).
UPWAITB GPLB4	A3	**User Programmable Machine Wait B**—This input signal is sampled as you need it when an access to an external slave is controlled by the UPMB in the memory controller. **General-Purpose Line 4 on UPMB**—This output signal reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB).
GPLA5	B3	**General-Purpose Line 5 on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). This signal can also be controlled by the UPMB.
PORESET	M3	**Power -On Reset**—When asserted, this input signal causes the MPC801 to enter the power-on reset state.
RSTCONF	N2	**Reset Configuration**—This input signal is sampled by the MPC801 during the assertion of the HRESET signal. If it is asserted, the configuration mode is sampled in the form of the hard reset configuration word driven on the data bus. When this signal is negated, the default configuration mode is adopted by the MPC801. Notice that the initial base address of internal registers is determined in this sequence.
HRESET	M2	**Hard Reset**—This open drain line, when asserted, causes the MPC801 external crystal to enter the hard reset state.
SRESET	L3	**Soft Reset**—This open drain line, when asserted, causes the MPC801 external crystal to enter the soft reset state.
XTAL	N1	This output signal is one of the connections to an external crystal for the internal oscillator circuitry.
EXTAL	M1	This signal is one of the connections to an external crystal for the internal oscillator circuitry.
XFC	O3	**External Filter Capacitance**—This input signal is the connection pin to an external capacitor filter for the PLL circuitry.
CLKOUT	P5	**Clock Out**—This output signal is the clock system frequency.
EXTCLK	L1	**External Clock**—This input signal is the external input clock from an external source.
TEXP	L2	**Timer Expired**—This output signal reflects the status of the TEXPS bit in the PLPRCR in the CLOCK interface.
DSCK/AT1	H4	**Development Serial Clock**—This input signal is the clock for the debug port interface. **Address Type 1**—This bidirectional three-state signal is driven by the MPC801 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.
IWP[0:1] VFLS[0:1]	G1 and G2	**Instruction Watchpoint 0-1**—These output signals report the detection of an instruction watchpoint in the program flow executed by the internal core. **Visible History Buffer Flushes Status**—These output signals are output by the MPC801 when you need program instructions flow tracking. They report the number of instructions flushed from the history buffer in the internal core.
AT2	H2	**Address Type 2**—This bidirectional three-state signal is driven by the MPC801 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.

Table 2-1. Signal Descriptions (Continued)

PIN NAME	PIN NUMBER	DESCRIPTION
IWP2 VF2	F1	**Instruction Watchpoint 2**—This output signal reports the detection of an instruction watchpoint in the program flow executed by the internal core. **Visible Instruction Queue Flush Status**—This output signal, together with VF0 and VF1, is output by the MPC801 when you need program instructions flow tracking. VF*x* reports the number of instructions flushed from the instruction queue in the internal core.
LWP0 VF0	F2	**Load/Store Watchpoint 0**—This output signal reports the detection of a data watchpoint in the program flow executed by the internal core. **Visible Instruction Queue Flushes Status**—This output signal, combined with VF1 and VF2, is output by the MPC801 when you need program instructions flow tracking. VF reports the number of instructions flushed from the instruction queue in the internal core.
LWP1 VF1	G3	**Load/Store Watchpoint 1**—This output signal reports the detection of a data watchpoint in the program flow executed by the internal core. **Visible Instruction Queue Flushes Status**—This output signal, combined with VF0 and VF2, is output by the MPC801 when you need program instructions flow tracking. VF reports the number of instructions flushed from the instruction queue in the internal core.
DSDI $\overline{\text{IRQ5}}$	I3	**Development Serial Data Input**—This input signal is the data in for the debug port interface. **Interrupt Request 5**—This input signal is one of the eight external signals that can request through the internal interrupt controller, a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{\text{IRQ5}}$) and the DP2/$\overline{\text{IRQ5}}$ (if defined to function as $\overline{\text{IRQ5}}$).
$\overline{\text{PTR}}$ AT3	H3	**Program Trace**—This output signal is asserted by the MPC801 to indicate an instruction fetch is taking place to allow program flow tracking. **Address Type 3**—This bidirectional three-state signal is driven by the MPC801 when it initiates a transaction on the external bus. When the transaction is initiated by the internal core, it indicates if the transfer is reserved for data transfers or a program trace indication for instructions fetch.
MODCK1 $\overline{\text{STS}}$	J3	**Mode Clock 1**—This input signal is sampled at $\overline{\text{PORESET}}$ negation to configure the PLL/clock mode of operation. **Special Transfer Start**—This output signal is driven by the MPC801 to indicate the start of a transaction on the external bus or to signal the beginning of an internal transaction in show cycle mode.
MODCK2 DSDO	J2	**Mode Clock 2**—This input signal is sampled at $\overline{\text{PORESET}}$ negation to configure the PLL/clock mode of operation. **Development Serial Data Output**—This output signal is the data out of the debug port interface.
BADDR[28:30]	J1, I1, and H1	**Burst Address**—These output signals duplicate the value of A[28:29] when: • An internal master in the MPC801 initiates a transaction on the external bus. • An asynchronous external master initiates a transaction. • A synchronous external master initiates a single beat transaction. These signals are 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.
$\overline{\text{AS}}$	I4	**Address Strobe**—This input signal is driven by an external asynchronous master to indicate a valid address on the A[6:31] signals. The memory controller in the MPC801 synchronizes this signal and controls the memory device addressed under its control.
PB[31] $\overline{\text{SPISEL}}$	F16	**General-Purpose I/O Port B Bit 31**—This is Bit 31 of the general-purpose I/O port B. **$\overline{\text{SPISEL}}$**—This is the serial peripheral interface slave select input pin.
PB[30] SPICLK	G13	**General-Purpose I/O Port B Bit 30**—This is Bit 30 of the general-purpose I/O port B. **SPICLK**—This is the serial peripheral interface output clock when it is configured as a master or serial peripheral interface input clock when it is configured as a slave.

Table 2-1. Signal Descriptions (Continued)

PIN NAME	PIN NUMBER	DESCRIPTION
PB[29] SPIMOSI	G15	**General-Purpose I/O Port B Bit 29**—This is Bit 29 of the general-purpose I/O port B. **SPIMOSI**—This is the serial peripheral interface output data when it is configured as a master or serial peripheral interface input data when it is configured as a slave.
PB[28] SPIMISO	G16	**General-Purpose I/O Port B Bit 28**—This is Bit 28 of the general-purpose I/O port B. **SPIMISO**—This is the serial peripheral interface input data when it is configured as a master or serial peripheral interface output data when it is configured as a slave.
PB[27] I2CSDA	H14	**General-Purpose I/O Port B Bit 27**—This is Bit 27 of the general-purpose I/O port B. **I2CSDA**—This is the I^2C serial data pin. It is bidirectional and should be configured as an open-drain output.
PB[26] I2CSCL	H15	**General-Purpose I/O Port B Bit 26**—This is Bit 26 of the general-purpose I/O port B. **I2CSCL**—This is the I^2C serial clock pin. It is bidirectional and should be configured as an open-drain output.
PB[25] UGPIO1	J13	**General-Purpose I/O Port B Bit 25**—This is Bit 25 of the general-purpose I/O port B. **UGPIO1**—This is the general-purpose input/output pin of UART1. It can be configured as a general input or output or it can serve as the source of the clock to the baud rate generator. It can also output the bit clock at the selected baud rate.
PB[24] UGPIO2	J15	**General-Purpose I/O Port B Bit 24**—This is Bit 24 of the general-purpose I/O port B. **UGPIO2**—This is the general-purpose input/output pin of UART2. It can be configured as a general input or output or it can serve as the source of the clock to the baud rate generator. It can also output the bit clock at the selected baud rate.
PB[23] $\overline{\text{UCTS1}}$	K16	**General-Purpose I/O Port B Bit 23**—This is Bit 23 of the general-purpose I/O port B. **$\overline{\text{UCTS1}}$**—This is an active low clear-to-send input of UART1 that is used to control the transmitter.
PB[22] $\overline{\text{UCTS2}}$	K15	**General-Purpose I/O Port B Bit 22—**This is Bit 22 of the general-purpose I/O port B. **$\overline{\text{UCTS2}}$**—This is an active low clear-to-send input of UART2 that is used to control the transmitter.
PB[21] URXD1	K14	**General-Purpose I/O Port B Bit 21**—This is Bit 21 of the general-purpose I/O port B. **URXD1**—This signal is the receive data serial input of UART1.
PB[20] URXD2	L16	**General-Purpose I/O Port B Bit 20**—This is Bit 20 of the general-purpose I/O port B. **URXD2**—This signal is the receive data serial input of UART2.
PB[19] $\overline{\text{URTS1}}$	L15	**General-Purpose I/O Port B Bit 19**—This is Bit 19 of the general-purpose I/O port B. **$\overline{\text{URTS1}}$**—This is an active low ready-to-send output from UART1 used to indicate that the receiver is ready.
PB[18] $\overline{\text{URTS2}}$	L14	**General-Purpose I/O Port B Bit 18**—This is Bit 18 of the general-purpose I/O port B. **$\overline{\text{URTS2}}$**—This is an active low ready-to-send output from UART2 used to indicate that the receiver is ready.
PB[17] UTXD1	M16	**General-Purpose I/O Port B Bit 17**—This is Bit 17 of the general-purpose I/O port B. **UTXD1**—This signal is the transmit data serial output from UART1.
PB[16] UTXD2	M15	**General-Purpose I/O Port B Bit 16**—This is Bit 16 of the general-purpose I/O port B. **UTXD2**—This signal is the transmit data serial output from UART1.

Table 2-1. Signal Descriptions (Continued)

PIN NAME	PIN NUMBER	DESCRIPTION
Power Supply	A8, J16, K1, and P9 P3 P2 P1 O1	**VDDH**—This signal is the power supply of the I/O buffers and certain parts of the clock control. **VDDSYN**—This signal is the power supply of the PLL circuitry. **VSSSYN—This signal is the power supply for the clock synthesizer.** **VSSSYN1—This signal is the power supply for the clock synthesizer.** **KAPWR**—This signal is the power supply of the internal oscillator, real-time clock, periodic interrupt timer, decrementer, and timebase.
TCK DSCK	I15	**Test Clock**—This input signal is the clock of the JTAG interface. **Development Serial Clock**—This input signal is the clock for the debug port interface.
TMS	H13	**Test Mode Select**—This input signal controls the scan chain test mode operations. It should be powered through a pull-up resistor if unused.
TDI DSDI	I16	**Test Data Input**—This input signal is the data in for the JTAG interface. **Development Serial Data Input**—This input signal is the data for the debug port interface.
TDO DSDO	H16	**Test Data Output**—This three-state output signal is the data out of the JTAG interface. **Development Serial Data Output**—This output signal is the data out of the debug port interface.
$\overline{\text{TRST}}$	I17	**Test Reset**—This input signal is the asynchronous reset of the TAP machine on the JTAG interface.

Table 2-2. Pin Breakout

	PIN NAME	PIN NUMBER
ADDRESS BUS PINS 6–31	A6	G17
	A7	E16
	A8	F15
	A9	D16
	A10	G15
	A11	F17
	A12	C16
	A13	D15
	A14	E17
	A15	B15
	A16	C15
	A17	C14
	A18	B12
	A19	A15
	A20	B17
	A21	C13
	A22	C11
	A23	B13
	A24	A14
	A25	C12
	A26	B11
	A27	A16
	A28	A12
	A29	B16
	A30	A13
	A31	A11

Table 2-2. Pin Breakout (Continued)

	PIN NAME	PIN NUMBER
DATA BUS PINS 0–31	D0	O16
	D1	P17
	D2	P12
	D3	P11
	D4	P16
	D5	P10
	D6	P8
	D7	P6
	D8	N14
	D9	O13
	D10	N11
	D11	P13
	D12	P12
	D13	O15
	D14	M10
	D15	N10
	D16	M9
	D17	O14
	D18	O10
	D19	N9
	D20	O9
	D21	N8
	D22	O8
	D23	N12
	D24	O7
	D25	N7
	D26	M7
	D27	P15
	D28	N6
	D29	O6
	D30	N5
	D31	M6

Table 2-2. Pin Breakout (Continued)

	PIN NAME	PIN NUMBER
CHIP SELECT PINS 0–7	$\overline{CS0}$	B4
	$\overline{CS1}$	A4
	$\overline{CS2}$	C5
	$\overline{CS3}$	B5
	$\overline{CS4}$	A6
	$\overline{CS5}$	B6
	$\overline{CS6}$	C6
	$\overline{CS7}$	A5

SECTION 3
MEMORY MAP

The MPC801 internal memory resources are mapped within a contiguous block of 16K storage. The location of this block within the global 4G real storage space can be mapped on 64K resolution through an implementation specific special register called the internal memory map register. Refer to **Section 12 System Interface Unit** and **Appendix A Quick Reference Guide to MPC801 Registers** for more information. The following table defines the internal memory map of the MPC801.

Table 3-1. MPC801 Internal Memory Map

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
GENERAL SYSTEM INTERFACE UNIT			
000	SIUMCR	SIU Module Configuration Register	32
004	SYPCR	System Protection Control Register	32
008	RES	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	RES	Reserved	—
UART1 CONTROLLER			
040	CNTREG1	Control Register	16
044	BAUDREG1	Baud Control Register	16
048	GLOBREG1	Global Register	16
04C	RXREG1	Receiver Register	16
050	TXREG1	Transmitter Register	16
054	RXREG1	Receiver Register (Special Read Mode)	16
058 to 05F	RES	Reserved	—

Table 3-1. MPC801 Internal Memory Map (Continued)

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
UART2 CONTROLLER			
060	CNTREG2	Control Register	16
064	BAUDREG2	Baud Control Register	16
068	GLOBREG2	Global Register	16
06C	RXREG2	Receiver Register	16
070	TXREG2	Transmitter Register	16
074	RXREG2	Receiver Register (Special Read Mode)	16
078 to 0FF	RES	Reserved	—
MEMORY CONTROLLER			
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
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	RES	Reserved	—
164	MAR	Memory Address Register	32
168	MCR	Memory Command Register	32
16C to 16F	RES	Reserved	—
170	MAMR	Machine A Mode Register	32
174	MBMR	Machine B Mode Register	32
178	MSTAT	Memory Status Register	16

Table 3-1. MPC801 Internal Memory Map (Continued)

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
17A	MPTPR	Memory Periodic Timer Prescaler	16
17C	MDR	Memory Data Register	32
I^2C CONTROLLER			
180	I2CER	I^2C Event Register	8
184	I2CMR	I^2C Mask Register	8
188	I2CRD	I^2C Receive Data Register	16
18C	I2CTD	I^2C Register	16
190	I2MOD	I^2C Mode Register	8
194	I2ADD	I^2C Address Register	8
198	I2BRG	I^2C BRG Register	8
19C	I2COM	I^2C Command Register	8
1A0 to 1AF	RES	Reserved	—
SERIAL CONTROLLER			
1B0	SECCOM	Serial Controller Command Register	8
1B4 to 1BF	RES	Reserved	—
SERIAL PERIPHERAL INTERFACE			
1C0	SPIER	SPI Event Register	8
1C4	SPIMR	SPI Mask Register	16
1C8	SPIRD	SPI Receive Data Register	32
1CC	SPITD	SPI Transmit Data Register	32
1D0	SPMODE	SPI Mode Register	16
1D4	SPCOM	SPI Command Register	8
1D8 to 1DF	RES	Reserved	—
PORT B			
1E0	PBODR	Port B Open-Drain Register	16
1E4	PBDAT	Port B Data Register	16
1E8	PBDIR	Port B Data Direction Register	16
1EC	PBPAR	Port B Pin Assignment Register	16
AF0 to 1FF	RES	Reserved	—

Table 3-1. MPC801 Internal Memory Map (Continued)

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
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	RES	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	RES	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	RES	Reserved	—
CLOCKS AND RESET			
280	SCCR	System Clock Control Register	32
284	PLPRCR	PLL, Low-Power and Reset Control Register	32
288	RSR	Reset Status Register	32
28C to 2FF	RES	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	RES	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

Table 3-1. MPC801 Internal Memory Map (Continued)

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
330 to 33F	RES	Reserved	—
340	PISCRK	Periodic Interrupt Status and Control Register Key	32
344	PITCK	Periodic Interrupt Count Register Key	32
348 to 37F	RES	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	RES	Reserved	—

SECTION 4
RESET

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. Possible Reset Results

RESET SOURCE	RESET EFFECT						
	RESET LOGIC AND PLL STATES RESET	SYSTEM CONFIG RESET	CLOCK MODULE RESET	HRESET PIN DRIVEN	DEBUG PORT CONFIG	OTHER INTERNAL LOGIC RESET	SRESET PIN DRIVEN
Power-On Reset	Yes	Yes	Yes	Yes	Yes	Yes	Yes
External Hard Reset Loss-of-Lock Software Watchdog Check Stop Debug Port Hard Reset JTAG Reset	No	Yes	Yes	Yes	Yes	Yes	Yes
External Soft Reset Debug Port Soft Reset	No	No	No	No	Yes	Yes	Yes

4.1 TYPES OF RESET

The MPC801 has several types of inputs to the reset logic:

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

- External soft reset
- Internal soft reset
  - — Debug port soft reset

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

4.1.1 Power-On Reset

Power-on reset is an active low input pin called $\overline{\text{PORESET}}$. In a system with power-down low-power mode, this pin should only be activated when a voltage in the keep alive power (KAPWR) rail fails. When this pin is asserted, the MODCK bits are sampled and the phase-locked loop multiplication factor and pitrtclk and tmbclk sources are changed to their default values. When this pin is negated, internal MODCK values are unchanged. The $\overline{\text{PORESET}}$ pin should be asserted for a minimum of 3 microseconds. After detecting this assertion, the MPC801 enters the power-on reset state and stays there until the following events occur:

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

When $\overline{\text{PORESET}}$ is asserted, the MPC801 enters the power-on reset (POR) state in which $\overline{\text{SRESET}}$ and $\overline{\text{HRESET}}$ are asserted by the core. When the MPC801 remains in POR, the extension counter of 512 is reset,and the MODCK pins are sampled when POR pin is negated. After the negation of $\overline{\text{PORESET}}$ or PLL lock, the core enters the state of internal initiated $\overline{\text{HRESET}}$ and continues driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins for 512 cycles. When the timer expires, which is usually after the 512 cycles, the configuration is sampled from the data pins and the core stops driving the pins. An external pull-up resistor should drive the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins high. After the pins are negated, a 16-cycle period passes before the presence of an external (hard/soft) reset is tested. Refer to **Section 4.3.1 Hard Reset** for more information.

4.1.2 External $\overline{\text{Hard Reset}}$

$\overline{\text{HRESET}}$ (hard reset) is a bidirectional, active low I/O pin. The MPC801 can only detect an external assertion of $\overline{\text{HRESET}}$ if it occurs while the MPC801 is not asserting reset. During $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$ is asserted. $\overline{\text{HRESET}}$ is an open-collector type of pin. $\overline{\text{SRESET}}$ (soft reset) is a bidirectional, active low I/O pin. The MPC801 can only detect an external assertion of $\overline{\text{SRESET}}$ if it occurs while the MPC801 is not asserting reset. The $\overline{\text{SRESET}}$ is also an open-collector type of pin.

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

4.1.3 Internal Hard Reset

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

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

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

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

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

4.1.3.4 DEBUG PORT HARD RESET. When the development port receives a hard reset request from the development tool, an internal hard reset sequence is generated. In this case, the development tool must reconfigure the debug port. Refer to **Section 18.3.3.1.2 Development Serial Data In** for more information.

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

4.1.4 External Soft Reset

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

4.1.5 Internal Soft Reset

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

4.1.5.1 DEBUG PORT SOFT RESET. When the development port receives a soft reset request from the development tool, an internal soft reset sequence is generated. In this case the development tool must reconfigure the debug port. Refer to **Section 18.3.3.1.2 Development Serial Data In** for more information.

4.2 RESET STATUS REGISTER

The 32-bit reset status register (RSR) is powered by the keep alive power supply. It is memory-mapped into the MPC801 system interface unit register map and receives its default reset values at power-on reset.

RSR

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

EHRS—External Hard Reset Status

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

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

ESRS—External Soft Reset Status

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

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

LLRS—Loss-of-Lock Reset Status

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

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

SWRS—Software Watchdog Reset Status

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

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

CSRS—Check Stop Reset Status

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

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

DBHRS—Debug Port Hard Reset Status

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

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

DBSRS—Debug Port Soft Reset Status

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

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

JTRS—JTAG Reset Status

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

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

Bits 8–31—Reserved

These bits are reserved and should be set to 0.

4.3 HOW TO CONFIGURE RESET

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

4.3.1 Hard Reset

When a hard reset event occurs, the MPC801 reconfigures its hardware system as well as the development port configuration. The logical value of the bits that determine its initial mode of operation are sampled either from the data bus or from an internal default constant (D[0:31]=x'00000000). If, at sampling time, $\overline{\text{RSTCONF}}$ is asserted, the configuration is sampled from the data bus. Otherwise, it is sampled from the internal default. While $\overline{\text{HRESET}}$ and $\overline{\text{RSTCONF}}$ are asserted, the MPC801 pulls the data bus low through a weak resistor. You can overwrite this default by driving high to the appropriate bit (see Figure 4-1). The hardware reset configuration scheme for $\overline{\text{PORESET}}$ assertion is shown in Figures 4-2 through 4-4. While the $\overline{\text{PORESET}}$ input signal is being asserted, the core assumes the default reset configuration that changes when $\overline{\text{PORESET}}$ is negated or the CLKOUT signal starts oscillating. In this last case, the hardware configuration is sampled every nine clock cycles on the rising edge of the CLKOUT. The setup time required for the data bus is 15 cycles and the maximum rise time of $\overline{\text{HRESET}}$ should be less than six clock cycles. Refer to **Section 4.3.2 Soft Reset** for more information.

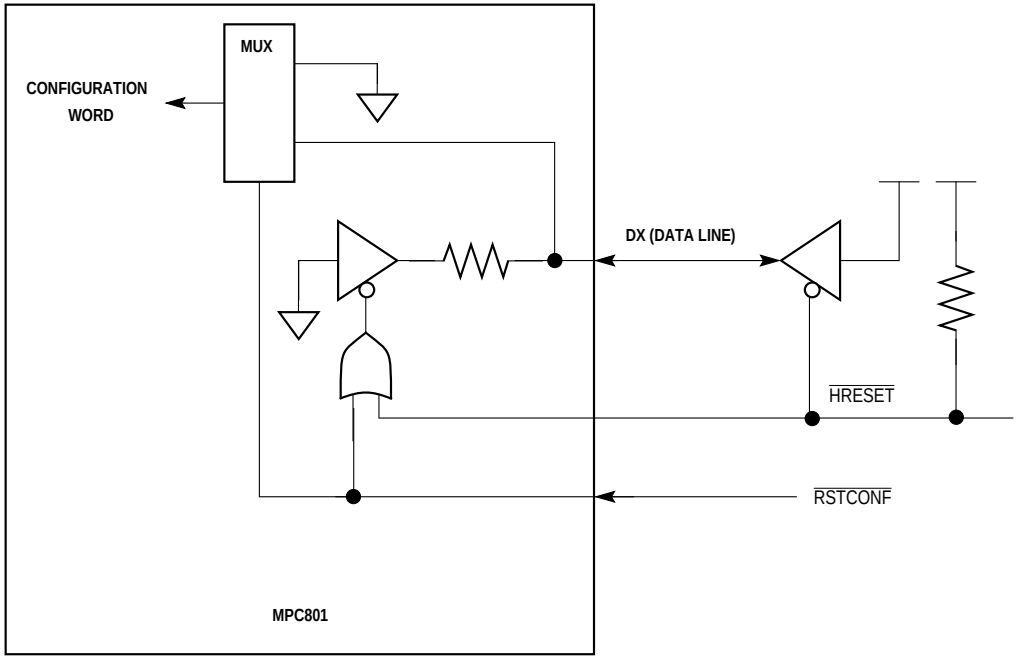


Figure 4-1. Reset Configuration Basic Scheme

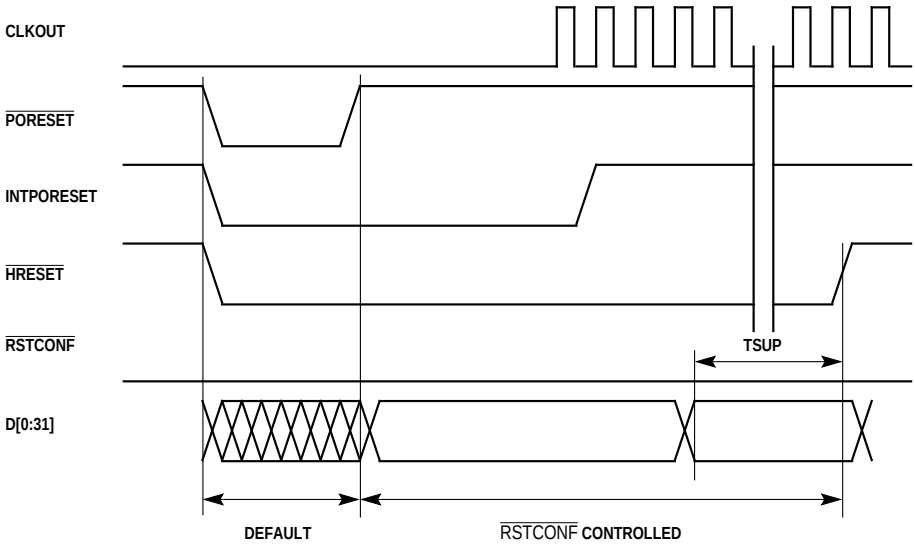


Figure 4-2. Reset Configuration Sampling Scheme For Short PORESET Assertion

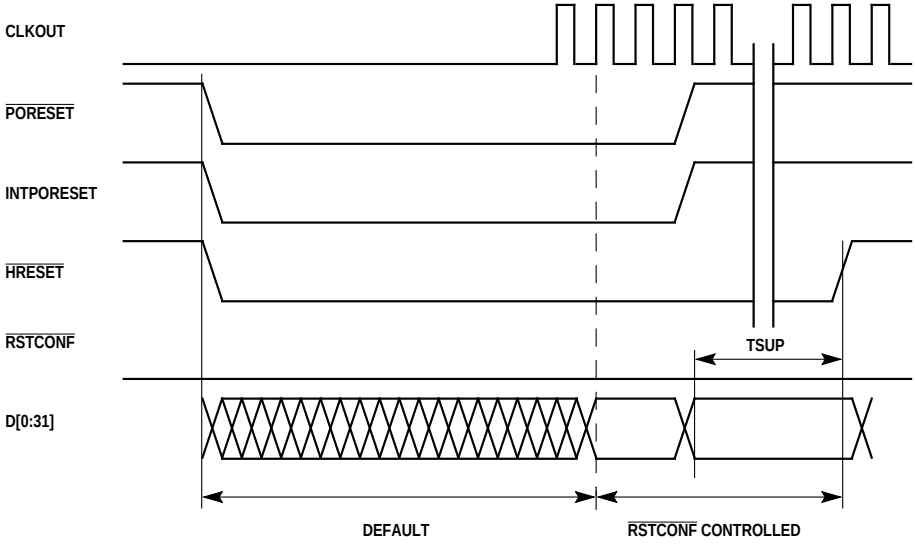


Figure 4-3. Reset Configuration Sampling Scheme For Long PORESET Assertion

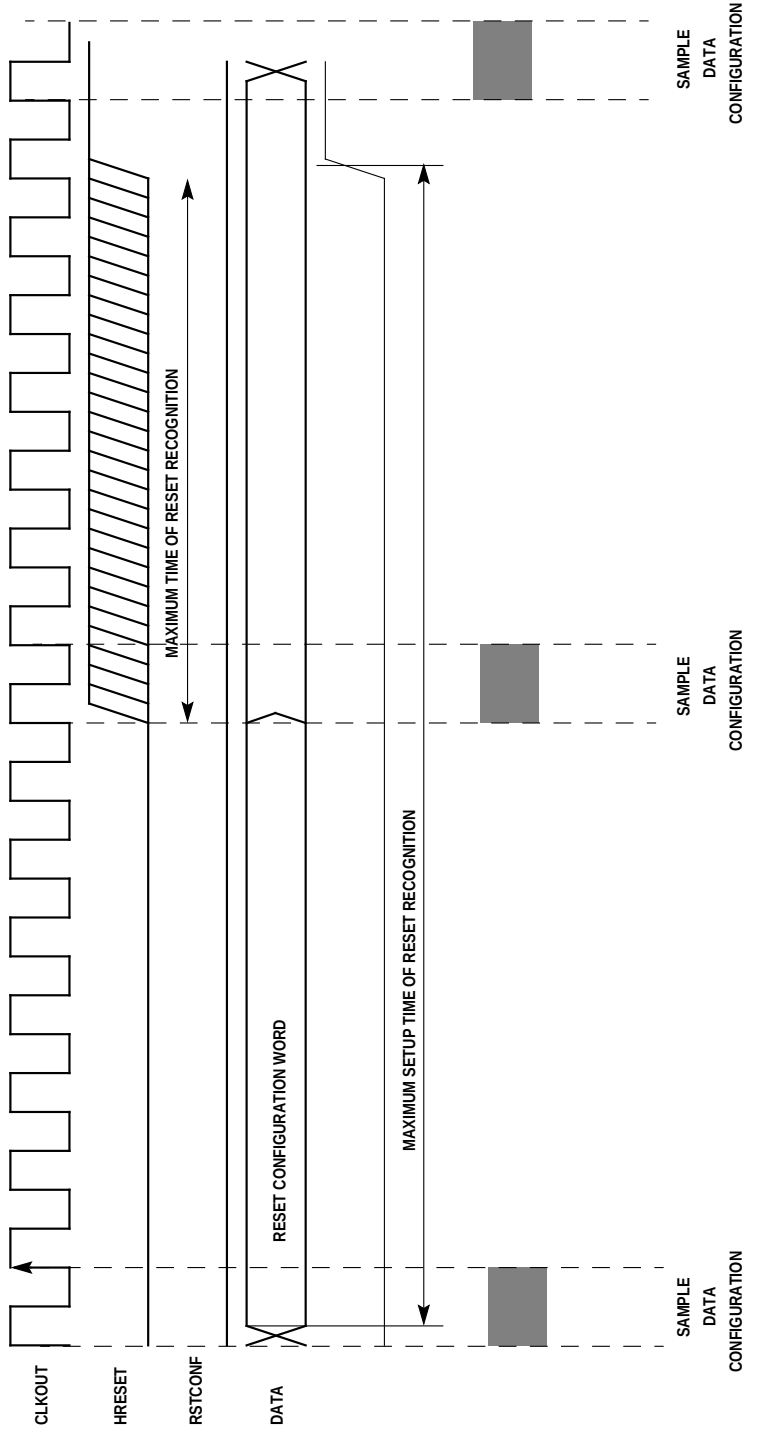


Figure 4-4. Reset Configuration Sampling Timing Requirements

4.3.1.1 HARD RESET CONFIGURATION WORD

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

HARD RESET CONFIGURATION WORD

BIT	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		RES
RESET	0	0	0	0	0		0	0		0		0		0		0
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															

EARB—External Arbitration

If this bit is set (1), external arbitration is assumed. If it is cleared (0), then internal arbitration is performed. See **Section 12.12.1.1 SIU Module Configuration Register** for more information.

$\overline{IP}$—Initial Interrupt Prefix

This bit defines the initial value of the MSR_{IP} immediately after reset. The MSR_{IP} bit defines the interrupt table location. If $\overline{IP}$ is zero (default), the MSR_{IP} initial value is one, but if it is sampled one, the MSR_{IP} initial value is zero. See **Section 6.3.1.2.1 Machine State Register**.

Bits 2, 6, and 15—Reserved

These bits are reserved and should be set to 0.

BPS—Boot Port Size

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

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

ISB—Initial Internal Space Base Select

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

00 = $00000000.
01 = $00F00000.
10 = $FF000000.
11 = $FFF00000.

DBGC—Debug Pins Configuration

This field configures the functionality of the following pins.

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

10 = Reserved.

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

DBPC—Debug Port Pins Configuration

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

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

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

10 = Reserved.

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

EBDF—External Bus Division Factor

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

CLES—Core Little-Endian Swap

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

4.3.2 Soft Reset

When a soft reset event occurs, the MPC801 reconfigures the development port. Refer to **Section 18.3.2.2 Entering Debug Mode** and **Section 18.3.3.3.1 Clock Mode Selection** for more information.

SECTION 5
CLOCKS AND POWER CONTROL

The PowerQUICC has an on-chip oscillator, clock synthesizer, and low-power divider that gives you a comprehensive set of choices for generating system clocks. They provide you with many opportunities to save power and system cost without forcing you to sacrifice flexibility or control. The main timing reference for the MPC801 can be a high frequency crystal of 4MHz, a low frequency crystal of 32KHz, or an external frequency source at 4MHz or the system frequency. The on-chip phase-locked loop (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 MPC801 shrink level. For the most part, they are usually stable, but it is impossible to guarantee that they will remain that way because the MPC801 process may change or the external component may shift.

NOTE

The internal frequency of the MPC801 and the output of the CLKO pins is dependent on the quality of the crystal circuit and the multiplication factor used in the PLLCR.

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 operating frequency that should be at least 15MHz. It can be divided by a power of two to generate the system operating frequencies. Another responsibility of the MPC801 and part of the clock section are the clocks to the timebase, decrementer, real-time clock, and periodic interrupt counter. The oscillator, timebase, decrementer, real-time clock, and periodic interrupt counter are all powered by the keep alive power supply (KAPWR) that allows the counters to continue counting at 32KHz/4MHz, even when the main power to the MPC801 is off. While the power is off, the periodic interrupt timer can be used to notify the integrated circuit power supply that power should be sent to the system at specific intervals. This is the power-down wake-up feature. When the core is not in power-down low-power mode, the keep alive power (KAPWR) is powered to the same voltage value as that of the I/O buffers and logic. Therefore, if the internal power supply is 2V and the I/O buffers and logic voltage are 3.3V, the KAPWR is (2.9 ÷ 3.3)V. For more details refer to **Section 5.1 The Clock Module** and **Section 5.10.1 Configuration**. Figure 5-1 illustrates the clock unit's functional block diagram.

5

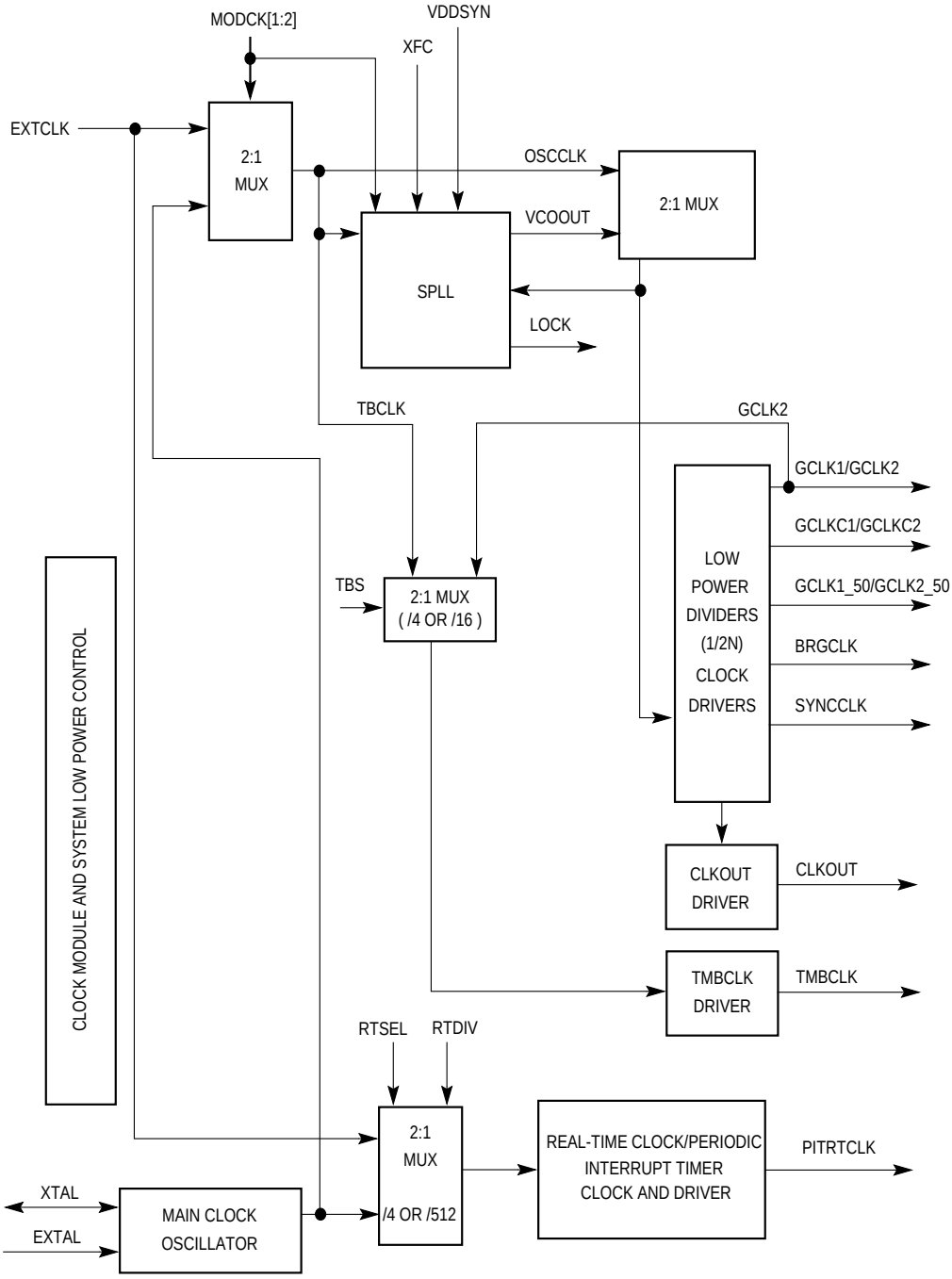


Figure 5-1. Clock Unit Block Diagram

5.1 THE CLOCK MODULE

The MPC801 clock module consists of the main crystal oscillator, the SPLL, low-power divider, clock generator/driver blocks, and clock module/system low-power control block. The clock module and system low-power control block receives control bits from the system clock control register, the PLL, the low-power and reset control register, and the reset status register. To improve noise immunity, the charge pump and the VCO of the SPLL have their own set of power supply pins (VDDSYN and VSSSYN), whereas KAPWR and VSS power the following clock unit modules.

- Oscillator
- pitrtclk and tmbclk generation logic
- DB
- Decrementer
- Real-time clock
- Periodic interrupt timer
- System clock and reset control register (SCCR)
- PLL low-power and reset control register (PLLRCR)
- Reset status register (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 at least as big as the one connected to VDDL. The power supply for each block is listed in Table 5-1 and described in **Section 5.9 Basic Power Structure**.

Table 5-1. MPC801 Power Supply

	VDDH	VDDL	VDDSYN	KAPWR
I/O Pad Logic	X			
CLKOUT	X			
SPLL (Digital)	X			
Clock Block	X			
Internal Logic		X		
Clock Drivers		X		
SPLL (Analog)			X	
Main Oscillator				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 = 3.3V ±10%
- VDDH ≥ VDDL ≥ 2.2V ±10%
- VDDH ≥ KAPWR ≥ VDDH - 0.4V at normal operation
- KAPWR ≥ 2V in power-down mode

The timing diagram for the internal clocks generated in the MPC801 is illustrated in Figure 5-2.

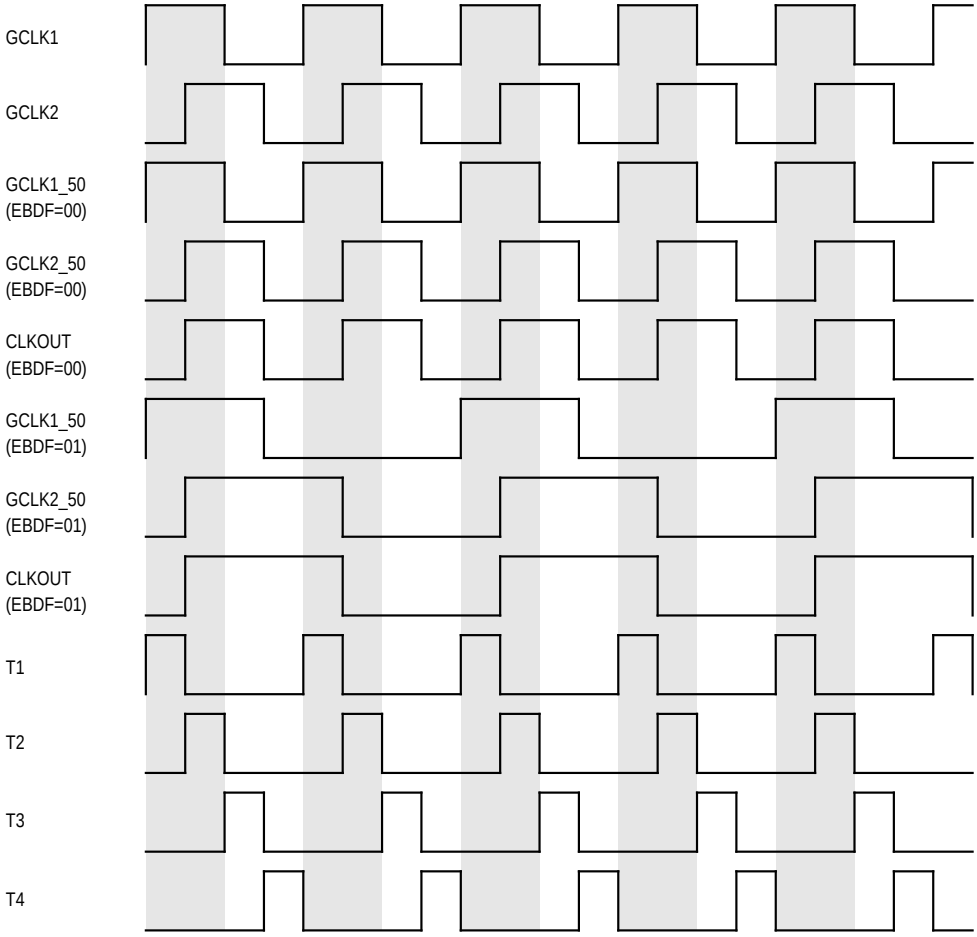


Figure 5-2. MPC801 Clocks Timing Diagram

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 MPC801 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 MPC801 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. See 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 $\overline{\text{PORESET}}$ pin. The configuration modes are shown in the table below. MODCK1 specifies the input source to the SPLL and, combined with MODCK2, specifies the multiplication factor (MF) 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 pins should not be sampled on wake-up from this mode. In this case, the $\overline{\text{PORESET}}$ pin should remain negated while the $\overline{\text{HRESET}}$ pin is asserted during the power-up wake-up stage.

Table 5-2. Reset Clocks Source Configuration

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

When the MODCK1 bit is clear (0), the output of the oscillator with a 4MHz or 32MHz frequency is 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 the DFNH and DFNL 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 4MHz clock is supplied as oscclk, but when it is clear (0), the input frequency is either 32KHz (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 the oscillator is the timing reference to the real-time clock and periodic interrupt timer, the frequency of the oscillator connected to oscillator should be in the 32KHz range. The TBS bit in the system clock and reset control register (SCCR) can select the timebase clock to be either the SPLL input clock or gclk2. The periodic interrupt timer and real-time clock frequency and source are specified by the RTDIV and RTSEL bits in the SCCR. 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 AT RESET	MF + 1	TMBCLK DIVISION
1	—	—	16
0	0	—	4
0	1	1, 2	16
0	1	> 2	4

NOTE

The voltage on the MODCK1 and MODCK2 pins should always be less than or equal to the VDDH power supply voltage applied to the part.

5.2 ON-CHIP OSCILLATORS AND EXTERNAL CLOCK INPUT

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

NOTE

The internal frequency of the MPC801 and the output of the CLKO pins is dependent on the quality of the crystal circuit and multiplication factor 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 3MHz ÷ 5MHz or it should be at the system frequency of at least 15MHz (1:1 mode). When the external clock input is the timing reference to the system, PLL skew elimination between the EXTCLK and CLKOUT pins is less than ±1ns.

For normal operation, at least one clock source should be active, but you can also configure both clock sources to be active. In this configuration, EXTCLK provides the oscclk and oscillator provides the pitrtclk. The input of an unused timing reference should be grounded.

5.3 THE SYSTEM PHASE-LOCKED LOOP

The system PLL performs frequency multiplication and skew elimination and 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 the cost by eliminating the need to add another oscillator to the system. The system PLL block diagram is illustrated in Figure 5-3.

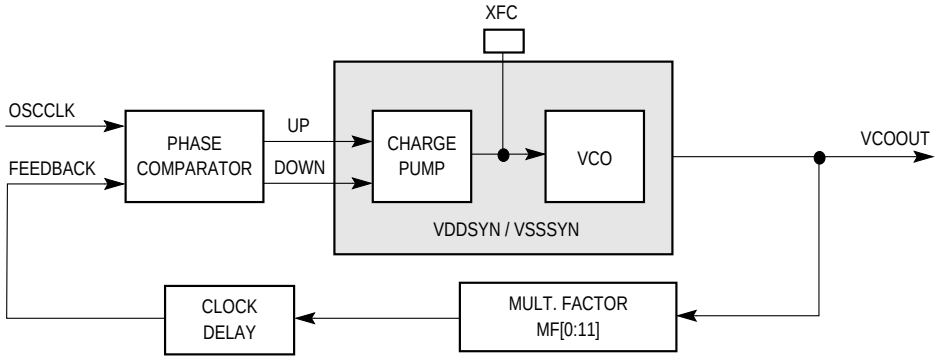


Figure 5-3. System PLL Block Diagram

5.3.1 Multiplying the Frequency

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 bits in the PLL low-power and reset control register. Even though you can program any integer value from 1 to 4,096, the resulting VCO output frequency must be in the range specified in **Section 20 Electrical Characteristics**. As defined in Table 5-5, the multiplication factor is set to a predetermined value during power-on reset.

5.3.2 Eliminating Skew

The PLL is capable of eliminating the skew between the external clock entering the core, 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 can be much larger.

5.3.3 Operating the PLL Block

The reference signal is sent to the phase comparator that controls the up and down direction of the charge pump driving the voltage across the external filter capacitor. 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 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 keep alive power is lost, power-on reset should be asserted by the 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 the SCCR are set to the value of 0 (÷1), respectively. This value then programs the SPLL to generate the default system frequency of approximately 16.7MHz for a 32KHz input frequency and 20MHz for a 4MHz input frequency.

5.4 THE LOW-POWER DIVIDER

The output of the PLL is sent to a low-power divider block that generates all other clocks in normal operation, but divides the output frequency of the VCO before it generates the SYNCCLK, SYNCCLKS, BRGCLK, and general system clocks sent to the rest of the MPC801. GCLK1C and GCLK2C are the system timing references for the PowerPC core as well as the instruction and data caches and memory management units. GCLK1 and GCLK2 are the system timing references for all other modules. GCLK1_50 and GCLK2_50 can operate 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 you to reduce and restore the operating frequencies of different sections of the MPC801 without losing the PLL. Using the low-power divider block, full chip operation can be obtained at a lower frequency. This feature is called slow-go or gear mode. The selection and speed of the slow-go mode can be changed at any time and the changes occur immediately. The low-power divider block is controlled in the SCCR and its default state is to divide all clocks by one. So, for a 40MHz system, the SYNCCLK, SYNCCLKS, BRGCLK, and general system clocks are each 40MHz.

5.5 INTERNAL CLOCK SIGNALS

The internal logic of the MPC801 uses 9 internal clock signals:

- General system clocks—GCLK1C, GCLK2C, GCLK1, GCLK2, GCLK1_50, and GCLK2_50
- Baud rate generator clock—BRGCLK
- Synchronization clocks—SYNCCLK and SYNCCLKS

The MPC801 also generates an external clock signal called CLKOUT. The PLL synchronizes these clock signals to each other.

5.5.1 The General System Clocks

The general system clocks—GCLK1C, GCLK2C, GCLK1, GCLK2, GCLK1_50, and GCLK2_50—are the basic clocks supplied to all modules and submodules on the MPC801. GCLK1C and GCLK2C are supplied to the PowerPC core, data and instruction caches, and memory management units and they can be stopped when the core enters the doze low-power mode. GCLK1 and GCLK2 are supplied to the system interface unit, clock module, and communication modules.

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

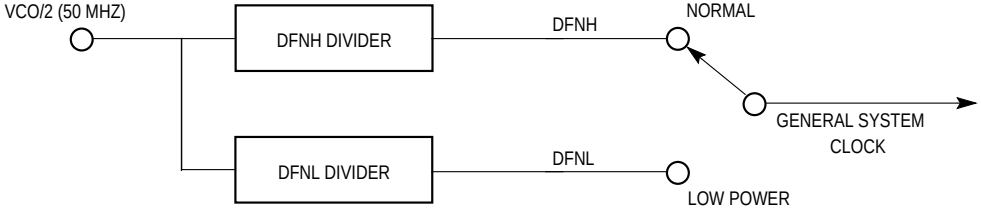


Figure 5-4. General System Clocks Select

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 and DFNH=0. The general system clock can be operated at a low or high frequency. Low is defined by the DFNL bits of the SCCR and high is defined by the DFNH bits.

Software can change the frequency of the general system clock on-the-fly. The general system clock can be forced to switch to its low frequency. However, in some applications, a high frequency is required during certain periods. For example, interrupt routines require a higher performance than a low frequency operation provides, but they consume less power than a maximum frequency operation provides.

The MPC801 is capable of automatically switching between low and high frequency operation whenever one of the following conditions exist:

- A pending interrupt from the interrupt controller occurs. This option is maskable by the PRQEN bit in the SCCR.
- The POW bit in the core's machine state register is clear. This option is maskable by the PRQEN bit in the SCCR.

When none of these conditions exist and the CSRC bit of the PLL low-power reset control register is set, the general system clock automatically switches back to the low frequency. When the general system clock is divided, its duty-cycle is changed. One phase remains the same (12.5ns @ 40MHz) 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.

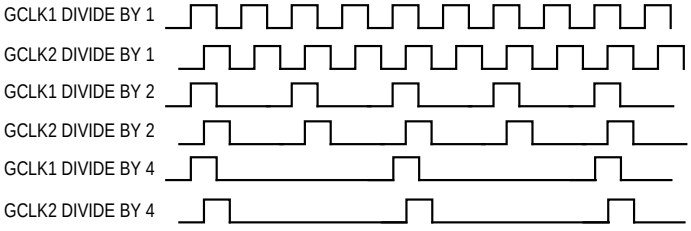


Figure 5-5. Divided System Clocks Timing Diagram

The frequency for system clocks GCLK1and GCLK2 is:

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

The frequency for clocks GCLK1_50 and GCLK2_50 is:

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

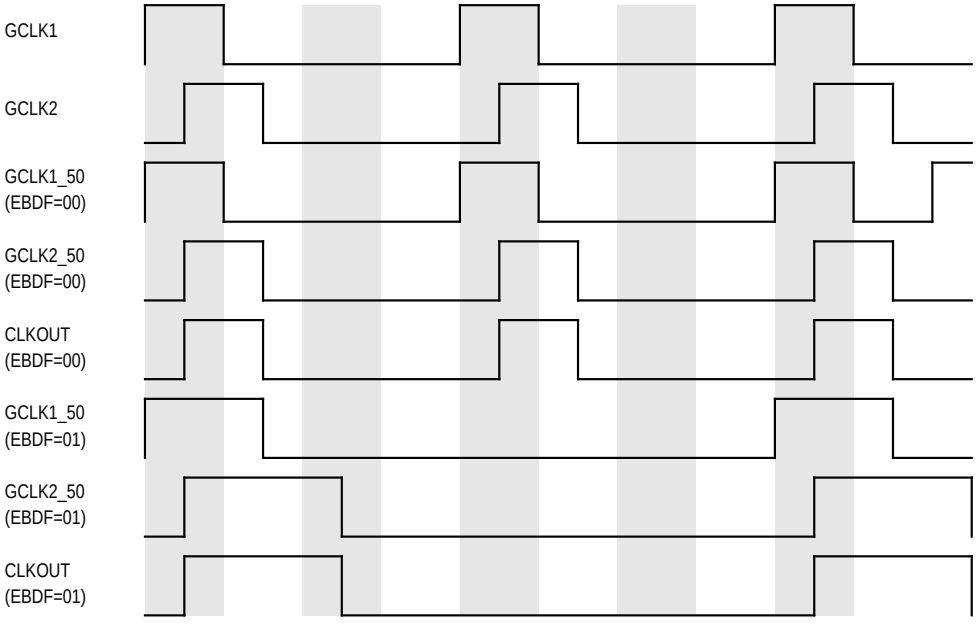


Figure 5-6. MPC801 Clocks For DFNH = 1 or DFNL = 0 Timing Diagram

5.5.2 The Baud Rate Generator Clock

The baud rate generator clock (BRGCLK**)** is used by the communication modules and the memory controller refresh counter. It defaults to VCO/2 = 40MHz, assuming a 40MHz system frequency. The purpose of BRGCLK is to allow the communication modules to continue operating at a fixed frequency, even when the rest of the MPC801 is operating at a reduced frequency. The baud rate generator clock frequency is:

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

5.5.3 The Synchronization Clocks

The synchronization clock (SYNCCLK) is used by the serial synchronization circuitry in the serial ports of the communication modules and includes the serial interface, serial communication controllers, and serial management controllers. It synchronizes externally generated clocks before they are used internally. SYNCCLK defaults to VCO/2 = 40MHz, assuming a 40MHz system frequency. The SYNCCLK is used by the serial interface internal logic.

The purpose of SYNCCLK is to allow the communication modules to continue operating at a fixed frequency, even when the rest of the MPC801 is operating at a reduced frequency. Thus, the SYNCCLK allows you 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 timeslot assigner in the serial interface is used.

The SYNC clock frequency is:

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

The CLKOUT is the same as GCLK2_50. It defaults to VCO/2 = 40MHz, assuming a 40MHz system frequency. The SCCR controls whether it drives full strength, half strength, or is disabled. 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.

5.6 THE PHASE-LOCKED LOOP PINS

The following pins are dedicated to PLL operation.

NOTE

The internal frequency of the MPC801 and the output of the CLKO pins is dependent on the quality of the crystal circuit and the MF bits of the PLPRCR. Please refer to the sections on phase-lock loop for details about PLL performance.

VDDSYN—Drain Voltage

The VDD pin is dedicated to 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

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

VSSSYN1—Source Voltage 1

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

XFC—External Filter Capacitor

This pin connects to the off-chip capacitor for the PLL filter. One terminal of the capacitor is connected to XFC and the other is connected to VDDSYN.

NOTE

30MΩ 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

	MINIMUM CAPACITANCE	MAXIMUM CAPACITANCE	UNIT
MF < = 4	XFC = MF * 425 - 125	XFC = MF * 590 - 175	pF
MF > 4	XFC = MF * 520	XFC = MF * 920	pF

5.7 CONTROLLING THE SYSTEM CLOCK

The system phase-loop lock has a 32-bit control register that is powered by keep alive power. This system clock and reset control register (SCCR) is memory-mapped into the MPC801 system interface unit register map.

SCCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	COM		RESERVED			TBS	RTDIV	RTSEL	RES	PRQEN	RESERVED		EBDF		RES
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RES	DFSYNC		DFBRG		DFNL			DFNH			RESERVED				

Bits 0, 3–5, and 9—Reserved

These bits are reserved and should be set to 0.

COM—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, both bits (disabling CLKOUT) should be set to minimize noise and power dissipation. The COM bits are cleared by a hard reset.

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

TBS—Timebase Source

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

0 = 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—Real-Time Clock Divide

This bit indicates if the clock to the real-time clock and periodic interrupt timer 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.

0 = The real-time clock and periodic interrupt timer are divided by 4.
1 = The real-time clock and periodic interrupt timer are divided by 512.

RTSEL—Real-Time Clock Circuit Input Source Select

This bit specifies the input source to the real-time clock. At power-on reset, this bit receives the value of the MODCK[1] bit.

0 = The real-time clock and periodic interrupt timer are divided by 4.
1 = The real-time clock and periodic interrupt timer are divided by 512.

PRQEN—Power Management Request Enable

This bit specifies whether or not the general system clock returns to the high frequency defined by DFNH while there is a pending interrupt from the interrupt controller or POW bit in the machine state register is clear. This bit is cleared by power-on or hard reset.

0 = The system remains in the lower frequency defined by DFNL even if there is a pending interrupt from the interrupt controller or POW bit in the machine state register is cleared.
1 = The system switches to high frequency defined by DFNH when there is a pending interrupt from the interrupt controller or POW bit in the machine state register is cleared.

Bits 11–12 and 15–16—Reserved

These bits are reserved and should be set to 0.

EBDF—External Bus Division Factor

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

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

DFSYNC—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.

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

DFBRG—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.

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

DFNL—Division Factor Lowest Frequency

These bits are required for two reasons—to reduce the general system clock to a frequency lower than that which can be obtained in the DFNH bits and to automatically switch between the DFNL and DFNH rates. These bits are cleared by the power-on or hard reset.

These bits can be loaded with the preferred divide value and the CSRC bit must be set 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.

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

DFNH—Division Factor High Frequency

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

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

Bits 27–31—Reserved

These bits are reserved and should be set to 0.

5.8 PLL LOW-POWER AND RESET CONTROL REGISTER

The 32-bit PLL low-power and reset control register (PLPRCR) is powered by a keep alive power supply. This register is memory-mapped into the MPC801 serial interface unit register map.

PLPRCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	MF												RESERVED			
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SPLSS	TEXPS	RES	TMIST	RES	CSRC	LPM		CSR	LOLRE	FIOPD	RESERVED				

MF—Multiplication Factor

The output of the voltage control oscillator (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. 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

This bit is not affected by hard reset. An out-of-lock indication sets the SPLSS bit and it remains set until the software clears it. At power-on reset, the state of the SPLSS bit is zero.

The SPLSS bit is negated by writing a 1 (writing a zero has no effect). The SPLSS bit is not affected by zero because of a software-initiated loss of lock, which is defined as an multiplication factor change or entering deep sleep and power-down modes.

0 = SPLL has remained in lock.
1 = SPLL has gone out of lock at least once, but not because of a change with the PLLEN or MF bits.

TEXPS—Timer Expired Status

This bit is set by a reset. If it is enabled, 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. TEXPS is negated by writing a 1 (writing a zero has no effect). When TEXPS is set, the TEXP external signal is asserted and when it is reset, the TEXP external signal is negated.

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

TMIST—Timers Interrupt Status

This bit is cleared at reset and is set when either the real-time clock, periodic interrupt timer, timebase, or decrementer interrupt occurs. It is cleared by writing a 1 (writing a zero has no effect). The system clock frequency remains at a high frequency value defined by the DFNH bits if the TMIST bit is set. The clock frequency remains high if the CSRC bit in the PLPRCR is set and there are no conditions for switching to normal low mode.

0 = No timer expiration event is detected.
1 = A timer expiration event is detected.

CSRC—Clock Source

This bit specifies the bit that determines the general system clock—DFNH or DFNL. Setting this bit switches the general system clock to the DFNL value needed to enter slow-go low-power mode. Clearing this bit switches the general system clock to the DFNH value. CSRC is cleared at hard reset.

0 = General system clock is determined by the DFNH value
1 = General system clock is determined by the DFNL value

LPM—Low-Power Modes

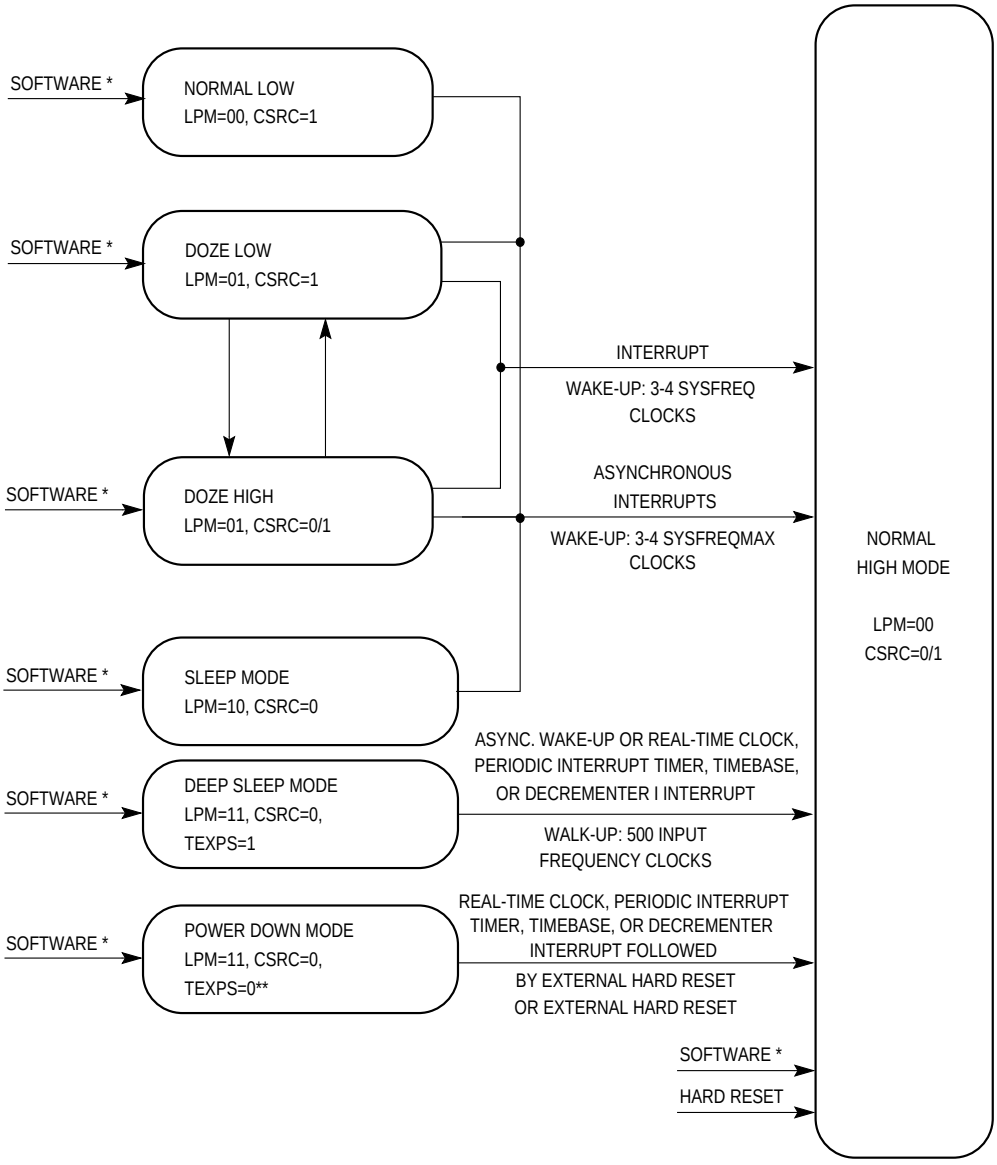
These bits are encoded to provide one normal operating mode and four low-power modes. In normal and doze modes, the system can be in the high state defined by the DFNH bits or in the low state defined by the DFNL bits. The normal high operating mode is the state out of reset. This is also the state the 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. Table 5-5 provides more details on these bits.

Table 5-5. MPC801 Low-Power Modes

OPERATION MODE	SPLL	CLOCKS	WAKE-UP METHOD	RETURN TIME FROM WAKE-UP EVENT TO NORMAL HIGH	MPC801 POWER CONSUMPTION AT 50MHZ	FUNCTIONALITY
Normal High LPM=00	Active	Full Freq. div 2^{DFNH}	—	—	× 20 mWatt+ $1/2^{DFNH}$ Watt	Full Functions Not In Use Are Shut Off
Normal Low (“Gear”) LPM=00	Active	Full Freq. div 2^{DFNL+1}	Software or Interrupt	Asynchronous Interrupts: 3-4 Maximum System Cycles Synchronous Interrupts: 3-4 Actual System Cycles	× 20 mWatt+ $1/2^{(DFNL+1)}$ Watt	
Doze High LPM=01	Active	Full Freq. div 2^{DFNH}	Interrupt		× 20mWatt+ $0.4/2^{DFNH}$ Watt	Enabled: RTC, PIT, and MEMC, Disabled: Extended Core
Doze Low LPM=01	Active	Full Freq. div 2^{DFNL+1}	Interrupt		× 20 mWatt+ $0.4/2^{(DFNL+1)}$ Watt	
Sleep LPM=10	Active	Not Active	Interrupt	3-4 Maximum System clocks	<10 mW	Enabled: RTC, PIT, TB, and DEC
Deep Sleep LPM=11 TEXPS=1	Not Active	Not Active	Interrupt	<500 Oscillator Cycles 16 msec-32 KHz 125 µsec-4 MHz	TBD	
Power-Down LPM=11 TEXPS=0	Not Active	Not Active	Interrupt	<500 Oscillator Cycles + Power Supply Wake-Up (PwSp_Wake+ 16 msec) @ 32 KHz	32 KHz- ~10 µA, KAPWR = 3.0 V Temperature=50° C	

Table 5-5 describes all possible transfers between low-power modes. The MPC801 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 asynchronous interrupt is responsible for exiting from normal, low doze high, low, and sleep modes to normal high mode. The asynchronous interrupt sources are:

- Asynchronous wake-up interrupt from the interrupt controller
- Real-time clock, periodic interrupt timer, timebase, or decrementer 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 5-7. MPC801 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 40MHz system, this wake-up takes 75ns to 100ns. 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 real-time clock, periodic interrupt timer, timebase, or decrementer interrupts set status bits in the PLPRCR. 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 50MHz system where DFNL=111 (divided by 256), the wake-up time is 12.8 to 25.6μs. In normal and doze modes, if the $PLPRCR_{CSRC}$ 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:

- A pending interrupt from an interrupt controller must occur
- The POW bit in the machine state register must be cleared (normal mode)

If none of these conditions exist, the $PLPRCR_{CSRC}$ bit is set, the asynchronous interrupt status bits are reset, and the system switches back to normal 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 caused by:

- An asynchronous wake-up interrupt from the interrupt controller
- Real-time clock, periodic interrupt timer, timebase, or decrementer interrupts (if enabled)

In deep sleep mode, the PLL is disabled and the wake-up time from this mode is a maximum of 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 32KHz the wake-up time is less than 15.6ms and if it is 4MHz the wake-up time is less then 125μs.

An 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 real-time clock, periodic interrupt timer, timebase, or decrementer interrupt. The hard reset takes longer than the time it takes the power supply to wake up, in addition to the time it takes the PLL to lock. Hard reset assertion when the TEXPS bit and TEXP pin are clear, sets the bit and the pin values, thus causing an exit from power-down low-power mode. For more details on power-down mode, refer to **Section 5.10.1 Configuration**.

NOTE

The chip is only allowed to enter deep sleep low-power mode and power-down mode if the main timing reference is a 32KHz crystal oscillator.

CSR—Checkstop Reset Enable

If this bit is set, an automatic reset is generated when the 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 system interface unit does nothing when a checkstop signal is received from the core. However, if debug mode is enabled, then the part enters debug mode when entering checkstop mode. In this case, the core does not assert the checkstop signal to the reset circuitry.

0 = The checkstop condition does not cause automatic reset.
1 = The checkstop condition causes automatic reset.

LOLRE—Loss of Lock Reset Enable

This bit specifies the manner in which the clocks handle a loss of lock indication. When this bit is clear, a hard reset is not asserted if a loss of lock indication occurs. However, when it is set, a hard reset is asserted when a loss-of-lock indication occurs.

0 = Loss of lock does not cause reset.
1 = Loss of lock causes reset.

FIOPD—Force I/O Pull Down

This bit 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 deep sleep low-power mode, the A and D 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), the A and D external pins are unaffected.

0 = The address and data pins are not driven by an internal pull-down device.
1 = The address and data pins are driven by an internal pull-down device.

5.9 BASIC POWER STRUCTURE

The MPC801 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 more details about the relationships between the power supplies, refer to **Section 5.1 The Clock Module**.

The I/O buffers, logic, and clock block are fed by a 3.3V 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 2V source allowing a considerable reduction in power consumption. The PLL is fed by a 3.3V power supply (VDDSYN) to achieve a high stability in its output frequency. The oscillator, real-time clock, periodic interrupt timer, timebase, or decrementer are all fed by the KAPWR rail, thus allowing the external power supply unit to disconnect all other subunits at low-power deep sleep mode. The TEXP pin (fed by the same rail) can switch between sources using the external power supply unit.

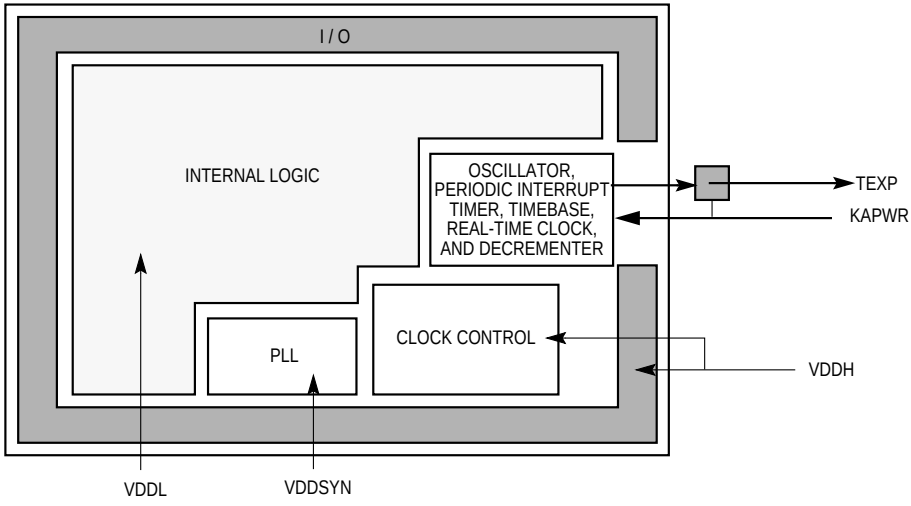


Figure 5-8. MPC801 Basic Power Supply Configuration

5.10 KEEP ALIVE POWER

5.10.1 Configuration

An example of a switching scheme for an optimized low-power system is illustrated in Figure 5-9.

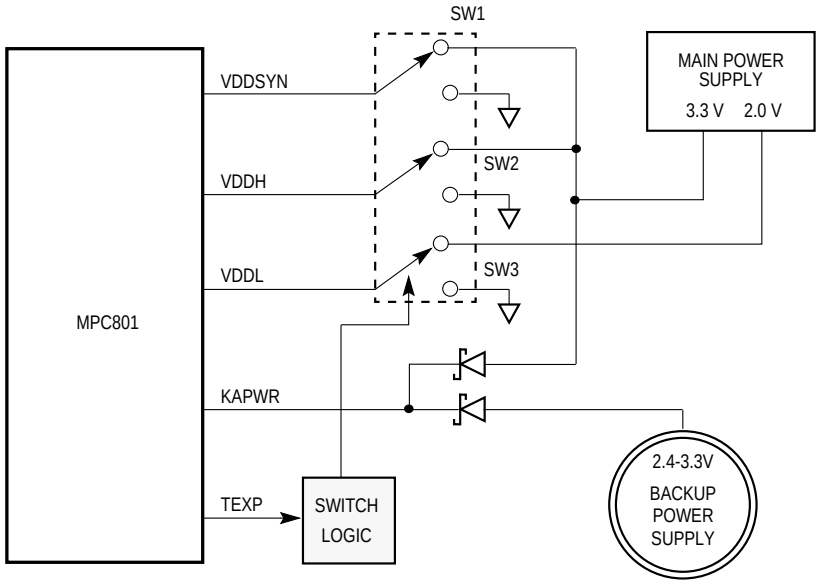


Figure 5-9. External Power Supply Scheme (2.0 V Internal Voltage)

SW1 and SW2 can be unified in one switch if VDDSYN and VDDH are supplied by the same source. If VDDL is fed with 3.3V, SW2 and SW3 can be combined into one switch. The TEXP signal, if enabled, is asserted by the MPC801 when the real-time clock or timebase time value matches the value programmed in its associated alarm register or when the periodic interrupt timer or decrementer decrements their value to zero. The TEXP signal is negated when writing to the TEXPS status bit with the corresponding data bit being 1. The KAPWR power supply feeds the oscillator. The condition for main crystal oscillator stability is that the power supply value changes slowly. The maximum slope of the KAPWR power supply should be less than 1.7V/mS for a 32KHz input frequency. An exponential model of voltage change on the KAPWR rail should ensure that $\tau > 20/freq_{oscm}$.

5.10.2 The Key Mechanism

All the registers defined in the system integration, decrementer, timebase timers, and the clocks and reset of Table 3-1 are powered by the KAPWR supply. When power-down mode is entered, the value stored in these registers is preserved when the main power supply is disconnected. If this requirement is not met, data loss can occur in these registers. To reduce the possibility of data loss, the MPC801 includes a key mechanism that ensures data retention on locked registers. While a register is locked, all writes are ignored and a machine check exception is generated.

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

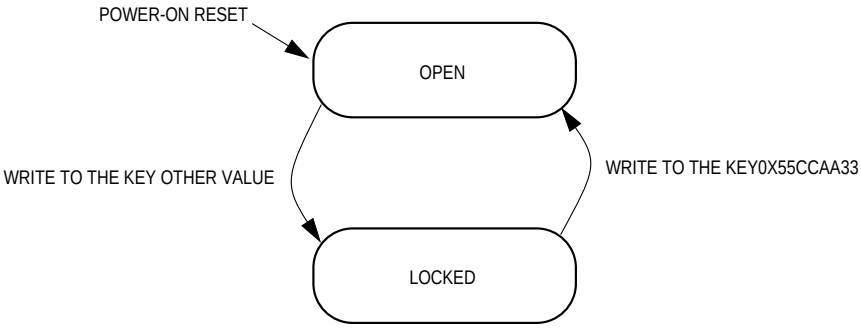


Figure 5-10. Key Mechanism Diagram

The key registers for the system integration timer and the clock and reset registers are defined in Table 3-1.

SECTION 6
THE POWERPC CORE

The core is the module of the MPC801 where the PowerPC™ architecture is implemented. It has the functionality of the PowerPC branch and fixed-point processors, in addition to all the PowerPC user mode instructions (with the exception of the floating-point instructions) and all the registers associated with them. It also contains part of the development support features of the MPC801, including breakpoint and watchpoint 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 based on a document called the *PowerPC Family: The Programming Environment (MPCFPE/D)* that explains the architecture of the PowerPC in great detail.

NOTE

As needed, this manual only contains references to those documents and does not duplicate any information already specified there.

6.1 FEATURES

The following list provides the main features of the MPC801 core:

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

6.1.1 Basic Structure of the Core

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

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

6.1.2 Instruction Flow Within the Core

When fetched, instructions enter the instruction queue, thus enabling 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 when an exception on a previous instruction, interrupt, or miss-predicted fetch occurs.

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

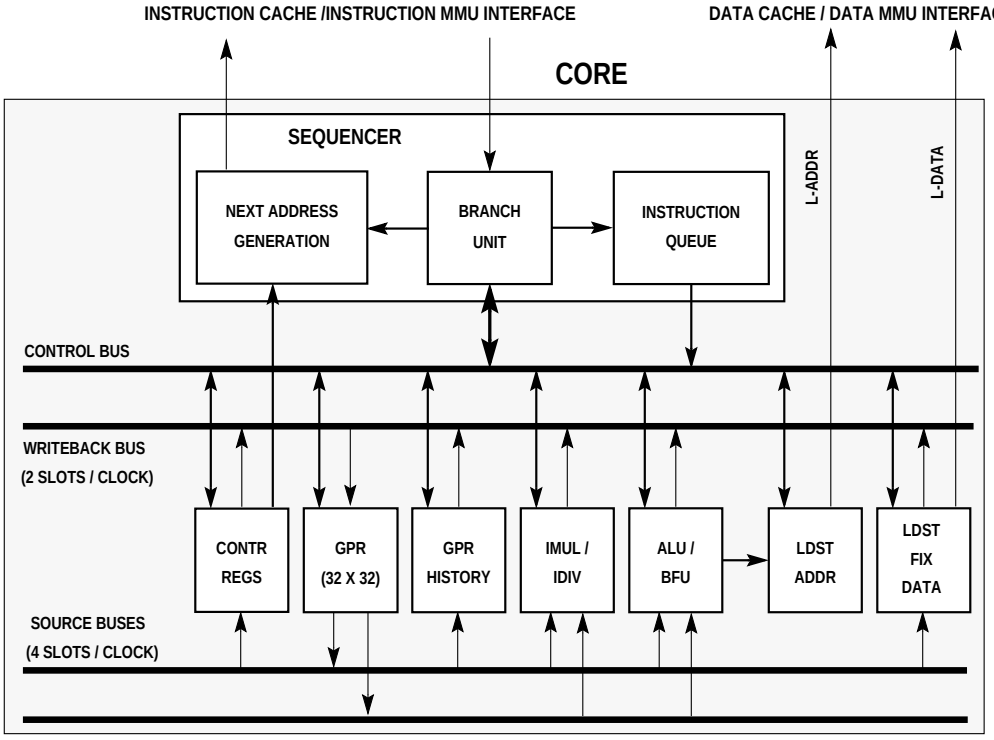


Figure 6-1. Core Block Diagram

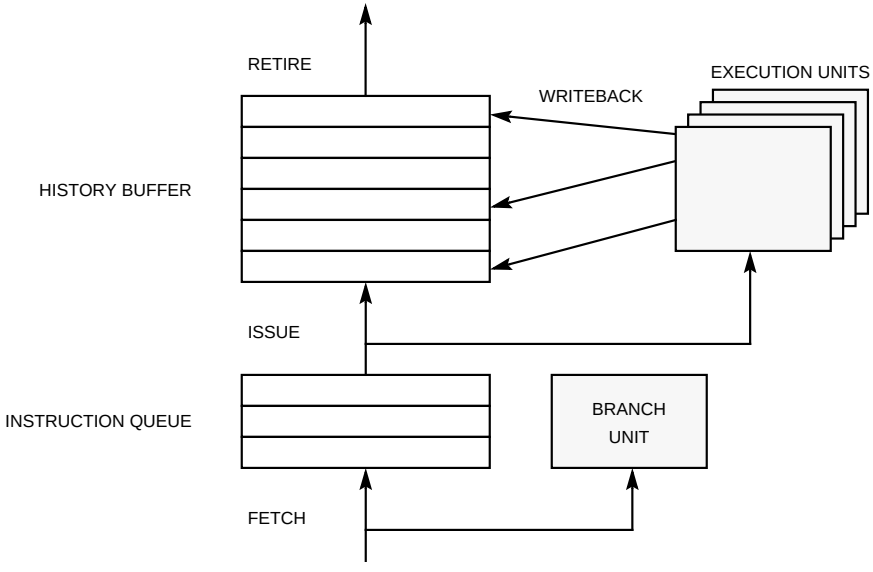


Figure 6-2. Instruction Flow Conceptual Diagram

6.1.3 Basic Instruction Pipeline

Figure 6-3 illustrates basic instruction pipeline timing.

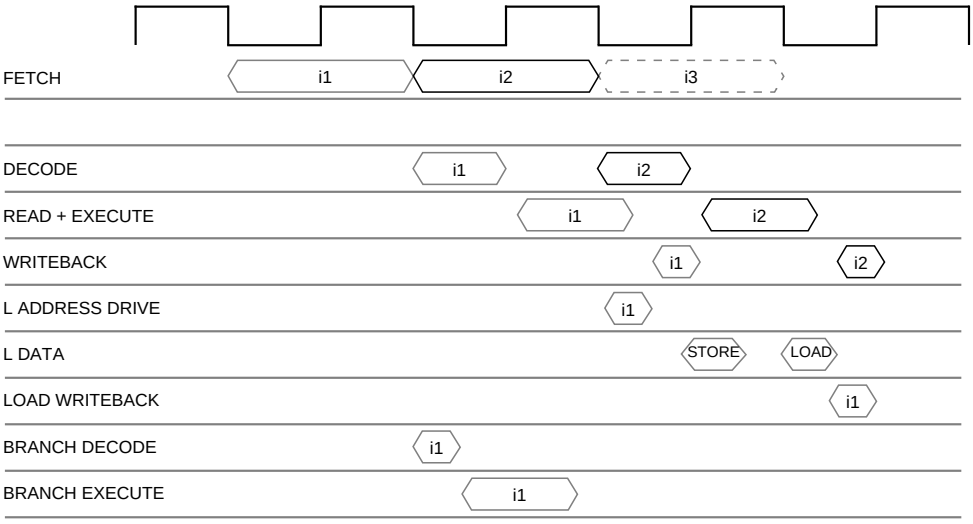


Figure 6-3. Basic Instruction Pipeline Timing Diagram

6.2 THE SEQUENCER UNIT

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

6.2.1 Flow Control

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

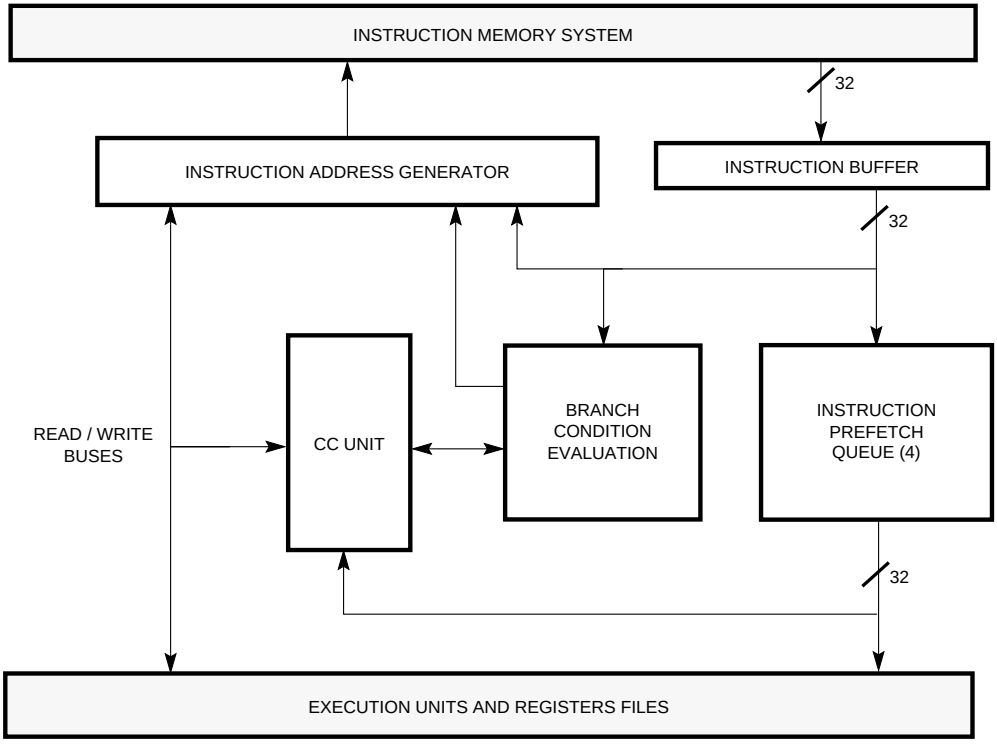


Figure 6-4. Sequencer Data Path

To execute branches in parallel with the execution of sequential instructions, the sequencer maintains an instruction prefetch queue that is four instructions deep. In an ideal situation, a sequential instruction would be issued every clock, even when branches are present in the code. This is referred to as branch folding. The instruction prefetch queue also eliminates stalls caused by long latency instruction fetches and all instructions are fetched into the instruction prefetch queue, but only sequential instructions are issued to the execution units when they reach the head of the queue. The reason branches enter the queue is for watchpoint marking (refer to **Section 18 Development Support** for details). Since branches do not prevent the issue of sequential instructions unless they come in pairs, the performance impact of entering branches in the instruction prefetch queue is negligible.

In addition to branch folding, the core implements a branch reservation station and static branch prediction so branches can issue as early as possible. The reservation station allows a branch instruction to issue even before its condition is ready. With the branch issued and out of the way, instruction prefetch can continue while the branch operand is being computed and the condition evaluated. Static branch prediction determines the instruction stream to be 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 predicted and these branches, which later turn out to have followed the wrong path, are mispredicted. Branch instructions that issue with source data already available are unpredicted and those instructions fetched under a predicted branch are fetched conditionally. The core ignores conditionally prefetched instructions fetched under a mispredicted branch.

Table 6-1. Branch Prediction Policy

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

6.2.2 Issuing Instructions

The sequencer attempts to issue a sequential instruction on each clock whenever possible. However, for an instruction to issue, the execution unit must be available and the required source data is available and no other executing instruction targets the same destination register. The sequencer informs the execution units of the instruction's existence on the instruction bus. The execution units then decode the instruction, interrogate the register unit, and inform the sequencer that it accepts the instruction for execution.

6.2.3 Interrupts

The core interrupts can be generated when an exception occurs. An exception results when an instruction is executed or an asynchronous external event occurs. There are five exception sources in the MPC801:

- External interrupt request
- Certain memory access conditions
- Internal errors, such as an attempt to execute an unimplemented opcode or floating-point arithmetic overflow
- Trap instructions
- Internal exceptions

Interrupt handling 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 machine state register. When the exception has been handled, the handler can resume execution of the user program without the knowledge that such an event ever occurred.

As specified in *PowerPC Family: The Programming Environment*, the core implements a precise interrupt model. An interrupt usually occurs under one of the following conditions:

- No instruction that logically follows the faulting instruction in the code stream has started executing.
- All instructions preceding the faulting instruction 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 may not have started executing (“before interrupt”), could be partially completed, or has completed (“after interrupt”) depending on the interrupt and instruction types. See Table 6-2 for details.

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

Table 6-2. “Before” and “After” Interrupts

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

6.2.4 Precise Exception Model Implementation

To achieve maximum performance, many pieces of the instruction stream are concurrently processed by the core, regardless of the sequence specified by the executing program. Instructions execute in parallel and are completed out of order. The hardware works hard to ensure that this out-of-order operation never has any effect besides the one specified by the program. This is most difficult to safeguard 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 that caused the interrupt. By doing it, the core implements a precise exception model. This is, of course, assuming the instruction that caused the exception has not already begun when the interrupt occurs.

To recover from an interrupt, a history buffer is used. This buffer is a FIFO queue that records the relevant machine state at the time of each instruction issue. When issued, instructions are placed on the tail of the queue and they percolate to the head of the queue while they are executing. Instructions remain in the queue until they finish executing 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 finish executing, they are released (retired) from the queue and the buffer storage is reclaimed for new instructions entering the queue.

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

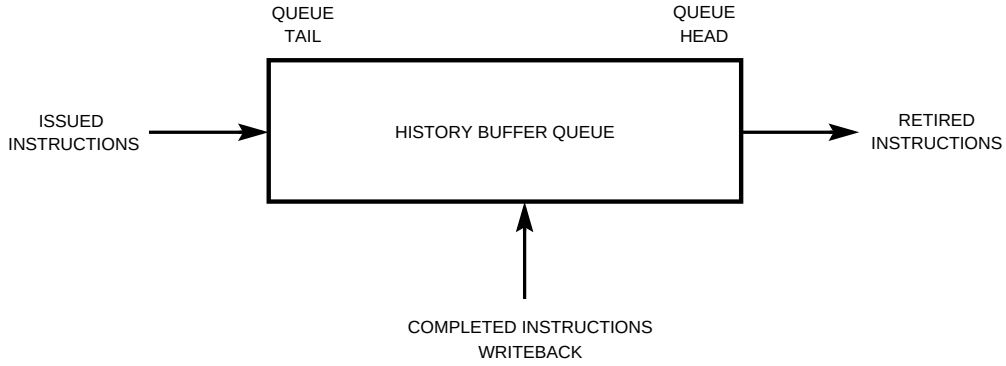


Figure 6-5. History Buffer Queue

To correctly restore the architectural state, the history buffer must record the value of the destination 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 (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) for issued instructions to fill up the history buffer. If so, instruction issue waits until the long latency operation (blocking retirement) finishes.

The following types of instructions can potentially cause the history buffer to fill up:

- Floating-point arithmetic instructions
- Integer divide instructions
- Instructions that affect/use resources external to the core

6.2.4.1 RESTARTABILITY AFTER AN INTERRUPT

Most interrupts in the core are restartable, but some are nonrestartable if they are only recognized when the machine status save/restore 0 and 1 registers (SRR0 and SRR1) are busy. System reset and machine check interrupts are the only interrupts that can be nonrestartable within the PowerPC architecture.

Most of the interrupt types defined in the architecture should be restartable. It 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 or between the restoration of these registers and the execution of the **rfi** 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. Since there might be a situation in which it is preferable to be restartable, a mechanism is defined to notify the interrupt handler code when it is in a restartable state.

The mechanism uses a bit within the machine state register (MSR) called the recoverable interrupt bit (MSR_{RI}). The MSR_{RI} shadow bit in the SRR1 register indicates if the interrupt is restartable or not. Notice that this bit does not need to be checked on interrupt types that are restartable by convention, except those mentioned above. 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 and cleared. Every time an **rfi** instruction is executed, MSR_{RI} is copied from its shadow in the SRR1 register. In addition, it can be altered by the software via the **mtmsr** (move to special register) instruction. The MSR_{RI} bit is intended to be set by the interrupt handler software after saving the machine state, and cleared by the interrupt handler software before retrieving the machine state.

In critical code sections where MSR_{EE} is negated, but the SRR0 and SRR1 registers are not busy, MSR_{RI} should be left asserted. In these cases, if an interrupt occurs, it is restartable. To facilitate the software manipulation of the MSR_{RI} and MSR_{EE} bits, the core includes special commands implemented as move to special register. The following table defines these special register mnemonics. A write (**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. For specific encoding, see Table 6-3.

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

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

6.2.5 Processing An Interrupt

The following table shows the significant events that occur when an interrupt is processed.

Table 6-4. Interrupt Latency

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

NOTES: 1. At time point A an instruction 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, thus implying that all instructions preceding it in the code stream have finished execution without generating an interrupt. Also, the excepting instruction itself has completed execution. At this time the exception is "recognized" and exception processing begins. If, at this point, the instruction had not generated an exception, it would have been retired.

3. At time point C the sequencer starts to fetch the interrupt handler's first instruction.

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

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

6.2.6 Serialization

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

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

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

6.2.6.1 SERIALIZATION LATENCY

The time required to serialize the machine is also the amount of time needed to complete the instructions currently in progress. This time is heavily dependent on the instructions in progress and the memory system latency. It is impossible to put an absolute upper bound on this time because the memory system design is not controlled by the core. 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.

6.2.7 The External Interrupt

The core provides one external interrupt line—the architectural maskable external interrupt. In the MPC801, this interrupt is generated by the on-chip interrupt controller. It is software acknowledged and maskable by the MSR_{EE} bit, which is automatically cleared by the hardware to disable external interrupts when any interrupt is taken.

6.2.7.1 LATENCY

When an external interrupt is detected, every instruction that can retire from the history buffer does so and the interrupt is assigned to the instruction at the head of the history buffer. However, the following conditions must be met before the instructions at the head of the queue can retire.

- The instructions at the head of the history buffer must be completed without exception
- The instructions at the head of the history buffer must either be a **mtspr**, **mtmsr**, or **rfi** instruction, a memory reference, or a storage or cache control instruction.

Any instruction that does not meet these criteria is discarded with all of its side effects and the execution at the end of the interrupt handler resumes with the first instruction that was discarded. If all the instructions in the history buffer were allowed to complete, execution at the end of the interrupt handler resumes with the next instruction. External interrupt latency depends on the time required to reference memory. The measurement is equal to one of the following, in addition to the interval between B and E shown 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 the interval between B and E. If the instruction prior to the one in which the interrupt gets assigned generates an exception, the exception is recognized first. If minimal interrupt latency is an important system parameter, interrupt handlers should save the machine context and reenable an external interrupt as quickly as possible so that a pending external interrupt will get fast service.

6.2.8 Interrupt Ordering

There are two major types of interrupts:

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

Instruction-related exceptions are detected while the instruction is in various stages of being processed by the core. Exceptions detected early in instruction processing avoid detection of other exceptions for the same instruction. 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. 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. Detection Order of Instruction-Related Interrupts

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
7[2]	Privileged Instruction	Attempt to Execute Privileged Instruction in Problem Mode
	Alignment Interrupt	Load/Store Checking
	System Call Interrupt	SC Instruction
	Trap	Trap Instruction
8	Implementation Dependent Data TLB Miss	Data 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 go as if no trace is enabled and trapping the second instruction. This, of course, refers to this second instruction.

2. Exclusive for any one instruction.

More than one asynchronous interrupt cause 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

6.3 THE REGISTER UNIT

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

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

6.3.1 The Control Registers

The following tables describe the control registers that are implemented within the MPC801.

Table 6-7. Standard Special-Purpose Registers

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

Table 6-8. Standard Timebase Register Mapping

SPR			REGISTER NAME	PRIVILEGED	SERIALIZE ACCESS
DECIMAL	SPR 5:9	SPR 0:4			
268	01000	01100	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. Additional Special-Purpose Registers

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

Table 6-9. Additional 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 Read: Synch Relative to Load/Store Operations
156	00100	11100	LCTRL1	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
157	00100	11101	LCTRL2	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
158	00100	11110	ICTRL	Debug	Fetch Sync on Write
159	00100	11111	BAR	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
630	10011	10110	DPDR	Debug	Read and Write
631	10011	10111	DPIR[4]	Yes	Fetch
638	10011	11110	IMMR	Yes	Write - As a Store
560	10001	10000	IC_CST	Yes	Write - As a Store
561	10001	10001	IC_ADR	Yes	Write - As a Store
562	10001	10010	IC_DAT	Yes	Write - As a Store
568	10001	11000	DC_CST	Yes	Write - As a Store
569	10001	11001	DC_ADR	Yes	Write - As a Store
570	10001	11010	DC_DAT	Yes	Write - As a Store
784	11000	10000	MI_CTR	Yes	Write - As a Store
786	11000	10010	MI_AP	Yes	Write - As a Store
787	11000	10011	MI_EPN	Yes	Write - As a Store
789	11000	10101	MI_TWC (MI_L1DL2P)	Yes	Write - As a Store
790	11000	10110	MI_RPN	Yes	Write - As a Store
816	11001	10000	MI_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
794	11000	11010	MD_AP	Yes	Write - As a Store

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

SPR			REGISTER NAME	PRIVILEGED	SERIALIZE ACCESS
DECIMAL	SPR 5:9	SPR 0:4			
795	11000	11011	MD_EPN	Yes	Write - As a Store
796	11000	11100	M_TWB (MD_L1P)	Yes	Write - As a Store
797	11000	11101	MD_TWC (MD_L1DL2P)	Yes	Write - As a Store
798	11000	11110	MD_RPN	Yes	Write - As a Store
799	11000	11111	M_TW (M_SAVE)	Yes	Write - As a Store
824	11001	11000	MD_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 6.2.4.1 Restartability After An Interrupt**.
2. Refer to **Section 18.3.3.3 Development Port Serial Communications**.
3. Protection of Registers designated with “debug” privilege is described in **Section 18.5.1 Protecting the Development Port Registers**.
4. This register is a fetch-only register, using **mtspr** is ignored and using **mfspr** gives an undefined value.

Table 6-10. Other Control Registers

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

6.3.1.1 PHYSICAL LOCATION OF SPECIAL REGISTERS

Some special registers 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** and **mfspr** instructions through the internal chip buses. Apart from the PowerPC timebase counter and decrementer that are implemented within the system interface unit, in the current implementation the following encoding is reserved for special registers not located within the core.

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

SPR		RESERVED FOR
SPR $_{5:9}$	SPR $_{0:4}$	
100xx 110xx	xxxxx	Core Registers External to the Core
1x0xx	x0xxx	Reserved
10011	x0xxx	System Interface Unit Internal Registers
0xxxx	xxxxx	If Appears on the Internal Bus Signifies Decrementer or Timebase
10000	x0xxx	Reserved
10000	x1xxx	Reserved
1100x	x0xxx	Instruction MMU Implementation Specific Control
1100x	x1xxx	Data MMU Implementation Specific Control
10001	x00xx	Instruction Cache Registers
10001	x10xx	Data Cache Registers

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

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

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

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

6.3.1.2 BIT ASSIGNMENT OF THE CONTROL REGISTERS

6.3.1.2.1 Machine State Register. The 32-bit machine state register (MSR) defines the state of the processor. It can be read by the **mfmsr** instruction.

MSR

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

Bits 0–12—Reserved

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

POW—Power Management Enable

Power management may not exist on all implementations. If it is not implemented, this bit is reserved.

0 = Power management is disabled (normal operation mode).
1 = Power management is enabled (reduced power mode).

Bit 14—Reserved

This bit is reserved and should be set to 0.

ILE—Interrupt Little-Endian Mode

When an interrupt occurs, this bit is copied to MSR_{LE} to select the endian mode for the context established by the interrupt.

EE—External Interrupt Enable

0 = The processor is disabled against external and decrementer interrupts.
1 = The processor is enabled to take an external or decrementer interrupt.

PR—Problem State

0 = The processor is privileged to execute any instruction.
1 = The processor can only execute the nonprevileged instructions.

FP—Floating-Point Available

0 = The processor cannot execute any floating-point instructions, including floating-point loads, stores, or moves.
1 = The processor can execute floating-point instructions.

ME—Machine Check Enable

0 = Machine check interrupts are disabled.
1 = Machine check interrupts are enabled.

FE0—Floating-Point Exception Mode 0

0 = Ignore Exceptions mode.
0 = Imprecise Nonrecoverable mode.
1 = Imprecise Recoverable mode.
1 = Precise mode.

SE—Single-Step Trace Enable

This bit controls the optional Trace interrupt. If the Trace interrupt is not implemented, this bit is reserved.

BE—Branch Trace Enable

This bit controls the optional Trace interrupt. If the Trace interrupt is not implemented, this bit is reserved.

FE1—Floating-Point Exception Mode 1

0 = Ignore Exceptions mode.
1 = Imprecise Nonrecoverable mode.
0 = Imprecise Recoverable mode.
1 = Precise mode.

Bit 24—Reserved

This bit is reserved and should be set to 0.

IP—Interrupt Prefix

In the following description, nnnn is the offset of the interrupt:

0 = Interrupts are vectored to the real address 0x000n_nnnn in 32-bit implementations and real address 0x0000_0000_000n_nnnn in 64-bit implementations.
1 = Interrupts are vectored to the real address 0xFFFn_nnnn in 32-bit implementations and real address 0xFFFF_FFFF_FFFn_nnnn in 64-bit implementations.

IR—Instruction Relocate

0 = Instruction address translation is off.
1 = Instruction address translation is on.

DR—Data Relocate

0 = Data translation is off.
1 = Data translation is on.

Bits 28–29—Reserved

These bits are reserved and should be set to 0.

RI—Recoverable Interrupt

0 = Interrupt is not recoverable.
1 = Interrupt is recoverable.

LE—Little-Endian Mode

0 = The processor runs in big-endian mode.
1 = The runs in little-endian mode.

6.3.1.2.2 Condition Register. The 32-bit condition register (CR) reflects the result of certain operations and provides a mechanism for testing and branching. The bits in this register are grouped into eight 4-bit fields.

BIT	0	1	2	3	4	5	6	7
FIELD	CR0	CR1	CR2	CR3	CR4	CR5	CR6	CR7

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

- Specified fields of the condition register can be set by a move instruction (**mtcrf**) from a general-purpose register.
- A specified field of the condition register can be copied to the field by the **mcrxr** instruction.
- A specified field of the fixed-point exception cause register can be copied to the condition register by the **mcrfs** instruction.
- Condition register logical instructions can be used to perform logical operations on the specified bits in the condition register.
- CR0 can be the implicit result of an integer instruction.
- CR1 can be the implicit result of a floating-point instruction.
- A specified CR field can indicate the result of either an integer or floating-point compare instruction.

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

XER

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

SO—Summary Overflow

This bit is set whenever an instruction (except **mtspr**) sets the OV bit. Once set, the SO bit remains set until it is cleared by an **mtspr** or **mcrxr** instruction. It is not altered by compare instructions or other instructions that cannot overflow. Executing an **mtspr** instruction to the XER, supplying the values zero for SO and one for OV, causes SO to be cleared and OV to be set.

OV—Overflow

This bit is set to indicate that an overflow has occurred during the execution of an instruction. Add, subtract from, and negate instructions have OE = 1 set the OV bit if the carry out of the MSB is not equal to the carry out of the MSB + 1, and clear it otherwise. Multiply low and divide instructions have OE = 1 set the OV bit if the result cannot be represented in 64 bits (**mulid**, **divd**, **divdu**) or in 32 bits (**mulw**, **divw**, **divwu**) and clear it otherwise. The OV bit is not altered by compare instructions that cannot overflow (except **mtspr** to the XER and **mcrxr**).

CA—Carry

This bit is set during the execution of the following instructions:

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

The CA bit is not altered by compare instructions or by other instructions that cannot carry (except shift right algebraic, **mtspr** to the XER, and **mcrxr**).

Bits 3–24—Reserved

These bits are reserved and should be set to 0.

BCNT—Byte Count for Load/Store String Operations

These bits specify the number of bytes to be transferred by a **lswx** or **stswx** instruction.

6.3.1.3 INITIALIZING THE CONTROL REGISTERS

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

6.3.1.3.2 Hard/Soft Reset. When a hard or soft reset occurs, the registers affected by system reset are set similar to the way the system reset interrupt is set. The following list shows 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).

6.4 THE FIXED-POINT UNIT

The fixed-point unit implements all fixed-point processor instructions, except the fixed-point storage access instructions that are implemented by the load/store unit. See the *PowerPC Family: The Programming Environment (MPCFPE/D)* manual for more information.

6.4.1 Updating XER with Divide Instructions

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

6.5 THE LOAD/STORE UNIT

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

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

- Supports many instructions
- 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 and its two queues. The address queue is a 2-entry queue shared by all load/store instructions and the fixed-point data queue is a 2-entry, 32-bit wide queue that holds fixed-point data. The load/store unit has a dedicated writeback bus so that loaded data received from the internal bus is written directly back to the fixed-point or floating-point register files.

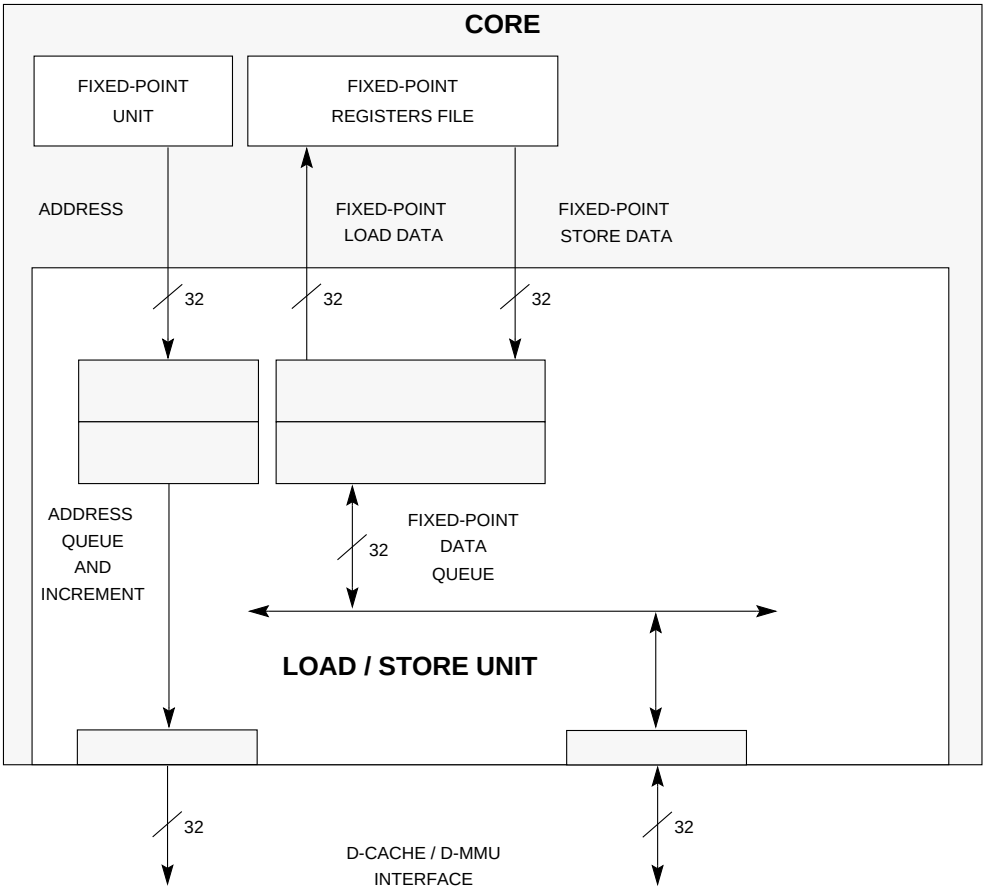


Figure 6-6. Load/Store Unit Functional Block Diagram

To execute the multiple 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.

6.5.1 Load/Store Instruction

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

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

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

6.5.2 Synchronizing Load/Store Instructions

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

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

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

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

6.5.3 Instructions Issued to the Data Cache

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

6.5.4 Issuing Store Instruction

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

6.5.5 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, 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.

6.5.6 Executing Unaligned Instructions

The load/store unit supports fixed-point unaligned accesses in the hardware. The 32-bit L-bus only supports naturally aligned transfers. 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 illustrates the number of bus cycles needed to execute unaligned instructions.

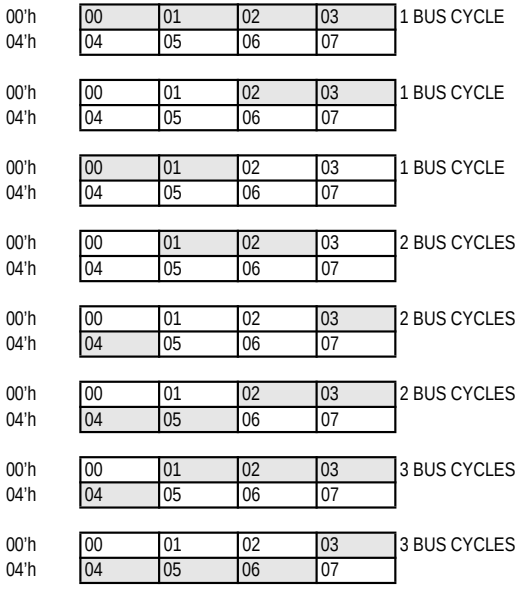


Figure 6-7. Number of Bus Cycles Needed For Unaligned, Single Register Fixed-Point Load/Store Instructions

6.5.7 Little-Endian Mode Support

The load/store unit implements little-endian mode in which the modified address is issued to the data cache. When an individual scalar unaligned transfer or the execution of a multiple/string instruction occurs, an alignment exception is generated.

6.5.8 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 relates to special internal bus snoop logic. This logic is implemented in the data cache.

When a **lwarx** instruction is executed the load/store unit issues a cycle to the data cache with a special attribute. In the 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 to see if a reservation was lost. If loss of reservation has occurred, the cycle is blocked from going to the external bus and the external bus interface module notifies the load/store unit of a **stwcx** failure.

The MPC801 storage reservation supplies hooks for the support of storage reservation implementation in a hierarchical bus structure. Refer to **Section 13.4.9 Storage Reservation Protocol** 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.

6.5.9 Instruction Timing

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

Table 6-13. 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.

6

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

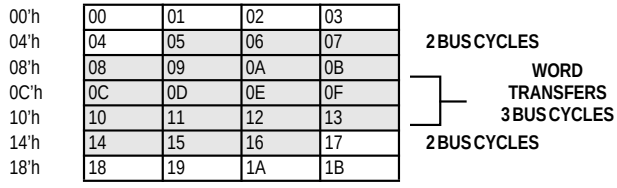


Figure 6-8. Number of Bus Cycles Needed For String Instruction Execution

6.5.10 Stalling Storage Control Instructions

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

6.5.11 Accessing Off-Core Special Registers

Access to special registers that are implemented off-core is executed by the load/store unit via the internal bus using a special cycle. Refer to **Section 6.3.1.1 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 are previously synchronized. In other words, the instruction is not taken until all prior instructions terminate.

6.5.12 Storage Control Instructions

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

6.5.13 Exception Processing

An exception is a special condition that preempts normal processing. Exception processing is the transition from normal mode program execution to the execution of a routine that deals with an exception.

6.5.13.1 USING DAR, DSISR, AND BAR

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

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

Table 6-14. DAR, BAR, and DSISR Value 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 on the PowerPC™ architecture as defined in the *PowerPC Architecture Books I, II, and III*. It also describes features that exist in the architecture, but are not supported by the core. The information in this section is based on the PowerPC books, but you should refer to the *PowerPC Family: The Programming Environment (MPCFPE/D)* manual for more information.

7.1 POWERPC USER INSTRUCTION SET ARCHITECTURE (BOOK I)

7.1.1 Computation Modes

The core is a 32-bit fixed-point implementation of the PowerPC architecture. Any reference to 64-bit implementations in the *PowerPC Architecture* is not supported by the core. In addition, no floating point of the architecture is implemented.

7.1.2 Reserved Fields

Reserved fields in instructions are described under the specific instruction definition sections. Unless otherwise stated in the specific instruction description, fields marked *I*, *II*, and *III* in the instruction are discarded by the core decoding. Thus, this type of invalid form instructions yield results of the defined instructions with the appropriate field zero.

In most cases, the reserved fields in registers are ignored on write and return zeros for them on read for any control register implemented by the core. Exceptions to this rule are bits 16:23 of the fixed-point exception cause register (XER) and the reserved bits of the machine state register (MSR), which are set by the source value on write and return the value last set for it on read.

7.1.3 Classes of Instructions

Nonoptional instructions (except floating-point load, store, and compute instructions) are implemented by the hardware. Optional instructions are executed by implementation dependent code and any attempt to execute one of these commands causes the core to take the implementation dependent software emulation interrupt (offset x'01000' of the vector table).

Illegal and reserved instruction class instructions are supported by implementation dependent code and, thus, the core hardware generates the implementation dependent software emulation interrupt. The way the core treats invalid and preferred instruction forms is described in the specific processor compliance sections.

7.1.4 Exceptions

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

7.1.5 The Branch Processor

7.1.6 Fetching Instructions

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

7.1.7 Branch Instructions

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

7.1.7.1 INVALID BRANCH INSTRUCTION FORMS

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

7.1.7.2 BRANCH PREDICTION

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

7.1.8 The Fixed-Point Processor

The core implements the following fixed-point instructions:

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

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

7.1.8.0.1 Move To/From System Register Instructions. Move to/from invalid special registers in which **spr0** =1 invokes the privilege instruction error interrupt handler if the processor is in problem state. For a list of all implemented special registers, refer to **Section 6.3.1 The Control Registers**.

7.1.8.0.2 Fixed-Point Arithmetic Instructions. Attempting to perform any of the following divisions in the **divw[o][.]** instruction

0x80000000 ÷ -1

<anything> ÷ 0

causes the contents of RT to be 0x80000000 and if RC =1, the contents of the bits in the CR field 0 are LT = 1, GT = 0, EQ = 0, and SO is set to the correct value. 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 the bits in the CR field 0 are LT = 1, GT = 0, EQ = 0, and SO is set to the correct value. In the **cmpi**, **cmp**, **cmpli**, and **cmpl** instructions, the L bit is applicable for 64-bit implementations and if L = 1 the instruction form is invalid. The core ignores this bit so the behavior when L = 1 is identical to the valid form instruction with L = 0.

7.1.9 The Load/Store Processor

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

7.1.9.1 FIXED-POINT LOAD 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, the instruction completes normally. RA is then loaded from the memory location as follows:

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

7.1.9.3 FIXED-POINT LOAD STRING INSTRUCTIONS

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

7.1.9.4 STORAGE SYNCHRONIZATION INSTRUCTIONS

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

7.1.9.5 OPTIONAL INSTRUCTIONS

No optional instructions are supported.

7.1.9.6 LITTLE-ENDIAN BYTE ORDERING

The load/store unit supports little-endian byte ordering as specified in *PowerPC User Instruction Set Architecture (Book I)*. In little-endian mode, attempting to execute an individual scalar unaligned transfer, as well as a multiple or string instruction, causes an alignment interrupt.

7.2 POWERPC VIRTUAL ENVIRONMENT ARCHITECTURE (BOOK II)

7.2.1 Storage Model

The MPC801 instruction and data caches are defined as follows:

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

7.2.1.1 MEMORY COHERENCE

Hardware memory coherence is not supported in the MPC801 hardware, but can be performed in the software or by defining storage as cache inhibited as needed. In addition, the MPC801 does not provide any data storage attributes for an external system.

7.2.1.2 ATOMIC UPDATE PRIMITIVES

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

The MPC801 does not cause the system data storage error handler to be invoked if the storage accessed by the **lwarx** and **stwcx** instructions is in the writethrough required mode. Also, the MPC801 does not provide support for snooping an external bus activity outside of the chip. The provision is made to cancel the reservation inside the MPC801 by using the $\overline{CR}$ and $\overline{KR}$ input pins.

7.2.2 The Effect of Operand Placement on Performance

The load/store unit hardware supports all of the PowerPC load/store instructions. An optimal performance can be obtained for naturally aligned operands. These accesses result in optimal performance (one bus cycle) for a maximum size of 4 bytes and good performance (two bus cycles) for double precision floating-point operands. Unaligned operands are supported in the hardware and are broken into a series of aligned transfers. The effect of operand placement on performance is as stated in *PowerPC Virtual Environment Architecture Book II,* except for the case of 8-byte operands.

Since the MPC801 uses a 32-bit wide data bus, the performance is good, rather than optimal. Refer to **Section 6.5.6 Executing Unaligned Instructions** for a description of fixed-point unaligned instruction execution and timing and to **Section 6.5.9 Instruction Timing** for a description of string instruction timing.

7.2.3 The Storage Control Instructions

The MPC801 interprets the cache control instructions—**icbi**, **isync**, **dcbt**, **dcbi**, **dcbf**, **dcbz**, **dcbst**, **eieio**, and **dcbtst**—as if they pertain only to the MPC801 cache. These instructions do not broadcast. Any bus activity caused by these instructions is a direct result of performing the operation on the MPC801 cache.

7.2.3.1 INSTRUCTION CACHE BLOCK INVALIDATE (icbi)

The effective address is translated by the memory management unit (according to the MSR_{IR}) and the associative block in the instruction cache is invalidated if hit.

7.2.3.2 INSTRUCTION SYNCHRONIZE (isync)

The **isync** instruction waits for all previous instructions to complete and then discards any prefetched instructions, thus causing the subsequent instruction to be fetched or refetched from memory and executed.

7.2.3.3 DATA CACHE BLOCK TOUCH (dcbt)

The block associated with this instruction is checked for hit in the cache. If it is a miss, the instruction is treated as a regular miss, except that the bus error does not cause an interrupt. If no error occurs, the line is written into the cache.

7.2.3.4 DATA CACHE BLOCK TOUCH FOR STORE (dcbtst)

The block associated with this instruction is checked for a hit in the cache. If it is a miss, the instruction is treated as a regular miss, except that bus error does not cause an interrupt. If no error occurs, the line is written into the cache.

7.2.3.5 DATA CACHE BLOCK SET TO ZERO (dcbz)

This instruction is executed according to the definition in *PowerPC Virtual Environment Architecture Book II*.

7.2.3.6 DATA CACHE BLOCK STORE (dcbst)

This instruction is executed according to the definition in *PowerPC Virtual Environment Architecture Book II*.

7.2.3.7 DATA CACHE BLOCK INVALIDATE (dcbi)

The effective address is translated by the memory management unit (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)

This instruction is executed according to the definition in *PowerPC Virtual Environment Architecture Book II*.

7.2.3.9 ENFORCE IN-ORDER EXECUTION OF I/O (eieio)

When executing an **eieio** instruction, the load/store unit waits until all previous accesses have terminated before issuing cycles related to load/store instructions that follow **eieio**.

7.2.4 Timebase

A description of the timebase register can 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 MPC801 has an internal memory space that includes memory-mapped control registers and memory that is used by various modules on the chip. This memory is part of the main memory as seen by the core but cannot be accessed by any external system master.

7.3.1 The Branch Processor

7.3.1.1 BRANCH PROCESSOR REGISTERS

The branch processor registers consist of the machine state and processor version registers.

7.3.1.1.1 Machine State Register. The floating-point exception mode is ignored by the MPC801. 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. In the processor version register (PVR), the value of the version field is x'0050'. In addition, the value of the revision field is x'0000' and it is incremented each time the software distinguishes between the core revisions.

7.3.1.2 BRANCH PROCESSOR INSTRUCTIONS

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

7.3.2 The Fixed-Point Processor

7.3.2.1 SPECIAL-PURPOSE REGISTERS

The special-purpose registers consist of the unsupported and added registers.

7.3.2.1.1 Unsupported Registers. The following registers are not supported by the MPC801. Refer to **Section 7.3.3 Storage Model** for more details.

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

7.3.2.1.2 Added Registers. For a list of the added special-purpose registers, see Table 6-9.

7.3.3 Storage Model

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

7.3.3.1 ADDRESS TRANSLATION

If address translation is disabled (MSR_{IR} =0 for instruction accesses or MSR_{DR} =0 for data accesses), the effective address is treated as the real address and is passed directly to the memory subsystem. Otherwise, the effective address is translated by using the translation lookaside buffer (TLB) mechanism of the memory management unit. 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 memory management unit 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 MPC801. However, the change bit is supported by using the data TLB error interrupt mechanism when writing to an unmodified page.

7.3.5 Storage Protection

Two main protection modes are supported by the MPC801:

- Domain manager mode
- PowerPC mode

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

7.3.6 Storage Control Instructions

7.3.6.1 DATA CACHE BLOCK INVALIDATE (dcbi)

This instruction is executed according to the definition in *PowerPC Operating Environment Architecture Book III*.

7.3.6.2 TLB INVALIDATE ENTRY (tlbie)

This instruction is performed as defined by the architecture, except that the 22 most-significant bits of the EA are used for address compare.

7.3.6.3 TLB INVALIDATE ALL (tlbia)

This instruction is performed as defined by the architecture.

7.3.6.4 TLB SYNCHRONIZE (tlbsync)

This instruction is implemented like and functions like a regular **mtspr** instruction as it relates to engine synchronization with no further effects.

7.3.7 Interrupts

7.3.7.1 CLASSES

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

7.3.7.2 PROCESSING

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

7.3.7.3 DEFINITIONS

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

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

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

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

OFFSET (HEX)	INTERRUPT TYPE
01100	Implementation Dependent Instruction TLB Miss
01200	Implementation Dependent Data TLB Miss
01300	Implementation Dependent Instruction TLB Error
01400	Implementation Dependent Data TLB Error
01500 - 01BFF	Reserved
01C00	Implementation Dependent Data Breakpoint
01D00	Implementation Dependent Instruction Breakpoint
01E00	Implementation Dependent Peripheral Breakpoint
01F00	Implementation Dependent Nonmaskable Development Port

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

SRR0—Save/Restore Register 0

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

SRR1—Save/Restore Register 1

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

MSR—Machine State Register

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

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

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

As defined in *PowerPC Operating Environment Architecture Book III*, machine check interrupts are enabled when MSR_{ME} =1. If MSR_{ME} = 0 and a machine check interrupt indication is received, the processor enters the checkstop state. The behavior of the core in checkstop state is dependent on the working mode as defined in **Section 18.3.1.2 Debug Mode Enable vs. Debug Mode Disable**. When the processor is in debug mode enable, it enters the debug mode instead of the checkstop state. When in debug mode disable, instruction processing is suspended and cannot be restarted without resetting the core.

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

SRR0—Save/Restore Register 0

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

SRR1—Save/Restore Register 1

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

MSR—Machine State Register

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

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

DSISR—Data/Storage Interrupt Status Register

0–14	Set to 0.
15–16	Set to bits 29–30 of the instruction if X-form 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.

DAR—Data Address Register

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

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

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

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

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

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

7.3.7.3.6 Program Interrupt. A floating-point enabled exception type program interrupt is not generated by the MPC801. Likewise, an illegal instruction type program interrupt is not generated by the core, but an implementation dependent software emulation interrupt is generated instead. A privileged instruction program interrupt is generated for an on-core valid special-purpose register (SPR) field or any SPR encoded as an external special register if $SPR_0 = 1$ and $MSR_{PR} = 1$, as well as if 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 MPC801. 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 if $MSR_{BE} = 1$ and a branch is completed. Notice that the trace interrupt does not occur after an instruction that causes an interrupt. The monitor/debugger software must change the vectors of other possible interrupt addresses to single-step these instructions. If this is unacceptable, other debug features can be used. Refer to **Section 18 Development Support** for more information. The following registers are set:

SRR0—Save/Restore Register 0

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

SRR1—Save/Restore Register 1

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

MSR—Machine State Register

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

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

7.3.7.3.9 Floating-Point Assist Interrupt. The floating-point assist interrupt is not generated by the MPC801. An implementation dependent software emulation interrupt will be taken on any attempt to execute a floating-point instruction.

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

- When executing any unimplemented instruction, including all illegal and unimplemented optional and floating-point instructions.
- When executing a **mtspr** or **mfspr** that specifies an on-core unimplemented register, regardless of SPR_0.
- When executing a **mtspr** or **mfspr** that specifies an off-core unimplemented register and SPR_0 =0 or MSR_{PR} =0. Refer to **Section 7.3.7.3.6 Program Interrupt** for more information.

In addition, the following registers are set:

SRR0—Save/Restore Register 0

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

SRR1—Save/Restore Register 1

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

MSR—Machine State Register

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

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

7.3.7.3.11 Implementation Specific Instruction TLB Miss Interrupt. This type of interrupt occurs if MSR_{IR} =1 and 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–Save/Restore Register 0

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

SRR1—Save/Restore Register 1

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

MSR—Machine State Register

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

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

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

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

The following registers are set:

SRR0—Save/Restore Register 0

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

SRR1—Save/Restore Register 1

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

MSR—Machine State Register

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

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

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

SRR0—Save/Restore Register 0

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

SRR1—Save/Restore Register 1

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

MSR—Machine State Register

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

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

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

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

The following registers are set:

SRR0—Save/Restore Register 0

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

SRR1—Save/Restore Register 1

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

MSR—Machine State Register

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

DSISR—Data/Storage Interrupt Status Register

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

DAR—Data Address Register

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

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

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

- When there is an internal breakpoint match. For more details, refer to **Section 18.2 Watchpoint And Breakpoint Generation**.
- 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 18 Development Support** for details on how to generate the development port request.

The following registers are set:

SRR0—Save/Restore Register 0

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

SRR1—Save/Restore Register 1

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

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

MSR—Machine State Register

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

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

BAR—Breakpoint Address Register

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

DSISR—Data/Storage Interrupt Status Register

Do not change.

DAR—Data Address Register

Do not change.

Execution resumes at offset from the base address that MSR_{IP} indicates:

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

7.3.7.4 PARTIALLY EXECUTED INSTRUCTIONS

In general, the architecture allows instructions to be partially executed when an alignment or data storage interrupt occurs. In the core, instructions are not executed if an alignment interrupt condition is detected or if a data storage interrupt is never generated by the hardware. In the MPC801, the instruction can be partially executed only when load/store instructions occur that cause multiple access to the memory subsystem—multiple/string and unaligned load/store instructions.

In this instance, the instruction can be partially completed if one of the accesses (except the first one) causes a miss in the data TLB. The implementation specific data TLB miss interrupt is taken in this case. For the update forms, the update register 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 that are not discussed here are not implemented by the MPC801 hardware. Any attempt to execute any of these instructions causes an implementation dependent software emulation interrupt to occur.

SECTION 8
INSTRUCTION EXECUTION TIMING

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

Table 8-1. Instruction Execution Timing

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

Table 8-1. Instruction Execution Timing (Continued)

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

Table 8-1. Instruction Execution Timing (Continued)

INSTRUCTIONS	LATENCY	BLOCKAGE	EXECUTION UNIT	SERIALIZING INSTRUCTION
Move to / from Special-Purpose Register mtspr, mfspr	Serialize + 1	Serialize + 1	LDST	Yes (Before)
String Instructions: lswi, lswx, stswi, stswx	Serialize + 1 + Number of Words Accessed	Serialize + 1 + Number of Words Accessed	LDST	Yes
Storage Control Instructions: isync	Serialize	Serialize	Branch	Yes
Order Storage Access: eieio	1	1	LDST	Next Load or Store is Synchronized Relative to All Prior Load or Store
Cache Control: icbi	1	1	LDST, I-Cache	No

NOTES:
1. See Table 6-11 for more information.
2. Refer to **Section 6.3.1 The Control Registers**.
3. See Table 6-10.
4. $$DivisionLatency = \begin{bmatrix} NoOverflow \Rightarrow 3 + \left(\frac{34 - divisorLength}{4}\right) \\ Overflow \Rightarrow 2 \end{bmatrix}$$

   Where:

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

5. $DivisionBlockage = DivisionLatency$
6. Blocking the multiply instruction is dependent on the subsequent instruction. For any subsequent multiply instruction, the blockage is 1 clock and for any subsequent divide it is 2 clocks.
7. Assuming nonspeculative aligned access, on-chip memory, and available bus. For details, refer to **Section 6.5.5 Nonspeculative Load Instructions**, **Section 6.5.6 Executing Unaligned Instructions**, and **Section 6.5.9 Instruction Timing**.
8. Although a store (as well as **mtspr** for special registers external to the core) issued to the load/store unit buffer frees the core pipeline, the next load or store will not actually be performed on the bus until the bus is free.

8.1 INSTRUCTION EXECUTION TIMING EXAMPLES

All examples assume an instruction cache hit.

8.1.1 Data Cache Load

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

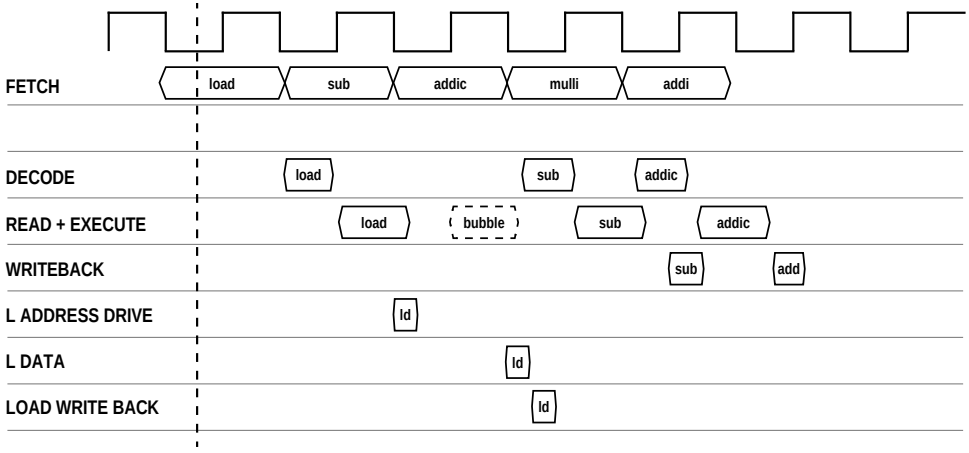


Figure 8-1. Example of a Data Cache Load

This is an example from a data cache with zero wait states. The **sub** instruction is dependent on the value loaded by the load to r12. This causes a bubble to occur in the instruction stream as shown in the execute line. Refer to **Section 8.1.2.2 Private Writeback Bus Dedicated for Load** for instances in which no such dependency exists.

8.1.2 Writeback

8.1.2.1 WRITEBACK ARBITRATION

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

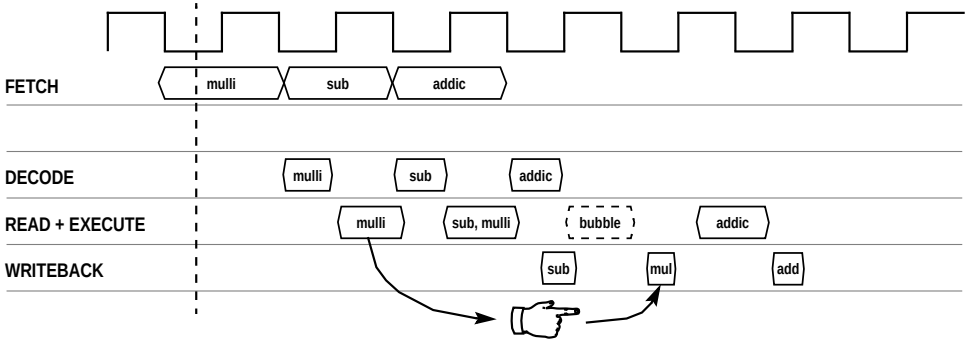


Figure 8-2. Example of a Writeback Arbitration

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

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

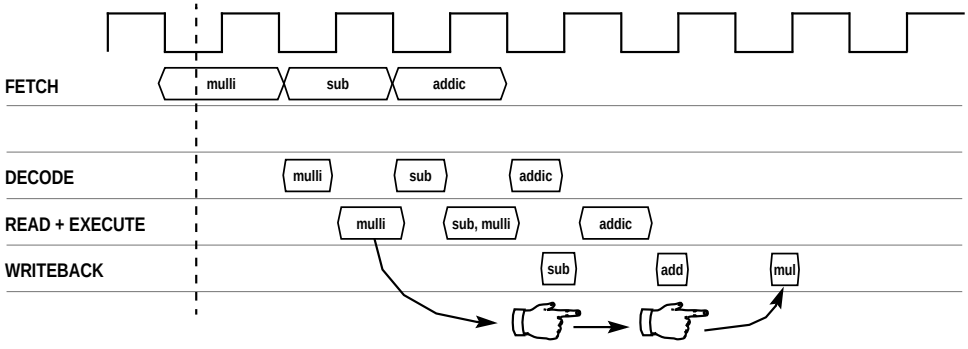


Figure 8-3. Another Example of a Writeback Arbitration

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

8.1.2.2 PRIVATE WRITEBACK BUS DEDICATED FOR LOAD

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

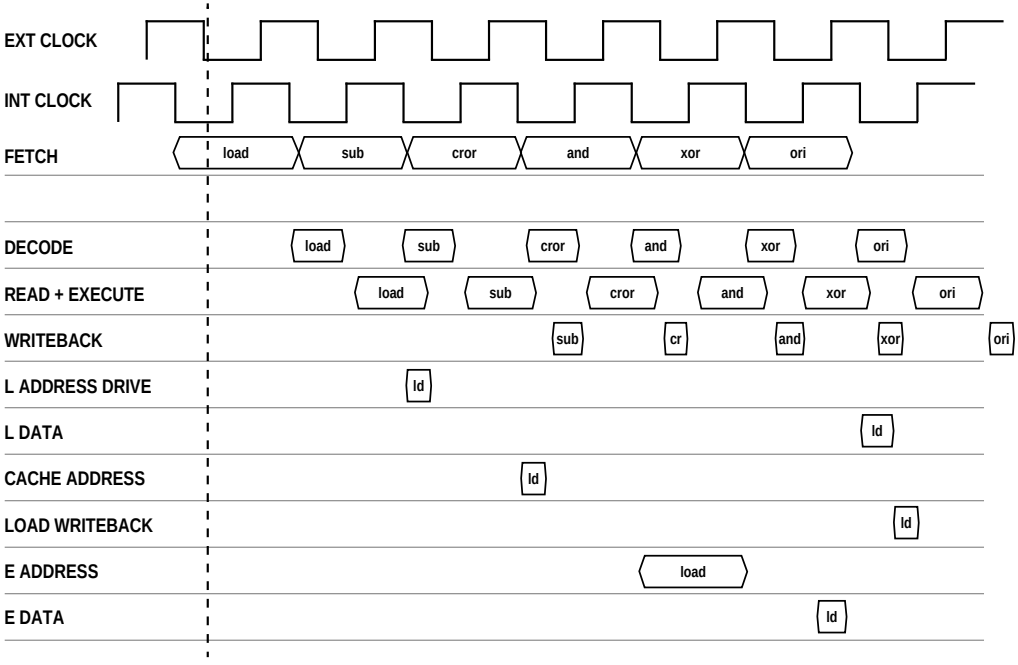


Figure 8-4. Example of a Private Writeback Bus Load

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

8.1.3 Fastest External Load (Data Cache Miss)

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

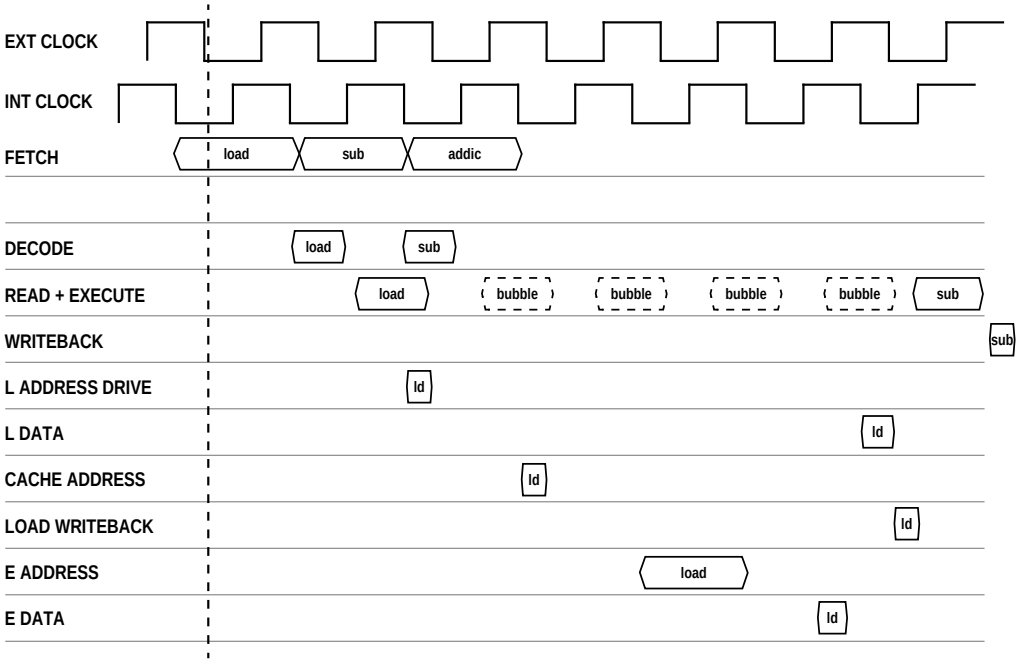


Figure 8-5. Example of an External Load

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

8.1.4 A Full History Buffer

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

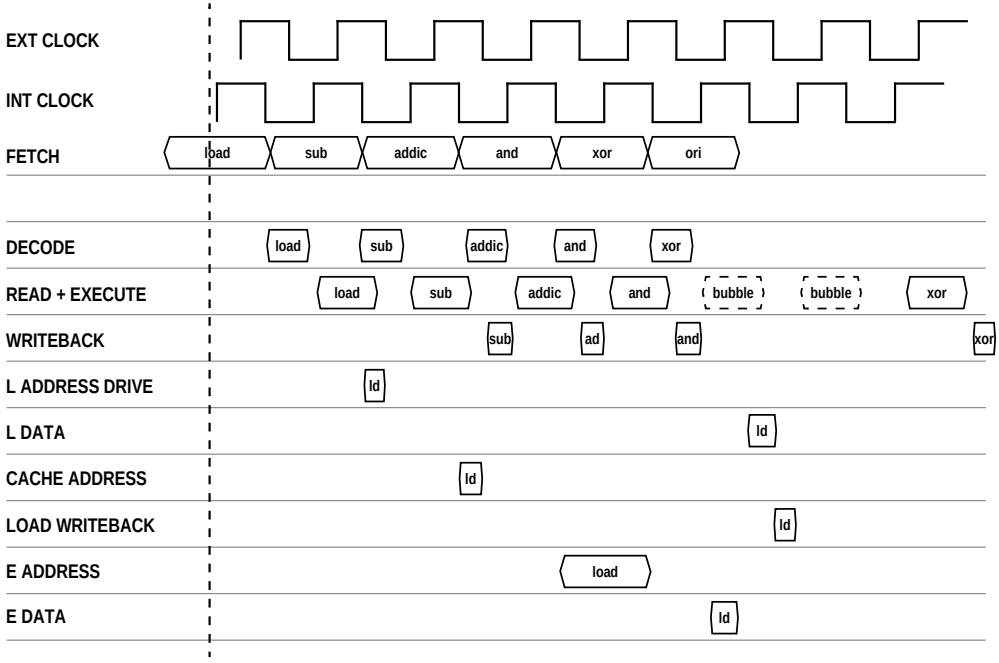


Figure 8-6. Example of a Full History Buffer

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

8.1.5 Branch Folding

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

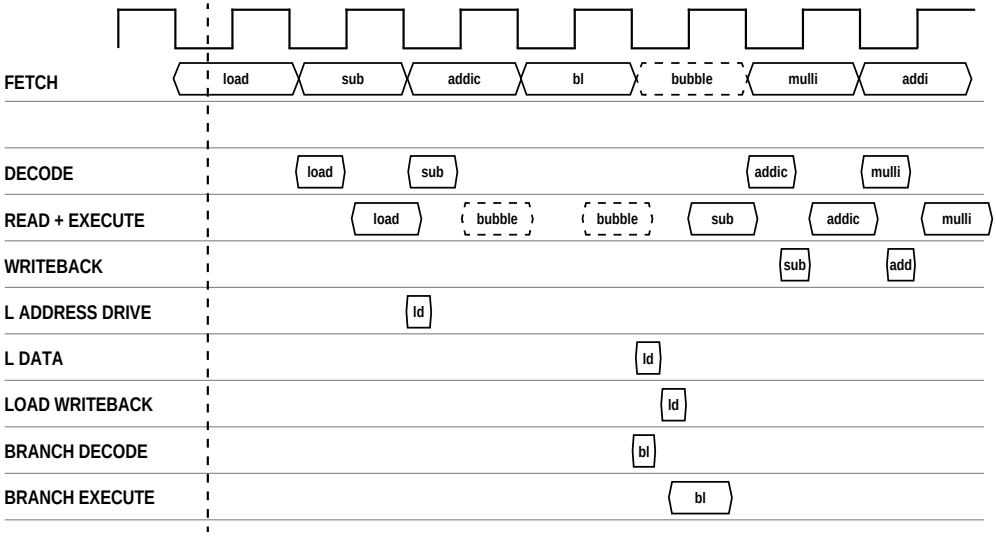


Figure 8-7. Example of Branch Folding

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

8.1.6 Branch Prediction

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

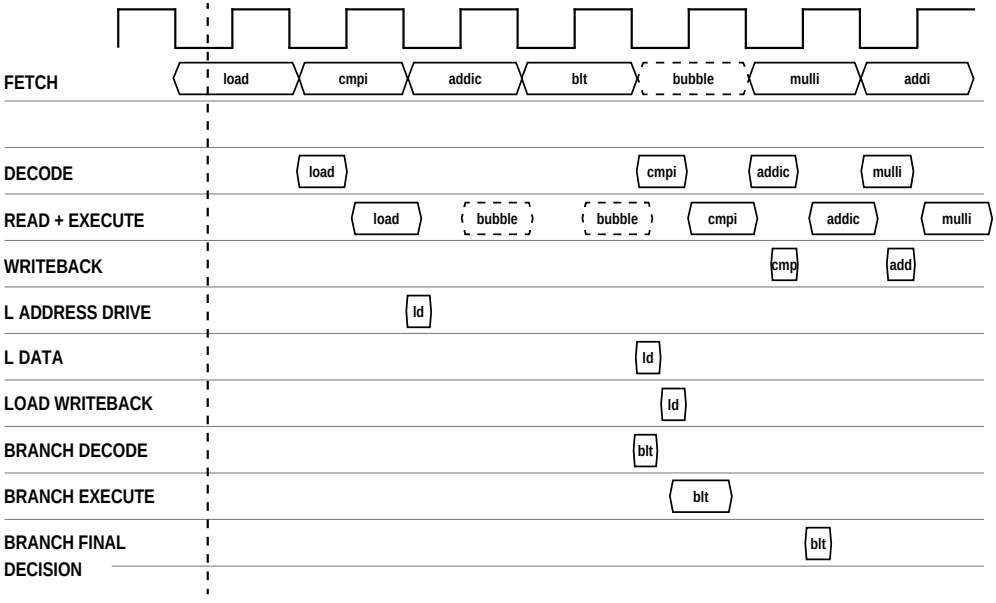


Figure 8-8. Example of Branch Prediction

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

SECTION 9
INSTRUCTION CACHE

The MPC801 instruction cache is a 2K two-way, set-associative cache. It is organized into 64 sets at 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. If a cache hit occurs, the instruction is delivered to the instruction unit and if a cache miss occurs, the cache initiates a burst read cycle on the internal bus with the address of the requested instruction. The first word received from the bus is considered the requested instruction. The cache forwards this instruction to the instruction unit of the core as soon as it is received from the internal bus. A cache line is then selected to receive the data that will be coming from the bus. A least recently used (LRU) replacement algorithm is used to select a line when no empty lines are available.

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

9.1 FEATURES

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

- 2K Two-Way, Set-Associative at Four Words Per Line
- Adheres to the LRU Replacement Policy
- Parked on the Internal Bus
- Lockable SRAM
- "Critical word first", Burst Access
- Responds to Stream Hit, Which Allows Fetching from the Burst Buffer and of the Word Currently on the Internal Bus.
- Serves the Core Request in Parallel to Bringing the Tail of the Previously Missed Line

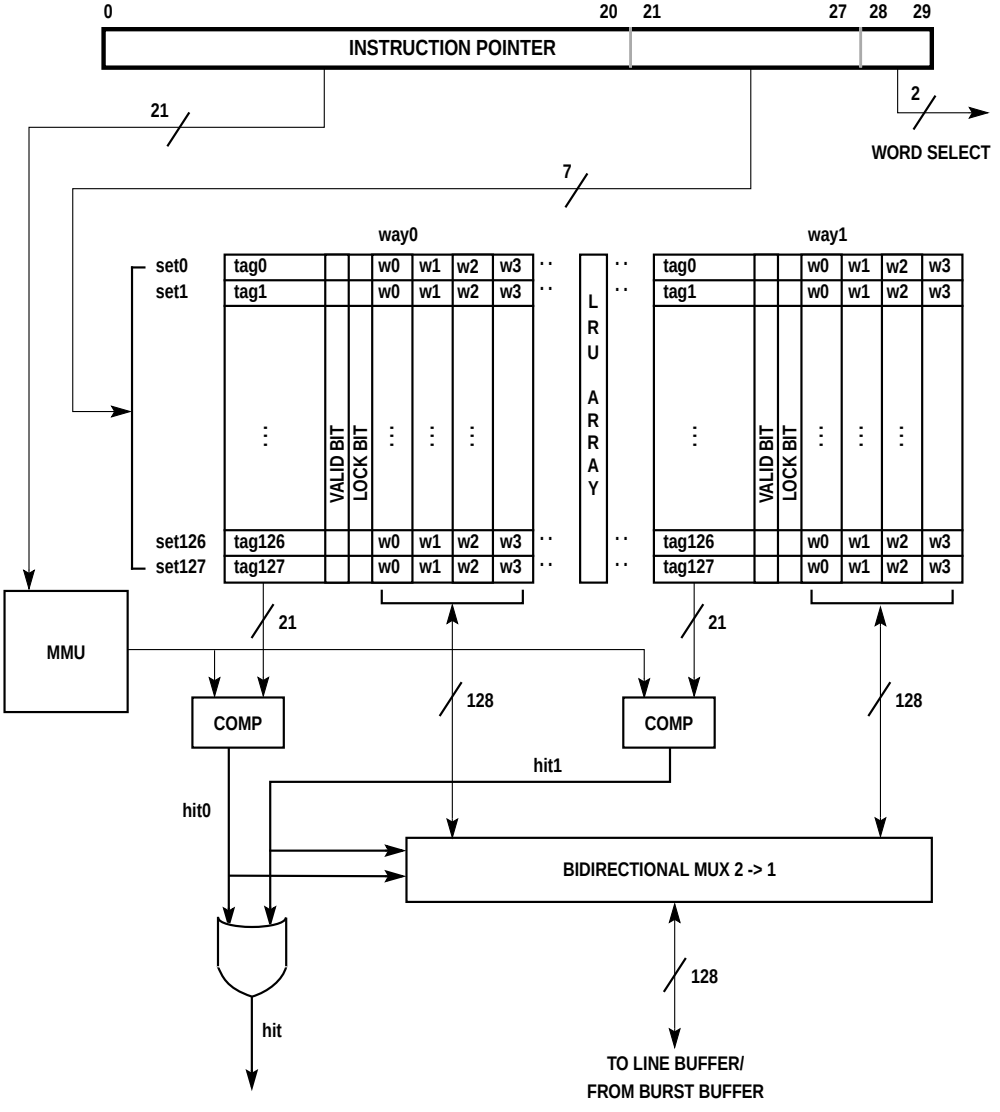


Figure 9-1. Instruction Cache Organization

- Cache Control
- Supports Cache Inhibit
- Efficiently Uses the Pipeline of the Internal Bus by Initiating a New Burst Cycle While Bringing the Tail of the Previously Missed Line into the Cache from Memory
- Performance Enhanced for Cache-Inhibited Regions by Fetching a Full Line to the Internal Burst Buffer. Instructions Stored in the Burst Buffer and Those Originated in a Cache-Inhibited Region are Only Used Once Before Being Refetched.

- Instruction Unit Request has Priority Over a Burst Buffer Write to an Array, thus Increasing the Overall Core Performance
- Miss Latency is Reduced by Sending Addresses to the Cache and Internal Bus Simultaneously and Aborting When a Hit Before a Cycle Goes External
- Minimum Operational Power Consumption
- Tags and All Their Attributes and Data Arrays Have Read/Write Capability
- Special Support is Available When the MPC801 Processor is in Debug Mode.

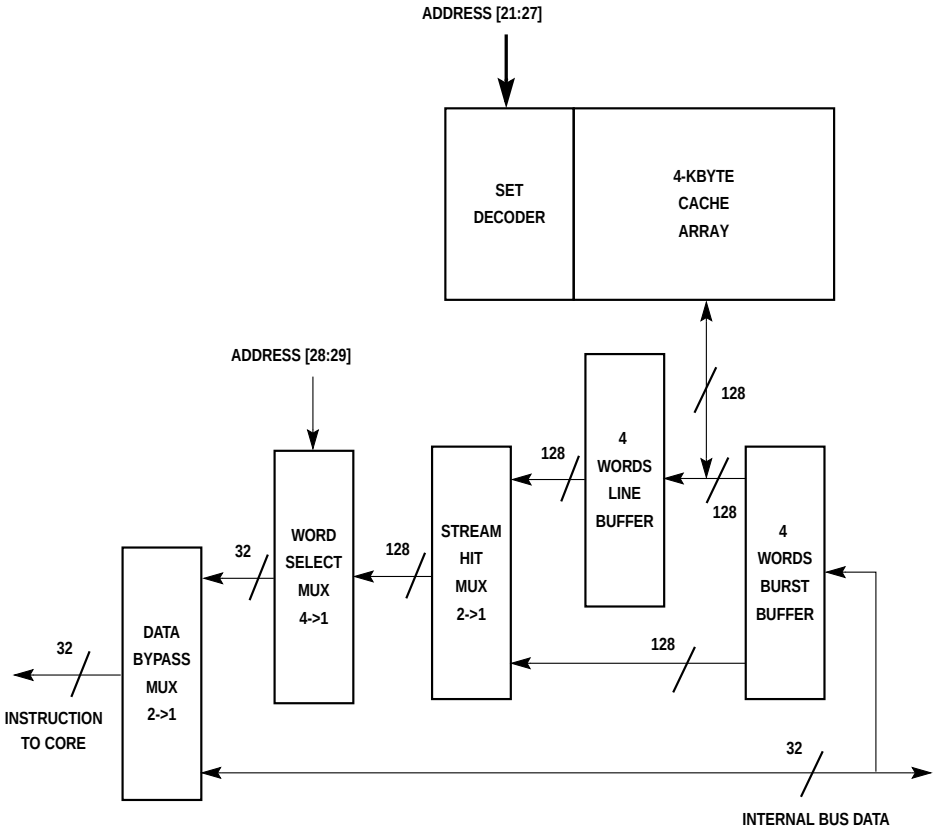


Figure 9-2. Cache Data Path Block Diagram

9.2 PROGRAMMING THE INSTRUCTION CACHE

The instruction cache programming model implements specific operations. All PowerPC special registers can be accessed via the **mfspr** and **mtspr** instructions. Three of these special-purpose registers are used to control the instruction cache:

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

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

9.2.1 Instruction Cache Control and Status Register

Reset value is 0x00000000.

IC_CST

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

IEN—Instruction Cache Enable Status

This bit is read-only. Any attempt to write it is ignored. It reflects the state of the instruction cache.

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

Bits 1–3, 7–9, and 13–31—Reserved

These bits are reserved and should be set to 0.

CMD—Instruction Cache Commands

Reading these bits always delivers a 0.

000 = Reserved.
001 = Cache Enable.
010 = Cache Disable.
011 = Load & Lock.
100 = Unlock Line.
101 = Unlock All.
110 = Invalidate All.
111 = Reserved.

CCER1—Instruction Cache Error Type 1

These bits are sticky and set by the hardware. They are read-only and cleared when read.

0 = No error.
1 = Error.

CCER2—Instruction Cache Error Type 2

These bits are sticky and set by the hardware. They are read-only and cleared when read.

0 = No error.
1 = Error.

CCER3—Instruction Cache Error Type 3

These bits are sticky and set by the hardware. They are read-only and cleared when read.

0 = No error.
1 = Error.

9.2.2 Instruction Cache Address Register

This register is used to read and write arrays for debug and load & lock.

IC_ADR

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

ADR—Address

These bits represent the address to be used in the command programmed in the command and status register. Reset value is undefined.

9.2.3 Instruction Cache Data Port Register

This register is used to read and write arrays for debug and load & lock.

IC_DAT

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

DAT—Data

These bits represent the data received when reading information from the instruction cache. Reset value is undefined.

9.3 HOW THE INSTRUCTION CACHE WORKS

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

9.3.1 Instruction Cache Hit

When a cache hit occurs, bits 28-29 of the instruction address are used to select one word from the cache line whose tag matches bits 0–20 of the instruction's address. The instruction is then immediately transferred to the instruction unit of the core.

9.3.2 Instruction Cache Miss

When an instruction cache miss occurs, the address of the missed instruction is driven onto 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 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 it is not busy with an instruction unit request. If a bus error is encountered on the access to the requested instruction, then a machine check interrupt is taken. If a bus error occurs on any access to other words in the line, then the burst buffer is marked invalid and the line is not written to the array. However, if no bus error is encountered, the burst buffer is marked valid and eventually written to the array.

The page is marked noncacheable in the memory management unit. The line is only written to the burst buffer and not to the cache. Instructions that are stored in the burst buffer and originate in a cache-inhibited memory region, are only used once before they are refetched. Refer to **Section 9.4.7 Instruction Cache Read** for more information.

9.3.3 Instruction Fetch On A Predicted Path

The core allows branch prediction so branches can issue as early as possible. This mechanism allows instruction prefetch to continue while an unresolved branch is being computed and the condition is being evaluated. Instructions fetched after unresolved branches are considered fetched on a predicted path. These instructions may be discarded later if it turns out that the machine has followed the wrong path. To minimize power consumption, the MPC801 instruction cache does not initiate a miss sequence when the instruction is inside a predicted path. The MPC801 instruction cache evaluates fetch requests to see if they are inside a predicted path and if a hit is detected, the requested data is delivered to the core. However, if a cache miss is detected, the miss sequence is usually not initiated until the core finishes branch evaluation.

9.4 INSTRUCTION CACHE COMMANDS

The MPC801 instruction cache supports the PowerPC invalidate instruction with some additional commands that help control the cache and debug the information stored in it. The additional commands are implemented using the three special-purpose control registers mentioned previously in **Section 9.2 Programming the Instruction Cache**. Most of the commands are executed immediately after the control register is written and unable to generate any errors. Therefore, when executing these commands there is no need to check the error status in the IC_CST register.

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

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

9.4.1 Instruction Cache Block Invalidate

The MPC801 implements the **icbi** instruction as if it only pertains to the MPC801 instruction cache. This instruction does not broadcast on the external bus and the MPC801 does not snoop this instruction if it is broadcasted by other masters. This command is not privileged and has no associated error cases. The instruction cache performs this instruction in one clock cycle. To accurately calculate the latency of this instruction, bus latency should be taken into consideration.

9.4.2 Invalidate All Instruction Cache

The invalidate all instruction cache operation is privileged and if you try to use it when the core is in the problem state (MSR_{PR} =1) a program interrupt will occur. When it is invoked and MSR_{PR} =0, all valid lines in the cache, except the lines that are locked, become invalid. As a result of this command, the lines' LRU points to an unlocked way or to way 0 if both lines are unlocked. This last feature is useful when initializing the instruction cache out of reset. For more information, refer to **Section 9.8 Reset Sequence**. To invalidate the whole cache, set the invalidate all command in the IC_CST register. This command has no associated error cases. The instruction cache performs this instruction in one clock cycle. To accurately calculate the latency of this instruction, you should consider bus latency.

9.4.3 Load & Lock

Load & lock is used to lock critical code segments in the instruction cache. This operation is privileged and if you try to use it when the core is in the problem state (MSR_{PR} =1) a program interrupt will occur. Load & lock is performed on a cache line granularity and after a line is locked, it operates as a regular instruction SRAM. It is not replaced during misses and it is not affected by invalidate commands. The hardware correct operation depends on the software to follow the exact steps mentioned in **Section 9.7 Updating Code And Memory Region Attributes**. To load and lock a line, follow these steps:

1. Read the error type bits in the IC_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 to see if the line containing the byte addressed by the IC_ADR is in the cache. If it is a hit, the line is locked and the command terminates with no exception. If it is not, a regular miss sequence is initiated. After the whole line is placed in the cache, the line is locked. You must check the error type bits in the IC_CST register to determine if the load & lock operation completed properly. Keep in mind that the load & lock command can generate two types of errors:

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

9.4.4 Unlock Line

Unlock line is used to unlock previously locked cache lines. This operation is privileged and if you try to use it when the core is in the problem state (MSR_{PR} =1) a program interrupt will occur. Unlock line is performed on a cache line granularity. If the line is found in the cache it is considered a hit, so it is unlocked and operates as a regular valid cache line. If the line is not found in the cache it is considered a miss, so there is no operation and the command terminates without an exception. To unlock a line, follow these steps:

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

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

9.4.5 Unlock All

Unlock all is used to unlock the entire cache. This operation is privileged and if you try to use it when the core is in the problem state (MSR_{PR} =1) a program interrupt will occur. It is performed on all cache lines. If a line is locked, it is unlocked and operates as a regular valid cache line. If a line is not locked or if it is invalid, no operation occurs. To unlock the whole cache, set the unlock all command in the IC_CST register. This command has no associated error cases. The instruction cache performs this instruction in one clock cycle. To accurately calculate the latency of this instruction, you should consider bus latency.

9.4.6 Instruction Cache Inhibit

Two levels of cache inhibit are supported by the MPC801—as a cache mode of operation and on memory regions supported by the memory management unit. To disable the instruction cache, set the cache disable command in the IC_CST register. This operation is privileged and if you try to use it when the core is in the problem state (MSR_{PR} =1) a program interrupt will occur. This command has no error cases that you need to check.

To enable the instruction cache, set the cache enable command in the IC_CST register. This operation is privileged and if you try to use it when the core is in the problem state (MSR_{PR} =1) a program interrupt will occur. This command has no error cases that you need to check. When fetching from cache-inhibited regions the full line is brought to the internal burst buffer. Instructions that originate in a cache-inhibited region are stored in the burst buffer and can be sent to the MPC801 core no more than once before being refetched. In the memory management unit, you can program a memory region to be cache-inhibited. When doing so, you must unlock all previously locked lines containing code that originated in this memory region, invalidate all lines containing code that originated in this memory region, and execute an **isync** instruction.

NOTE

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

When the MPC801 asserts the freeze signal, it indicates that the MPC801 is under debug and all fetches from the cache are treated as if they were from the cache-inhibited memory region. For more information on cache debug support, refer to **Section 9.9 Debug Support**.

9.4.7 Instruction Cache Read

The MPC801 allows you to read all data stored in the instruction cache, including the content of the tags array. However, this operation is privileged and if you try to use it when the core is in the problem state (MSR_{PR} =1) a program interrupt will occur. To read the data stored in the instruction cache, follow these steps:

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

So that it can access all parts of the instruction cache, the IC_ADR register is divided into the fields shown in Table 9-4.

Table 9-1. IC_ADR Bits Functionality for the Cache Read Command

0-17	18	19	20	21-27	28-29	30-31
Reserved	0 - Tag 1 - Data	0 - Way 0 1 - Way 1	Reserved	Set Select	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 register is placed in the targeted general-purpose register. When read from the tag array, the 21 bits of the tag and related information that is selected by the IC_ADR are all placed in the targeted general-purpose register. The following table provides the bit layout of the instruction cache data register when reading a tag.

Table 9-2. IC_DAT Bit Layout When Reading a Tag

0-21	22	23	24	25-31
Tag Value	0 - Not Valid 1 - Valid	0 - Not Locked 1 - Locked	LRU Bit	Reserved

9.4.8 Instruction Cache Write

Instruction cache write is only enabled when the MPC801 is in test mode.

9.5 RESTRICTIONS

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

9.6 INSTRUCTION CACHE COHERENCY

Cache coherency in a multiprocessor environment is maintained by the software and supported by the invalidation mechanism that is described above. All instruction storage is considered to be in Memory Coherence Not Required mode.

9.7 UPDATING CODE AND MEMORY REGION ATTRIBUTES

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

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

9.8 RESET SEQUENCE

To simplify the debug task of the system, the instruction cache is only disabled during hardware reset (IC_CST_{EN} = 0). This feature enables you to investigate the exact state of the instruction cache prior to the event that asserts the reset. To ensure proper operation of the instruction cache after reset, the unlock all, invalidate all, and instruction cache enable commands must be issued.

9.9 DEBUG SUPPORT

The MPC801 can be debugged either in debug mode or by a software monitor debugger. In both cases, the core asserts the internal freeze signal. When freeze is asserted the instruction cache treats all misses as if they were from cache-inhibited regions and, assuming the debug routine is not in the instruction cache, the cache state remains exactly the same. When 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 instruction cache) it is read from memory like any other miss. For performance reasons, you may prefer to run the debug routine from the cache by following these steps:

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

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

9.9.1 Fetching Instructions From the Development Port

When the MPC801 is in debug mode, all instructions are fetched from the development port, regardless of the address generated by the core. Therefore, the instruction cache is bypassed when the MPC801 is in debug mode.

SECTION 10
DATA CACHE

The MPC801 data cache is a 1K two-way, set-associative cache. It is organized into 32 sets at 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. Cache coherency in a multiprocessor environment is maintained by the software and supported by a fast hardware invalidation capability. The cache is designed for both writeback and writethrough modes of operation and a least recently used (LRU) replacement algorithm is used to select a line when no empty lines are available.

10.1 FEATURES

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

- 1K Two-Way, Set-Associative, and Physically Addressed
- Single-Cycle Cache Access on Hit and One Clock Latency Added for Miss
- Four Word Line Size
- "Critical word first" and Four Word Burst Line Fill
- LRU Replacement Policy
- 32-Bit Interface to Load/Store Unit
- One-Word Write Buffer
- Lockable Online Granularity
- Copyback/Writethrough Operation is Programmed per Memory Management Unit Page
- Coherency is Only Maintained By the Software 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

10.2 ORGANIZATION OF THE DATA CACHE

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

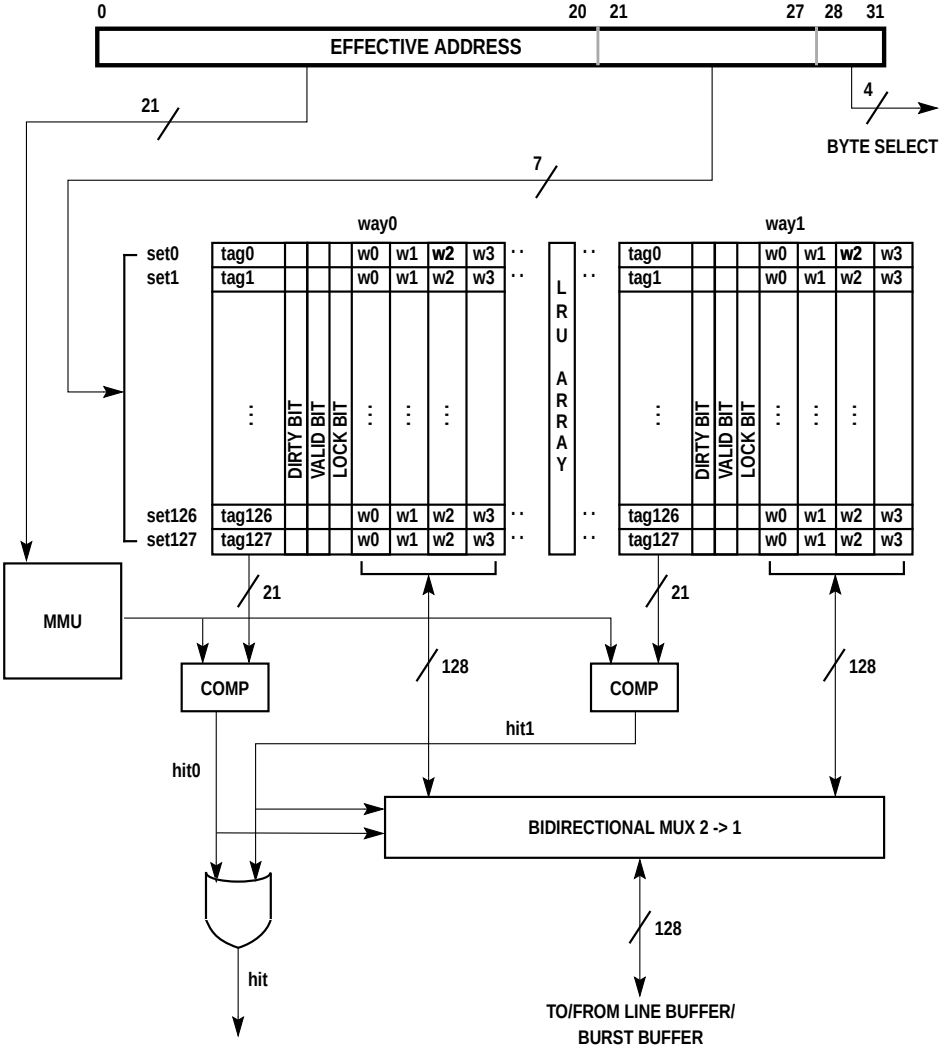


Figure 10-1. Data Cache Organization

10.3 PROGRAMMING THE DATA CACHE

10.3.1 PowerPC Architecture Instructions

The following PowerPC instructions are supported by the data cache.

10.3.1.1 POWERPC USER INSTRUCTION SET ARCHITECTURE

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

10.3.1.2 POWERPC VIRTUAL ENVIRONMENT ARCHITECTURE

The data cache supports the following instructions:

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

10.3.1.3 POWERPC OPERATING ENVIRONMENT ARCHITECTURE

The data cache also supports the data cache block invalidate (**dcbi**) instruction.

10.3.2 Implementation Specific Operations

The MPC801 data cache includes some extended features in addition to those in the PowerPC architecture. The following are implementation specific operations supported by the MPC801 data cache:

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

10.3.3 Special Registers of the Data Cache

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

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

These registers are privileged and if you try to access them while the core is in the problem state (MSR_{PR} =1) a program interrupt will occur.

10.3.3.1 DATA CACHE CONTROL AND STATUS REGISTER

DC_CST

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

DEN—Data Cache Enable Status

This bit is read-only and any attempt to write to it is ignored. It reflects the state of the data cache.

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

DFWT—Data Cache Force Writethrough

This bit is read-only and any attempt to write to it is ignored. It reflects the state of the data cache.

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

LES—Little-Endian Swap

This bit is a read-only bit and any attempt to write to it is ignored. Refer to **Section 14 Endian Modes** for details about how this bit is used to achieve the required endian behavior.

0 = Address of the data and the instruction caches is the unchanged address from the core. No byte swap is done on the data and instruction caches' external accesses.
1 = Address munging performed by the core is reversed before accessing the data cache, the instruction cache and storage. Byte swap is performed for the instruction and data cache's external accesses.

Bits 3, 8, and 9—Reserved

These bits are reserved and should be set to 0.

CMD—Data Cache Commands

0000 = Reserved.
0010 = Data Cache Enable.
0100 = Data Cache Disable.
0110 = Lock Line.
1000 = Unlock Line.
1010 = Unlock All.
1100 = Invalidate All.
1110 = Flush Data Cache Line.
0001 = Set Force Writethrough Mode.
0011 = Clear Force Writethrough Mode.
0101 = Set Little-Endian Swap Mode.
0111 = Clear Little-Endian Swap Mode.
Others = Reserved.

CCER1—Data Cache Error Type 1

These bits are sticky and set by the hardware. They are read-only and are cleared when they are read.

0 = No error.
1 = Error.

CCER2—Data Cache Error Type 2

These bits are sticky and set by the hardware. They are read-only and are cleared when they are read.

0 = No error.
1 = Error.

CCER3—Data Cache Error Type 3

These bits are sticky and set by the hardware. They are read-only and are cleared when they are read.

0 = No error.
1 = Error.

Bits 13–31—Reserved

These bits are reserved and should be set to 0.

10.3.3.2 DATA CACHE ADDRESS REGISTER

DC_ADR

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

ADR—Address

These bits represent the address to be used in the command programmed in the DC_CST register. It is also the internal address used to read tags and registers.

10.3.3.3 READING THE CACHE STRUCTURES

To read the data stored in the data cache tags or registers, follow these steps:

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

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

Table 10-1. DC_ADR Bit Functionality for Reading the Cache

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

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

- 0×00—Copyback data register 0
- 0×01—Copyback data register 1
- 0×02—Copyback data register 2
- 0×03—Copyback data register 3
- 0×04—Copyback address register

10.3.3.4 DATA CACHE DATA REGISTER

When reading from the data cache data (DC_DAT) register, the 21 bits of the tag and related information that is selected by the DC_ADR are placed in the targeted general-purpose register. The following table illustrates the DC_DAT register's bit layout when reading a tag. Writing to DC_DAT is illegal. A write to DC_DAT results in an undefined data cache state.

Table 10-2. DC_DAT Bit Layout For Reading a Tag

0-22	23	24	25	26	27-31
Tag Value	0 - Not Locked 1 - Locked	LRU Bit of this Set	0 - Clean 1 - Dirty	0 - Not Valid 1 - Valid	Reserved

10.4 OPERATING THE DATA CACHE

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

10.4.1 Data Cache Read

The cache read operation consists of the following items:

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

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

  After the line with the requested data has been brought from memory, the dirty line kept in the buffer is sent to the system interface unit to be written out (flushed) to memory. If a bus error is detected during the fetch of the missed "critical word", a machine check interrupt is generated. If a bus error occurs on any other word in the line transfer, the line is marked invalid.

On the other hand, if no bus error is encountered, the cache line is marked unmodified-valid. If a bus error is detected during the dirty line flush, a machine check interrupt is generated. For more information about reading the address and data of a line, see **Section 10.5.5 Reading the Data Cache Structures**.

10.4.2 Data Cache Write

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

10.4.2.1 COPYBACK MODE

In copyback mode, write operations do not necessarily update the external memory. For this reason, the copyback mode is the preferred mode of operation when it is important to keep bus bandwidth usage and power consumption to a minimum. Data cache operation on a write to copyback line is as follows:

- 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 way “zero” line is selected first. If neither of the two candidate lines in the selected set is invalid, then one of the lines is selected by the LRU algorithm for replacement. If the selected line is valid-modified (dirty), it is kept in a special buffer to be written out (flushed) to memory at a later time.

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

  After the line with the requested data has been brought from memory, the dirty line kept in the buffer is sent to the system interface unit to be written out (flushed) to memory. The data cache can support further requests as long as they hit in the cache while flushing the dirty line to memory. If a bus error is detected during a fetch of the missed line 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. For more information about reading the address and data of a line, see **Section 10.3.3.3 Reading the Cache Structures**.

10.4.2.2 WRITETHROUGH MODE

In writethrough mode, store operations always update memory. Use this mode when external memory and internal cache images must agree. It gives a lower worst-case interrupt latency at the expense of average performance. Data cache operation on a write to writethrough line is as follows:

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

10.4.3 Data Cache-Inhibited Accesses

If the cache access is to a page that has the CI bit set in the memory management unit, one of the following actions are performed:

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

10.4.4 Data Cache Freeze

The MPC801 can be debugged either in debug mode or by a software monitor debugger. In both cases, the core asserts the internal freeze signal. For a detailed description of MPC801 debug support, refer to **Section 18 Development Support**.

When freeze is asserted, the data cache will behave as follows:

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

10.4.5 Data Cache Coherency

The MPC801 data cache provides no support for snooping external bus activity. All coherency between the internal caches and memory/devices external to the extended core must be controlled by the software. In addition, there is no mechanism provided for DMA or other internal masters to access the data cache directly.

10.5 CONTROLLING THE DATA CACHE

10.5.1 Flushing and Invalidating

The MPC801 allows the software to explicitly flush and/or invalidate entries in the data cache. The data cache can be invalidated by 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. The data cache can be flushed by a software loop using the **dcbst** or **dcbf** instructions or the implementation-specific data cache flush cache line command. Notice that the PowerPC architecture instructions flush a line indexed by the address it represents, while the implementation-specific command indexes a line by its physical location within the data cache.

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

10.5.2 Disabling

The data cache can be enabled or disabled by using 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 the reset state of the data cache is disabled. Disabling the data cache does not affect the data address translation logic and translation is still controlled by the MSR_{DR} bit. Any write to the DC_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 a bus error on the copyback or 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.

10.5.3 Locking

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

10.5.4 Storage Control Instructions in the Data Cache

10.5.4.1 dcbi, dcbst, dcbf AND dcbz INSTRUCTIONS

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

10.5.4.2 TOUCH

The **dcbt** and **dcbtst** instructions in the MPC801 operate on a block basis of cache line, which is 16 bytes (4 words) long. They are treated as a nonoperation if the effective address of one of these instructions cannot be translated and relocation is enabled.

10.5.4.3 STORAGE SYNCHRONIZATION AND RESERVATION

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 cache-allowed mode, it is assumed that the system works with the single master in this storage region. Therefore, if a data cache miss occurs, the access on the internal and external buses does not have a reservation attribute.The MPC801 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 MPC801 does not provide support for snooping an external bus activity outside the chip. The provision is made to cancel the reservation inside the MPC801 by using the $\overline{CR}$ and $\overline{KR}$ input pins. Refer to **Section 13.4.9 Storage Reservation Protocol** for more information.

10.5.5 Reading the Data Cache Structures

To allow debug and recovery actions, the MPC801 allows the content of the tags array as well as the last copyback address and data buffers to be read. See **Section 10.3.3 Special Registers of the Data Cache** for details. This operation is privileged and if you try to use it when the core is in the problem state (MSR_{PR} =1), a program interrupt will occur.

SECTION 11
MEMORY MANAGEMENT UNIT

The MPC801 implements a virtual memory management scheme that provides cache control, storage access protection, and effective to real address translation. This implementation includes separate instruction and data memory management units. The MPC801 memory management unit (MMU) is compliant with the *PowerPC™ Microprocessor Family: The Programming Environment* in relation to the supported types of attributes, except that two new modes of operation have been added:

- PowerPC mode with extended encoding
- Domain manager mode

Available protection granularity sizes are page (4K, 16K, 512K, or 8M) or 1K subpage, the latter of which is only supported by 4K pages. Hereafter, the prefix M*x*_ that appears before an memory management unit control register name corresponds to both the MI_ and MD_ conditions.

11.1 FEATURES

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

- 8-Entry Fully Associative Data and Instruction Translation Lookaside Buffers (TLBs)
- Multiple Page Sizes
- High Performance
- Supports Up to 16 Virtual Address Spaces
- Supports 16 Access Protection Groups
- Each Entry Can be Programmed to Match Problem Accesses, Privileged Accesses, or Both.
- Generates Implementation Specific Interrupts
- Software Tablewalk Update Capability
- MSR_{IR} and MSR_{DR} Control Memory Management Unit Translation and Protection
- Supports PowerPC **tlbie** and **tlbia** Instructions. No **tlbsync** Instruction is Supported, but It Is Implemented as a NOP Instruction.
- Programming is Accomplished by Using the PowerPC **mtspr/mfspr** Instructions To or From the Implementation Specific Special-Purpose Registers
- A Special Scratch Register is Available for Software Tablewalks
- Designed for Minimal Power Consumption

11.2 ADDRESS TRANSLATION

The MPC801 core generates 32-bit effective addresses and, when enabled, the memory management unit translates the effective address to a real address that is used for cache or memory access. If disabled, the effective address is passed as the real address to the memory, which bypasses the appropriate TLB. Conceptually, in tables residing in the memory, the effective address is sought to provide the real address mapping and storage attributes. For performance reasons, instruction and data TLBs are implemented to hold recently used address translations. In the MPC801, the table lookup and TLB reload are performed by a software routine with little hardware assistance. This partition simplifies the hardware and gives the system the opportunity to choose the translation table structure.

A TLB hit in multiple entries is avoided during the TLB reload phase. The TLB logic recognizes that the effective page number (EPN) currently loaded into the TLB overlaps another EPN. At least when taking into account the page sizes, subpage validity flags, problem/privileged state, address space ID (ASID), and the SH values of the TLB entries. When such an event occurs, the current EPN is written into the TLB and the entry of the other EPN is invalidated from the TLB. The MPC801 memory management unit supports a multiple virtual address space model and, when enabled, each translation is associated with an ASID. For the translation to be valid, its ASID must be equal to the current address space ID (CASID) that is in effect when an access is performed.

11.2.1 Translation Lookaside Buffer Operation

The MPC801 has two translation lookaside buffers—one for instruction fetches and one for data accesses. Each of the translation lookaside buffers (TLB) contain pointers to pages in the real memory where data is indexed by the effective page number and it can hold entries with different page sizes. The entry page size controls the number of effective address bits to be compared and the number of least-significant effective address bits that remain untranslated and passes them as least-significant real address bits.

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

11.3 PROTECTION

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

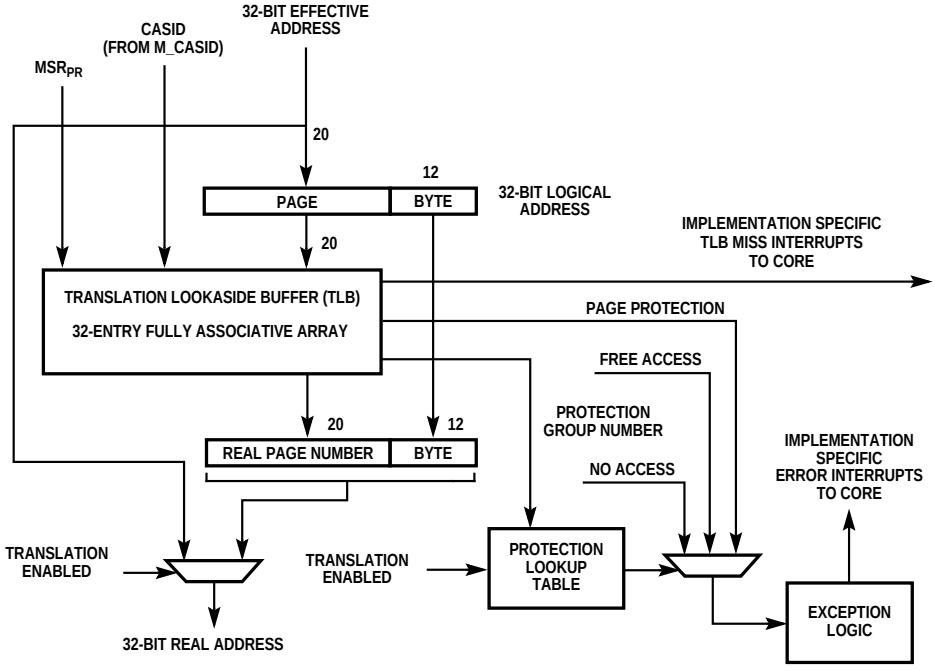


Figure 11-1. Effective to Real Address Translation For 4K Pages

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

11.4 STORAGE ATTRIBUTES

11.4.1 Reference and Change Bit Updates

The MPC801 does not generate an exception for an R bit update because there is no entry for an R bit in the TLB. The C bit updates are implemented by the software, but the hardware treats the C bit 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.

11.4.2 Storage Control

Each page can have different storage control attributes. The MPC801 supports CI, WT, and G attributes, but not the M attribute. A page that needs to be memory coherent must be programmed cache-inhibited. Refer to the definition of these attributes in the *PowerPC Microprocessor Family: The Programming Environment* manual. The effects of the CI and WT attributes in the MPC801 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 forces the access to stall until the access is nonspeculative or canceled by the core. Fetching from a guarded storage is prohibited and attempting to do so generates an implementation specific instruction storage interrupt. When MSR_{IR} or MSR_{DR} for instruction or data address translation are negated, default attributes are used. See Tables 11-6 and 11-7 for details.

11.5 TRANSLATION TABLE STRUCTURE

The MPC801 memory management unit includes special hardware to assist in a two-level software tablewalk. Other table structures are not precluded. Figures 11-2 and 11-3 illustrate the two levels of translation table structures supported by MPC860 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 (bits 0–9) of the effective address to get the level one page descriptor. For 8M pages, there must be two identical entries in the level one table for either Bit 9 = 0 or Bit 9 =1. See Table 11-2 for more information. The level two base address from the level one descriptor is indexed by the next ten least-significant bits (bits 0–9) to find the level two page descriptor. For pages larger than 4K, the entry in the level two table must be duplicated according to the page size. See Table 11-3 for more information.

During address translation by the memory management unit, the most-significant bits of the missed effective address are replaced by the real page address bits from the level two page descriptor. The number of replaced bits depends on the page size. The rest of the real address bits are taken directly from the effective address. When MD_CTR_{TWAM} = 0, the tablewalk begins at the level one base address placed in the M_TWB register. The level one table is indexed by the 12 most-significant bits (bits 0–11) of the effective address to get the level one page descriptor. For 8M pages, there must be eight identical entries in the level one table for bits 9–11 of the effective address. The level two base address from the level one descriptor is indexed by the next ten least-significant bits (bits 12–21) to find the level two page descriptor. For pages larger than 1K, the entry in the level two table must be duplicated according to the page size.

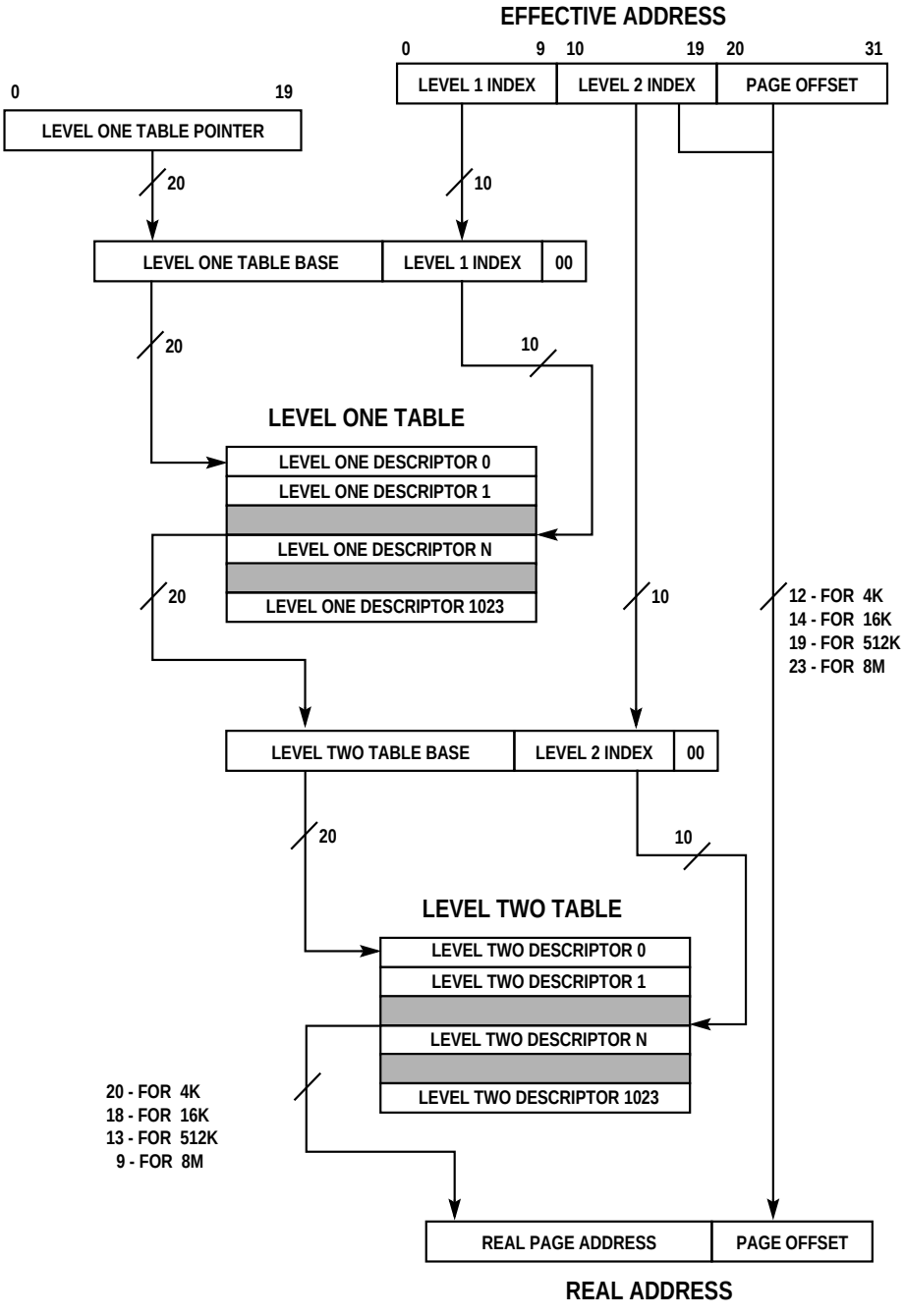


Figure 11-2. Two Level Translation Table When MD_CTR(TWAM) = 1

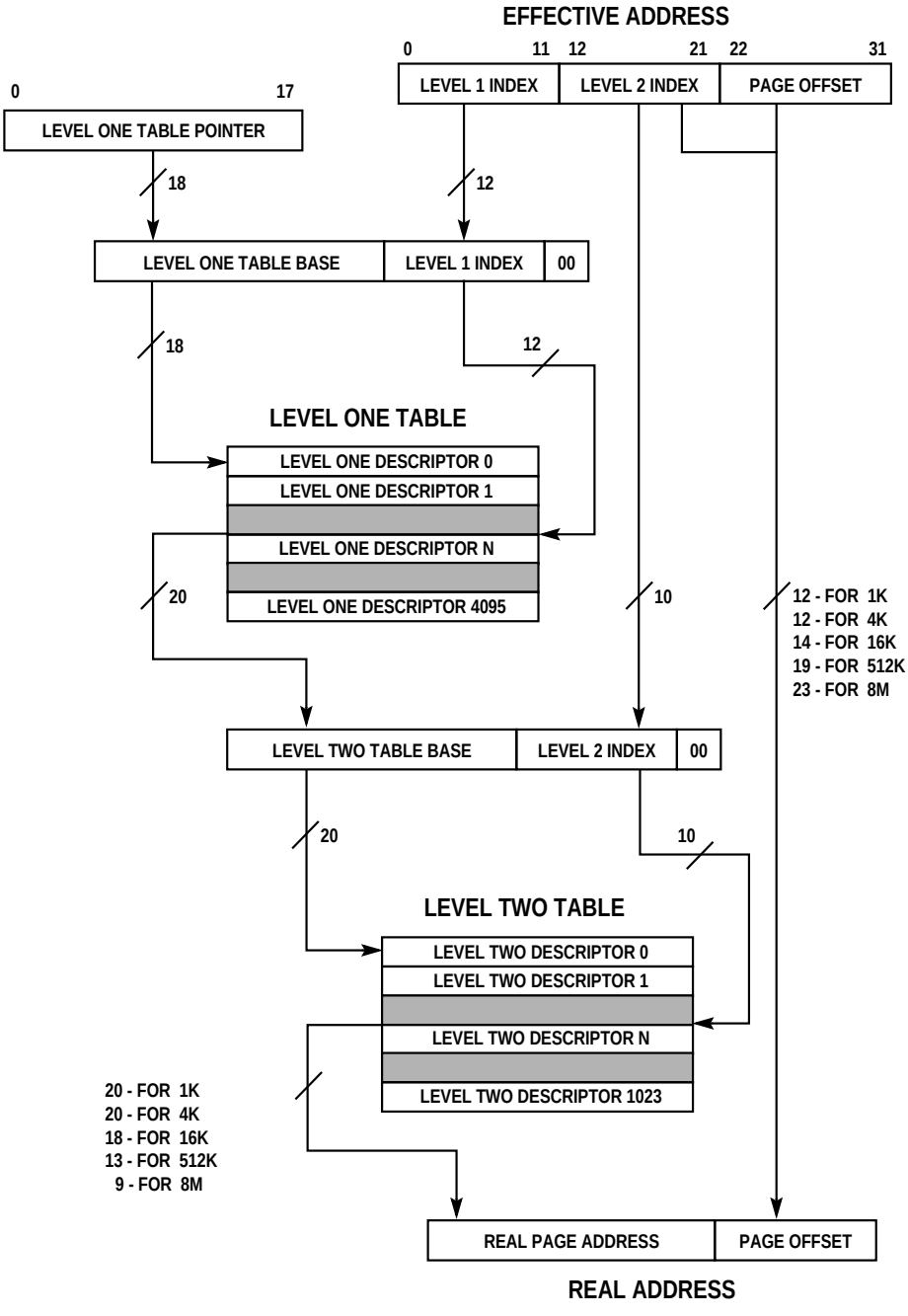


Figure 11-3. Two Level Translation Table When MD_CTR(TWAM) = 0

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

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

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

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

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

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

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

11.5.1 Level One Descriptor

The following table describes the hardware supported level one descriptor format that minimizes 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 -Unguarded Storage 1 - Guarded Storage
28-29	PS	Page Size Level One	00 - Small (4K Or 16K) 01 - 512K 11 - 8K 10 - Reserved
30	WT	Writethrough Attribute For Entry	0 - Copyback Cache Policy Region (Default) 1 - Writethrough Cache Policy Region
31	V	Segment Valid Bit	0 - Segment is Not Valid 1 - Segment is Valid

11.5.2 Level Two Descriptor

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

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

BITS	MNEMONIC	DESCRIPTION	4K PAGES WITH 1K RESOLUTION PROTECTION	4K RESOLUTION PROTECTION AND PAGES LARGER THAN 4K
0-19	RPN	Real Page Number	—	—
20-21	PP	Protection For The First 1K Subpage In A 4K Page	For Instruction Pages: Privileged / Problem 00 – No Access / No Access 01 – Executable / No Access 10 – Executable / Executable 11 – Executable / Executable For Data Pages Privileged / Problem 00 – No Access / No Access 01 – Read/Write / No Access 10 – Read/Write / Read-Only 11 – Read/Write / Read/Write	**For Instruction Pages:** **Privileged / Problem** Extended Encoding: 00 – No Access / No Access 01 – Executable / No Access 10 – Reserved 11 – Reserved PowerPC Encoding: 00 – Executable / Executable 01 – Executable / No Access 10 – Executable / Executable 11 – Executable / Executable **For Data Pages** **Privileged / Problem** Extended Encoding: 00 – No Access / No Access 01 – Read Only / No Access 10 – Reserved 11 – Reserved PowerPC Encoding: 11 – Read Only / Read Only 00 – Read/Write / No Access 01 – Read/Write / Read Only 10 – Read/Write / Read/Write
22-27	PP	Protection For A Second 1K Subpage In A 4K Page	For Instruction Pages: Privileged / Problem 00 – No Access / No Access 01 – Executable / No Access 10 – Executable / Executable 11 – Executable / Executable	0 – Bits 20-21 Contain PowerPC Encoding 1 – Bits 20-21 Contain Extended Encoding C – Change Bit For Entry 0 – Unchanged Region (Write-Protected) 1 – Changed Region (Write Allowed)

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

BITS	MNEMONIC	DESCRIPTION	4K PAGES WITH 1K RESOLUTION PROTECTION	4K RESOLUTION PROTECTION AND PAGES LARGER THAN 4K
			For Data Pages Privileged / Problem 00 – No Access / No Access 01 – Read/Write / No Access 10 – Read/Write / Read-Only 11 – Read/Write / Read/Write	MD_CTR(PPCS) = 0 0 – Subpage Is Not Valid 1 – Subpage Is Valid If The Page Size Is Larger Than 4K. Then All Four Bits Should Have The Same Value. MD_CTR(PCCS) = 1 1000 – Hit Only For Privileged Accesses 0100 – Hit Only For Problem Accesses 1100 – Hit For Both
28	SPS	Small Page Size	Should be 0	0 – 4K 1 – 16K
29	SH	Shared Page	0 – This Entry Marches Only If The ASID Filed In The TLB Entry Matches The Value Of The M_CASID Register. 1 – ASID Comparison Is Disabled For The Entry.	
30	CI	Cache Inhibit	Cache-Inhibit Attribute For The Entry.	
31	V	Valid Bit	Page Valid Bit	

11.6 MEMORY MANAGEMENT UNIT PROGRAMMING MODEL

All memory management unit special registers can be accessed by the PowerPC **mtspr**/**mfspr** instructions. In addition, the PowerPC **tlbie** and **tlbia** architecture instructions are supported. Memory management unit registers should be accessed when both MSR_{IR} =0 and MSR_{DR} =0. No similar restriction exists for the **tlbie** and **tlbia** instructions.

11.6.1 Configuration Registers

11.6.1.1 INSTRUCTION MMU CONTROL REGISTER

MI_CTR

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

GPM—Group Protection Mode

0 = PowerPC mode.

1 = Domain manager mode.

PPM—Page Protection Mode

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

CIDEF—CI Default

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

Bits 3 and 5—Reserved

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

RSVI—Reserve Two Instruction TLB Entries

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

PPCS—Privilege/Problem State Compare Mode

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

Bits 7–18 and 24–31—Reserved

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

ITLB_INDX—Instruction TLB Index

These bits act as pointers to the instruction TLB entry to be loaded. They are automatically decremented at every instruction TLB update.

11.6.1.2 MI_AP REGISTER. The reset value of the instruction MMU access protection (MI_AP) register is undefined.

MI_AP

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

GP—Group Protection

In domain manager mode MI_CTR (GPM) = 1, these bits have the following settings:

00 = No access.
01 = Client-access permission defined by page protection bits.
10 = Reserved.
11 = Manager-free access.

In PowerPC group protection mode MI_CTR(GPM) = 0, the GP bits have these settings and are Ks and Kp in *PowerPC Microprocessor Family: The Programming Environment:*

00 = All accesses are considered privileged.
01 = Access permission defined by page protection bits.
10 = Problem and privileged interpretation is swapped.
11 = All accesses are considered problem.

11.6.1.3 DATA MMU CONTROL REGISTER

MD_CTR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	GPM	PPM	CIDEF	WTDEF	RSV4D	TWAM	PPCS	RESERVED								
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED			DTLB_INDX					RESERVED							

GPM—Group Protection Mode

0 = PowerPC mode.
1 = Domain manager mode.

PPM—Page Protection Mode

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

CIDEF—CI Default

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

WTDEF—WT Default

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

RSVD—Reserve Two Data TLB Entries

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

TWAM—Tablewalk Assist Mode

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

PPCS—Privilege/Problem State Compare Mode

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

Bits 7–18 and 24–31—Reserved

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

ITLB_INDX—Instruction TLB Index

These bits act as pointers to the data TLB entry to be loaded. They are automatically decremented at every data TLB update.

11.6.1.4 MD_AP REGISTER. The reset value of the data MMU access protection (MC_AP) register is undefined.

MD_AP

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

GP—Group Protection

In domain manager MD_CTR (GPM) = 1 mode, these bits have the following settings:

00 = No access.
01 = Client-access permission defined by page protection bits.
10 = Reserved.
11 = Manager-free access.

In PowerPC group protection mode MI_CTR (GPM) = 0, the GP bits have these settings and are Ks and Kp in *PowerPC Microprocessor Family: The Programming Environment:*

00 = All accesses are considered privileged.
01 = Access permission defined by page protection bits.
10 = Problem and privileged interpretation is swapped.
11 = All accesses are considered problem.

11.6.1.5 CASID REGISTER. The reset value of the current address space ID (CASID) register is undefined. This register is typically loaded by a task switch handler.

M_CASID

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

Bits 0–27—Reserved

These bits are reserved and should be set to 0.

CASID—Current Address Space ID

This field is compared with the ASID field of a TLB entry to qualify a match.

11.6.2 Tablewalk Registers

11.6.2.1 M_TWB REGISTER. The reset value of the MMU tablewalk base (M_TWB) register is undefined.

M_TWB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	L1TB															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	L1TB				L1INDX										RESERVED	

L1INDX—Level 1 Table Index

These bits are ignored on write. They return MD_EPN[0:9] on read when $MD\_CTR_{TWAM} = 1$ and MD_EPN[2:11] when $MD\_CTR_{TWAM} = 0$.

Bits 30–31—Reserved

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

11.6.2.2 M_TW REGISTER. The MMU tablewalk scratch (M_TW) register is used in the software tablewalk interrupt handlers to save one of the interrupted task general-purpose registers. The reset value of this register is undefined.

M_TW

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

11.6.2.3 MI_EPN REGISTER. The reset value of the instruction MMU effective page number (MI_EPN) register is undefined. This register is used to define the effective page number and address space ID to be loaded into the translation lookaside buffer.

MI_EPN

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EPN															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EPN				RESERVED		EV	RESERVED					ASID			

EPN—Effective Page Number for the TLB Entry

This field is the effective address's default value of the last instruction TLB miss.

Bits 20–21 and 23–27—Reserved

These bits are reserved and should be set to 0.

EV—TLB Entry Valid Bit

This bit is automatically set to 1 on every instruction TLB miss.

0 = The TLB entry is invalid.
1 = The TLB entry is valid.

ASID—Address Space ID

These bits represent the address space ID of the instruction TLB entry to be compared with the CASID field of the M_CASID register.

11.6.2.4 MI_TWC REGISTER. The reset value of the instruction MMU tablewalk control (MI_TWC) register is undefined.

MI_TWC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED							APG				G	PS		RES	V

Bits 0–22 and 30—Reserved

These bits are reserved and should be set to 0.

APG—Access Protection Group

Up to 16 protection groups supported. Default value on instruction TLB miss is 0.

G—Guarded Storage Attribute for Entry

Default value on instruction TLB miss is 0.

0 = Unguarded storage.
1 = Guarded storage.

PS—Page Size Level One

Default value on instruction TLB miss is 00.

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

V—Entry Valid

Default value on instruction TLB miss is 1.

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

11.6.2.5 MI_RPN REGISTER. The reset value of the instruction MMU real page number port (MI_RPN) register is undefined. Writing to this register causes the values/attributes stored in MI_EPN, MI_TWC, and those written to this register, to be loaded into the TLB entry pointed by the MI_CTR register.

MI_RPN

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PP0		PP1		PP2		PP3		LPS	SH	CI	V

PP0–PP3—Page Protection 0–3

These bits represent the protection attributes for the first, second, third, and fourth subpages of a 4K page.

For 4K pages with 1K resolution protection, the PP bits function as follows.

	PRIVILEGED	PROBLEM
00	No Access	No Access
01	Executable	No Access
10	Executable	Executable
11	Executable	Executable

For 4K resolution protection and pages larger than 4K, the PP bits function as follows.

		PRIVILEGED	PROBLEM
Extended Encoding			
	00	No Access	No Access
	01	Executable	No Access
	10	Reserved	Reserved
	11	Reserved	Reserved
PowerPC Encoding			
	11	Executable	Executable
	00	Executable	No Access
	01	Executable	Executable
	10	Executable	Executable

SPV0–SPV3—1K Subpage Valid 0–3

For page sizes larger than 4K, all four bits should have the same value and be set as follows:

If MD_CTR(PCCS) = 0:
0 = Subpage is invalid.
1 = Subpage is valid.

If MD_CTR(PCCS) = 1:
1000 = Hit only for privileged accesses.
0100 = Hit only for problem accesses.
1100 = Hit for privileged and problem accesses.

LPS—Large Page Size

For 4K pages with 1K resolution protection, the PP bits should be 0. For 4K resolution protection and pages larger than 4K, the PP bits function as follows:

0 = 4K.
1 = 16K.

SH—Shared Page

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

CI—Cache Inhibit

Cache-inhibit attribute for the entry.

V—Valid

Entry valid indication.

11.6.2.6 MD_EPN REGISTER. The reset value of the data MMU effective page number (MD_EPN) register is undefined. This register is copied to the appropriate TLB entry when a write is made to the MD_RPN register.

MD_EPN

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EPN															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EPN						EV	RESERVED					ASID			

EPN—Effective Page Number for Entry

The default value is the effective address of the last data TLB miss.

EV—TLB Entry Valid

This bit is set to 1 on a data TLB miss. This bit specifies a valid TLB bridge when loaded into a TLB entry.

0 = The data TLB entry is invalid.
1 = The data TLB entry is valid.

Bits 23–27—Reserved

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

ASID—Address Space ID

These bits are the address space IDs of the TLB entry to be compared with the CASID field of the M_CASID register.

11.6.2.7 DATA MMU TABLEWALK CONTROL REGISTER.

MD_TWC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	L2TB															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	L2TB				L2INDX			APG/L2INDX				G/ L2INDX	PS/L2INDX		WT/ L2INDX	V/ L2INDX

L2INDX—Level 2 Table Index

When read, these bits return MD_EPN[10:19] when MD_CTR_{TWAM} = 1 and MD_EPN[12:21] when MD_CTR_{TWAM} = 0.

APG/L2INDX—Access Protection Group on Write and Level 2 Table Index on Read

When written, the APG bits support a maximum of 16 protection groups. They are set to 0000 on the data TLB miss.

G/L2INDX—Guarded on Write and L2INDX on Read

When written, the G bit of the entry has the following settings and is set to 0 on a data TLB miss:

0 = Unguarded storage.
1 = Guarded storage.

PS/L2INDX—Page Size on Write and L2INDX on Read

When written, the PS bits have the following settings and are set to 0 on a data TLB miss:

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

WT/L2INDX—Writethrough on Write and 0 on Read

When written, the WT bit has the following settings and is set to 1 on a data TLB miss:

0 = Copyback data cache policy page entry.
1 = Writethrough data cache policy page entry.

When read, a zero is returned.

V/L2INDX—Valid Entry on Write and 0 on Read

When written, the V bit has the following settings and is set to 1 on a data TLB miss:

0 = Entry is invalid.
1 = Entry is valid.

When read, a zero is returned.

11.6.2.8 MD_RPN REGISTER. Writing to the data MMU real page number port (MD_RPN) register causes the values/attributes stored in MI_EPN, MI_TWC, and those written to this register, to be loaded into the TLB entry pointed by the MI_CTR register.

MD_RPN

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PP0		PP1		PP2		PP3		LPS	SH	CI	V

PP0–PP3—Page Protection 0–3

These bits represent the protection attributes for the first, second, third, and fourth subpages of a 4K page.

For 4K pages with 1K resolution protection, the PP bits function as follows.

	PRIVILEGED	PROBLEM
00	No Access	No Access
01	Read/Write	No Access
10	Read/Write	Read-Only
11	Read/Write	Read/Write

For 4K resolution protection and pages larger than 4K, the PP bits function as follows.

		PRIVILEGED	PROBLEM
Extended Encoding			
	00	No Access	No Access
	01	Read-Only	No Access
	10	Reserved	Reserved
	11	Reserved	Reserved
PowerPC Encoding			
	11	Read-Only	Read-Only
	00	Read/Write	No Access
	01	Read/Write	Read-Only
	10	Read/Write	Read/Write

For 4K resolution protection and pages larger than 4K, the change bit for data TLB entry has the following settings:

0 = Unchanged region. A write access to this page invokes an implementation specific instruction memory management unit interrupt. The software should take the appropriate action before setting this bit to 1.
1 = Changed region. A write access to this page is allowed.

SPV0–SPV3—1K Subpage Valid 0–3

For page sizes larger than 4K, all four bits should have the same value and be set as follows:

If MD_CTR(PCCS) = 0:
0 = Subpage is invalid.
1 = Subpage is valid.

If MD_CTR(PCCS) = 1:
1000 = Hit only for privileged accesses.
0100 = Hit only for problem accesses.
1100 = Hit for privileged and problem accesses.

LPS—Large Page Size

For 4K pages with 1K resolution protection, the PP bits should be 0.

For 4K resolution protection and pages larger than 4K, the PP bits function as follows:

0 = 4K.
1 = 16K.

SH—Shared Page

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

CI—Cache Inhibit

Cache-inhibit attribute for the entry.

V—Valid

Entry valid indication

11.6.3 Instruction Debug Registers

The MI_DCAM, MI_DRAM0, and MI_DRAM1 registers are interface registers that enable them 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 to 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. If you try to write to the MI_DRAM0 and MI_DRAM1 registers using the **mtspr** instruction it will be considered a NOP.

11.6.3.1 MI_DCAM REGISTER. The reset value of instruction MMU CAM entry read (MI_DCAM) register is undefined.

MI_DCAM

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EPN															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EPN				PS			ASID				SH	SPV			

PS—Page Size

000 = 4K.
001 = 16K.
011 = 512K.
111 = 8M.
010 = Reserved.
100 = Reserved.
101 = Reserved.
110 = Reserved.

ASID—Address Space ID

These bits represent the data TLB entry to be compared with the CASID field in the M_CASID register.

SH—Shared Page

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

SPV—Subpage Validity

Bit 28:
0 = Subpage 0 (address[20:21]=00) is not valid.
1 = Subpage 0 (address[20:21]=00) is valid.

Bit 29:
0 = Subpage 1 (address[20:21]=01) is not valid.
1 = Subpage 1 (address[20:21]=01) is valid.

Bit 30:
0 = Subpage 2 (address[20:21]=10) is not valid.
1 = Subpage 2 (address[20:21]=10) is valid.

Bit 31:
0 = Subpage 3 (address[20:21]=11) is not valid.
1 = Subpage 3 (address[20:21]=11) is valid.

11.6.3.2 MI_DRAM0 REGISTER. The reset value of the instruction MMU RAM entry read register 0 (MI_DRAM0) is undefined.

MI_DRAM0

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PS_B			CI	APG				SFP			

PS_B—Page Size

000 = 4K.
001 = 16K.
011 = 512K.
111 = 8M.
010 = Reserved.
100 = Reserved.
101 = Reserved.
110 = Reserved.

APG—Access Protection Group

A maximum of 16 protection groups are supported and are represented in one’s compliment format.

SFP—Privileged (Supervisor) Fetch Permission

Bit 28:
0 = Subpage 0 (address[20:21]=00) privileged fetch is not permitted.
1 = Subpage 0 (address[20:21]=00) privileged fetch is permitted.

Bit 29:
0 = Subpage 1 (address[20:21]=01) privileged fetch is not permitted.
1 = Subpage 1 (address[20:21]=01) privileged fetch is permitted.

Bit 30:
0 = Subpage 2 (address[20:21]=10) privileged fetch is not permitted.
1 = Subpage 2 (address[20:21]=10) privileged fetch is permitted.

Bit 31:
0 = Subpage 3 (address[20:21]=11) privileged fetch is not permitted.
1 = Subpage 3 (address[20:21]=11) privileged fetch is permitted.

11.6.3.3 MI_DRAM1 REGISTER. The reset value of the instruction MMU RAM entry read register 1 (MI_DRAM1) is undefined.

MI_DRAM1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED										UFP				PV	G

Bits 0–25—Reserved

These bits are reserved and should be set to one.

UFP—Problem (User) Fetch Permission

Bit 26:
0 = Subpage 0 (address[20:21]=00) problem fetch is not permitted.
1 = Subpage 0 (address[20:21]=00) problem fetch is permitted.

Bit 27:
0 = Subpage 1 (address[20:21]=01) problem fetch is not permitted.
1 = Subpage 1 (address[20:21]=01) problem fetch is permitted.

Bit 28:
0 = Subpage 2 (address[20:21]=10) problem fetch is not permitted.
1 = Subpage 2 (address[20:21]=10) problem fetch is permitted.

Bit 29:
0 = Subpage 3 (address[20:21]=11) problem fetch is not permitted.
1 = Subpage 3 (address[20:21]=11) problem fetch is permitted.

PV—Page Validity

0 = Page is invalid.
1 = Page is valid.

G—Guarded Storage Attribute for Entry

This bit defines whether a page is guarded or not. It should be set to 1 for storage that has side effects on read. Otherwise, it should be set to zero.

0 = Unguarded storage.
1 = Guarded storage.

11.6.4 Data Debug Registers

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. If you try to write to the MD_DRAM0 and MD_DRAM1 registers using the **mtspr** instruction it will be considered a NOP.

11.6.4.1 MD_DCAM REGISTER. The reset value of the data MMU CAM entry read register is undefined.

MD_DCAM

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EPN															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EPN				SPVF				PS			SH	ASID			

SPVF—Subpage Validity Flags

For Bit 20:
0 = Subpage 0 (address[20:21] = 00) is not valid.
1 = Subpage 0 (address[20:21] = 00) is valid.

For Bit 21:
0 = Subpage 1 (address[20:21] = 01) is not valid.
1 = Subpage 1 (address[20:21] = 01) is valid.

For Bit 22:
0 = Subpage 2 (address[20:21] = 10) is not valid.
1 = Subpage 2 (address[20:21] = 10) is valid.

For Bit 23:
0 = Subpage 3 (address[20:21] = 11) is not valid.
1 = Subpage 3 (address[20:21] = 11) is valid.

PS—Page Size

000 = 4K.
001 = 16K.
011 = 512K.
111 = 8M.
010 = Reserved.
100 = Reserved.
101 = Reserved.
110 = Reserved.

SH—Shared Page

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

ASID—Address Space ID

These bits are the address space IDs of the TLB entry to be compared with the CASID field of the M_CASID register.

11.6.4.2 MD_DRAM0 REGISTER. The reset value of the data MMU RAM entry read register 0 (MD_DRAM0) is undefined.

MD_DRAM0

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PS			APGI				G	WT	CI	RESERVED	

PS—Page Size

000 = 4K.
001 = 16K.
011 = 512K.
111 = 8M.
010 = Reserved.
100 = Reserved.
101 = Reserved.
110 = Reserved.

APGI—Access Protection Group Inverted

These bits are represented in one’s complement format.

G—Guarded Storage Attribute for the Entry

0 = Unguarded storage.
1 = Guarded storage.

WT—Writethrough Attribute for the Entry

0 = Copyback data cache policy page entry.
1 = Writethrough data cache policy page entry.

Bits 30–31—Reserved

These bits are reserved and should be set to 0.

11.6.4.3 MD_DRAM1 REGISTER. The reset value of the data MMU RAM entry read register 1 (MD_DRAM1) is undefined.

MD_DRAM1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RES	C	EVF	SA				SAT	URP0	UWP0	URP1	UWP1	URP2	UWP2	URP3	UWP3

Bits 0–16—Reserved

These bits are reserved and should be set to 0.

C—Change Bit for Data Entry TLB

0 = Unchanged region. Write access to this page results in the implementation specific IMMU interrupt invocation. Software should take an appropriate action before setting this bit to 1.
1 = Changed region. Write access is allowed to this page.

EVF—Entry Valid Flag

0 = Entry is invalid.
1 = Entry is valid.

SA—Privileged (Supervisor) Access

For Bit 19:
0 = Subpage 0 (address[20:21]=00) privileged access is not permitted.
1 = Subpage 0 (address[20:21]=00) privileged access is permitted.

For Bit 20:
0 = Subpage 1 (address[20:21]=01) privileged access is not permitted.
1 = Subpage 1 (address[20:21]=01) privileged access is permitted.

For Bit 21:
0 = Subpage 2 (address[20:21]=10) privileged access is not permitted.
1 = Subpage 2 (address[20:21]=10) privileged access is permitted.

For Bit 22:
0 = Subpage 3 (address[20:21]=11) privileged access is not permitted.
1 = Subpage 3 (address[20:21]=11) privileged access is permitted.

SAT—Privileged (Supervisor) Access Type
0 = Privileged access type is read-only.
1 = Privileged access type is read/write.

URP0—Problem (User) Read Permission Page 0
0 = Subpage 0 (address[20:21]=00) problem read access is not permitted.
1 = Subpage 0 (address[20:21]=00) problem read access is permitted.

UWP0—Problem (User) Read Permission Page Zero
0 = Subpage 0 (address[20:21]=00) problem write access is not permitted.
1 = Subpage 0 (address[20:21]=00) problem write access is permitted.

URP1—Problem (User) Read Permission Page 1
0 = Subpage 1 (address[20:21]=01) problem read access is not permitted.
1 = Subpage 1 (address[20:21]=01) problem read access is permitted.

URP2—Problem (User) Read Permission Page Two
0 = Subpage 2 (address[20:21]=10) problem read access is not permitted.
1 = Subpage 2 (address[20:21]=10) problem read access is permitted.

UWP2—Problem (User) Write Permission Page Two
0 = Subpage 2 (address[20:21]=10) problem write access is not permitted.
1 = Subpage 2 (address[20:21]=10) problem write access is permitted.

URP3—Problem (User) Read Permission Page Three
0 = Subpage 3 (address[20:21]=11) problem read access is not permitted.
1 = Subpage 3 (address[20:21]=11) problem read access is permitted.

UWP3—Problem (User) Write Permission Page Three
0 = Subpage 3 (address[20:21]=11) problem write access is not permitted.
1 = Subpage 3 (address[20:21]=11) problem write access is permitted.

11.7 INTERRUPTS

11.7.1 Implementation Specific Instruction TLB Miss

The implementation specific instruction TLB miss interrupt occurs when MSR_{IR} =1 and there is an attempt to fetch an instruction from a page whose effective page number cannot be 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 11.8.1.1 Translation Reload Examples** for more information.

11.7.2 Implementation Specific Data TLB Miss

The implementation specific data TLB miss interrupt occurs when MSR_{DR} =1 and there is an attempt to access a page whose effective page number cannot be 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 11.8.1.1 Translation Reload Examples** for more information.

11.7.3 Implementation Specific Instruction TLB Error

The implementation specific instruction TLB error interrupt occurs under the following conditions:

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

The reason for invoking the instruction TLB error interrupt handler can be found in the save/restore register 1. For bit assignments, refer to **Section 7.3.7.3.12 Implementation Specific Instruction TLB Error Interrupt**. It is the software's responsibility to invoke the instruction storage interrupt handler.

11.7.4 Implementation Specific Data TLB Error

The implementation specific data TLB error interrupt occurs under one of the following conditions:

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

The data storage interrupt status register explains how the data TLB error interrupt handler is invoked. For bit assignments, refer to **Section 7.3.7.3.14 Implementation Specific Data TLB Error Interrupt**. It is the software's responsibility to invoke the data storage interrupt handler.

11.8 MANIPULATING THE TLB

11.8.1 Reloading the TLB

The TLB reload (tablewalk) function is performed in the software, but the hardware assists in the following ways:

- Automatically stores the missed effective data or instruction address and default attributes in the MI_EPN or MD_EPN registers, respectively. This value is loaded into the selected entry on a write to MI_RPN or MD_RPN for the instruction and data TLB.
- Automatically updates the replacement location counter to point to the entry to be replaced. This value is placed in the index field of the MI_CTR and MD_CTR registers.
- Generates a level one pointer 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.
- Generates a level two pointer when a **mfspr** Rx, MD_TWC is performed by the concatenation of the level two table base with the level two index.
- Performs a write to the TLB entry by loading the tablewalk level two entry value to the MI_RPN or MD_RPN register.
- Makes a special register available for the software tablewalk routine, in addition to the architecture's four operating system special registers—SPRG0- SPRG3. This allows the miss code to save enough general-purpose registers so that it can execute without corrupting the state of the existing general-purpose registers.

11.8.1.1 TRANSLATION RELOAD EXAMPLES

The following are code examples for generating the real page number using a two-level tree page table structure. The first example is of the data TLB reload and the second 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
```

11.8.2 Controlling the TLB Replacement Counter

The TLB replacement counter can be controlled to only select among the first 6 entries in each TLB by setting the $\overline{\text{RSVI}}$ bit in the MI_CTR register or the $\overline{\text{RSVD}}$ bit in the MD_CTR register. These control bits also affect the **tlbia** instruction. Replacement counters are cleared to zero after the **tlbia** instruction is executed and the counters decrement after an appropriate TLB reload.

11.8.3 Invalidating the TLB

The MPC801 implements the **tlbie** instruction to invalidate the TLB entries. This instruction invalidates TLB entries in the TLB that hits, including the reserved entries. The 22 most-significant bits of the effective address are used in comparison since no segment registers are implemented. Although, for entries with page sizes greater than 4K, some of the lower bits of the effective page number are ignored. The ASID value in the entry is ignored for the purpose of matching an invalid address, thus multiple entries may be invalidated if they have the same effective address and different ASID values. The MPC801 supports the **tlbia** instruction to invalidate all entries in both TLBs. If the $\overline{\text{RSVD}}$ or $\overline{\text{RSVI}}$ bit is set for a TLB, the two reserved entries will not be invalidated when **tlbia** is executed. However, the software can explicitly invalidate one or more of these entries by setting the index field in MD_CTR (DTLB_INDX) or MI_CTR (ITLB_INDX), negating the EV bit 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.

11.8.4 Loading the Reserved TLB Entries

To load a single reserved entry in the TLB, follow these steps:

1. Disable the TLB by clearing MSR_{IR} or MSR_{DR} as needed.
2. Clear the $\overline{\text{RSVI}}$ ($\overline{\text{RSVD}}$) bit in the MI_CTR (MD_CTR) register.
3. Invalidate the effective address of the reserved page by using the **tlbia** or **tlbie** instruction.
4. Set the ITLB_INDX (DTLB_INDX) fields of the MI_CTR (MD_CTR) register to the appropriate value between 27 and 31.
5. Load the MI_EPN (MD_EPN) register with the effective page number, the ASID of the reserved page, and 1 as the EV bit.
6. Run software tablewalk code to load the appropriate entry into the TLB.
7. If needed, repeat the three previous steps to load other TLB entries.
8. Set the RSVI ($\overline{\text{RSVD}}$) bit in the MI_CTR (MD_CTR) register.

11.9 REQUIREMENTS FOR ACCESSING THE MEMORY MANAGEMENT UNIT CONTROL REGISTERS

All instruction and data memory management unit 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_DCAM Rx instruction, an **eieio** instruction should be placed.

SECTION 12
SYSTEM INTERFACE UNIT

This section summarizes the MPC801 system interface unit (SIU) functions that control system startup, initialization, operation, protection, and the external bus. The system configuration and protection function controls the overall system and provides various monitors and timers, including the bus monitor, software watchdog timer, periodic interrupt timer, PowerPC™ decrementer, timebase, and real-time clock. The clock synthesizer generates the clock signals and other modules and external devices that the system interface unit uses. This circuitry generates the system clock from an inexpensive 32KHz/4MHz crystal. The system interface unit 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 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 buses and the memory or peripherals in the external address space. The MPC801 is designed to allow external bus masters to request and obtain mastership of the system bus. **Section 13 External Bus Interface** describes the bus operation, but the configuration control of the external bus interface is explained in this section. The memory controller module provides a glueless interface to many types of memory devices and peripherals. It supports a maximum of eight memory banks, each with its own device and timing attributes. Memory control services are provided to both internal and external masters.

NOTE

The chip's address bus is 26 bits wide, but the internal address bus is 32 bits wide. Therefore, external accesses are internally considered 26 bit accesses (A[6–31]) with A[0–5] equal to 0, while internal accesses are full 32-bit accesses.

The MPC801 implementation supports circuit board test strategies through a user-accessible test logic that is fully compliant with **Section 19 IEEE 1149.1 Test Access Port**.

12.1 FEATURES

The following is a list of the system interface unit's main features:

- System Configuration and Protection
- System Reset Monitoring and Generation
- Clock Synthesizer
- Power Management
- External Bus Interface Control
- Eight Memory Banks Supported by the Memory Controller
- Debug Support
- IEEE 1149.1 Test Access Port

12.2 SYSTEM CONFIGURATION AND PROTECTION

The MPC801 incorporates many system functions that normally must be provided in external circuits. It is designed to provide maximum system safeguards against hardware and/or software faults. The following features are provided in the system configuration and protection submodule:

- **System Configuration**—Allows you to configure the system according to particular requirements. The functions include control of parity checking, show cycle operation, and part and mask number constants.
- **Bus Monitor**—Monitors the $\overline{TA}$ response time for all bus accesses initiated by the internal masters. A $\overline{TEA}$ signal is asserted if the $\overline{TA}$ response limit is exceeded. This function can be disabled when necessary.
- **Software Watchdog Timer**—Asserts a reset or nonmaskable interrupt selected by the system protection control register (SYPCR) if the software fails to service the software watchdog timer (SWT) after a certain period of time. After a system reset this function is enabled, selects a maximum timeout period, and asserts a system reset if the timeout is reached. The software watchdog timer can be disabled or its timeout period 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 to be used with a real-time operating system or application software. The periodic interrupt timer (PIT) is clocked by the pitrtclk clock, thus providing a period from 122 microseconds to 8,000 milliseconds assuming a 32.768-KHz crystal. This function can be disabled if necessary.
- **PowerPC Timebase Counter**—This 64-bit counter that is defined by the PowerPC architecture provides a timebase reference for the operating system or application software. The timebase counter (TB) has four independent reference registers that generate a maskable interrupt when the timebase counter reaches the value programmed in one of the four reference registers. The associated bit in the timebase status register is set for the reference register that generated the interrupt. The timebase is clocked by the tmbclk clock.

- **PowerPC Decrementer**—This 32-bit decrementing counter that is defined by the PowerPC architecture provides a decrementer interrupt. This binary counter is clocked by the same frequency as the timebase. The decrementer (DEC) is clocked by the tmbclk clock and the period in which it is driven by a 4MHz oscillator is 4,295 second, which is approximately 71.6 minutes.
- **Real-Time Clock**—Provides time-of-day information to the operating system or application software. It is composed of a 45-bit counter and an alarm register. A maskable interrupt is generated when the counter reaches the value programmed in the alarm register. The real-time clock (RTC) is clocked by the pitrtclk clock.
- **Freeze Support**—The system interface unit allows you to control whether the software watchdog timer, periodic interrupt timer, timebase, decrementer, and real-time clock should continue to run during freeze mode.

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

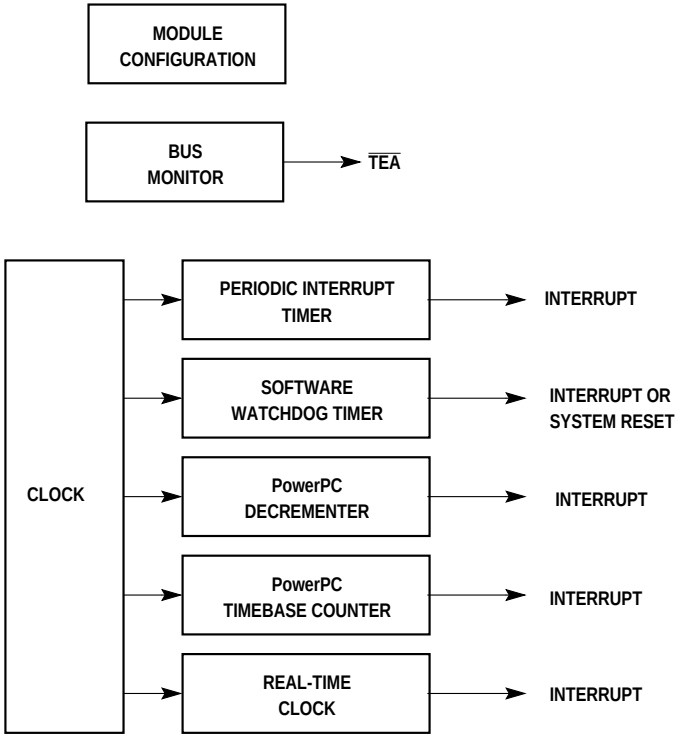


Figure 12-1. System Configuration and Protection Logic

12.3 CONFIGURING THE SYSTEM

Many aspects of the system configuration are controlled by the SIU module configuration register (SIUMCR). See **Section 12.12 Programming the System Interface Unit** for more information. The SIUMCR is the only register from the MPC860 that has been altered to fit the requirements of the MPC801.

12.3.1 Configuring the Interrupt

An overview of the MPC801 interrupt structure is illustrated in Figure 12-2. The system interface unit receives interrupts from internal sources, like the periodic interrupt timer or real-time clock, and from external pins $\overline{IRQ}$[0:7].

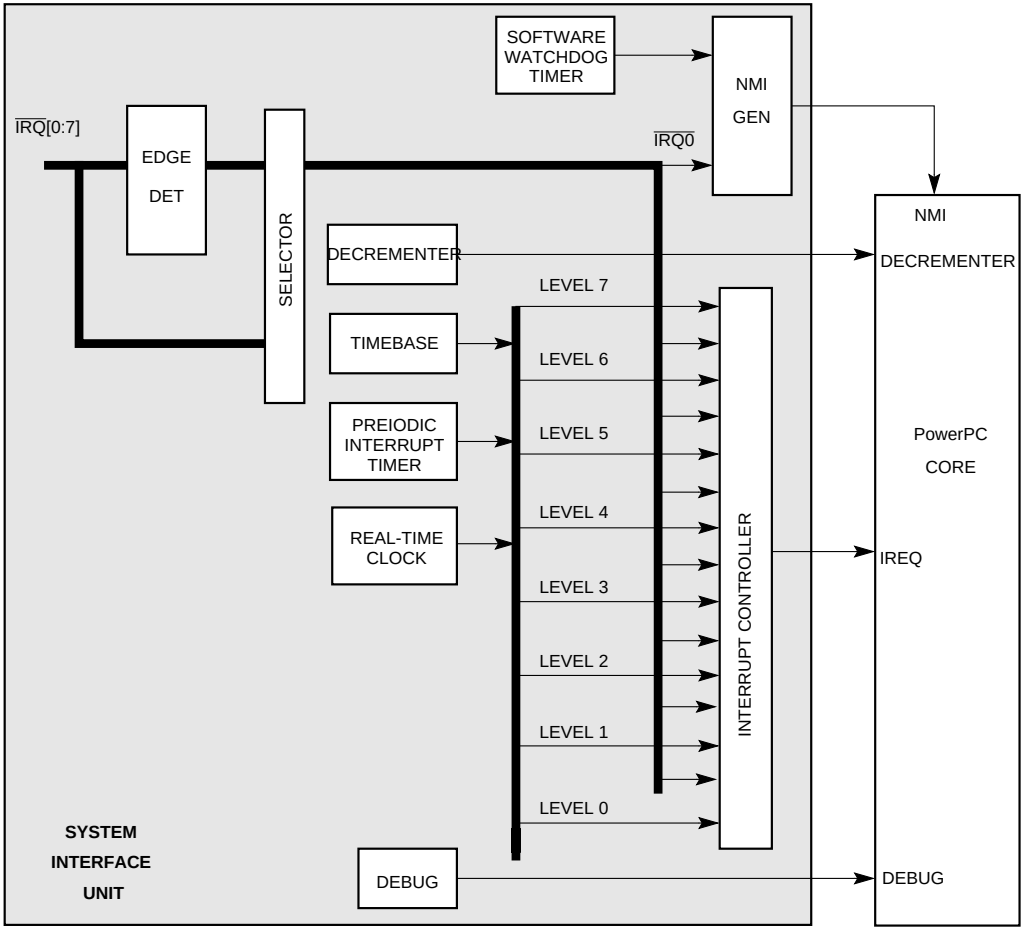


Figure 12-2. MPC801 Interrupt Structure

If it is programmed to generate an interrupt, the software watchdog timer always generates a nonmaskable interrupt (NMI) to the core. The external $\overline{\text{IRQ0}}$ pin can generate an NMI as well. The core takes the system reset interrupt when an NMI is asserted and the external interrupt when any other interrupt is asserted by the interrupt controller. Each one of the external $\overline{\text{IRQ}}$ pins has its own dedicated assigned priority level and there are eight additional interrupt priority levels. Each one of the system interface unit internal interrupt sources, the interrupt request which is generated by the CPM interrupt controller, can be assigned by the software to any one of those eight interrupt priority levels. Thus, you get a very flexible interrupt scheme.

12.3.2 Priority of the Interrupt Sources

The system interface unit has 15 interrupt sources that assert just one interrupt request to the PowerPC core. There are seven external $\overline{\text{IRQ}}$ pins (the eighth one generates an NMI) and eight interrupt levels. The priority between all interrupt sources is shown in Table 12-1.

Table 12-1. Priority of System Interface Unit Interrupt Sources

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

12.3.2.1 PROGRAMMING THE INTERRUPT CONTROLLER. The system interface unit interrupt controller contains the SIPEND, SIMASK, SIEL, and SIVEC registers.

12.3.2.1.1 SIU Interrupt Pend Register. The 32-bit SIU interrupt pend (SIPEND) register contains bits that individually correspond to an interrupt request. The bits associated with internal exceptions indicate, if set, that an interrupt service is requested. These bits reflect the status of the internal requestor device and are cleared when the appropriate actions are software-initiated in the device. 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 $\overline{\text{IRQ}}$ pin is defined as a “level” interrupt the corresponding bit behaves similar to the bits associated with internal interrupt sources. When the $\overline{\text{IRQ}}$ pin is defined as an “edge” interrupt and if the corresponding bit is set, it indicates that a falling edge was detected on the signal. This bit is reset 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	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
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															

IRQ—Interrupt Request 0–7

These bits reflect the status of the enternal interrupt requests.

0 = The appropriate interrupt service is not requested.
1 = The appropriate interrupt service is requested.

LVL—Level 0–7

These bits reflect the status of the internal requestor device.

0 = The appropriate interrupt service is not requested.
1 = The appropriate interrupt service is requested.

Bits 16–31—Reserved

These bits are reserved and should be set to 0.

12.3.2.2 SIU INTERRUPT MASK REGISTER. The 32-bit read/write SIU interrupt mask (SIMASK) register contains bits that individually correspond to the interrupt request bits in the SIPEND register. When a bit is set, it enables the generation of an interrupt request to the core. SIMASK is updated by the software and cleared at reset. It is the responsibility of the software to determine which of the interrupt sources are enabled at a given time.

SIMASK

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	IRM0	LVM0	IRM1	LVM1	IRM2	LVM2	IRM3	LVM3	IRM4	LVM4	IRM5	LVM5	IRM6	LVM6	IRM7	LVM7
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
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															

IRM—Interrupt Mask

These bits are used as mask bits to both internal and external interrupt requests.

0 = The appropriate interrupt service is not requested.
1 = The appropriate interrupt service is requested.

LVM—Level Mask

These bits...

0 = The appropriate interrupt service is not requested.
1 = The appropriate interrupt service is requested.

Bits 16–31—Reserved

These bits are reserved and should be set to 0.

12.3.2.3 SIU INTERRUPT EDGE LEVEL MASK REGISTER. The 32-bit read/write SIU interrupt edge level mask (SIEL) register contains pairs of bits that individually correspond to an external interrupt request. The ED*x* bit, if set, specifies that a falling edge in the corresponding $\overline{\text{IRQ}}$ signal is detected as an interrupt request. When the ED*x* bit is zero, a low logical level in the $\overline{\text{IRQ}}$ signal is detected as an interrupt request. The WM*x* bit, if set, indicates that a low-level detection in the corresponding interrupt request line causes the MPC801 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

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															

A couple of ED bits with the appropriate WM bits specifies the external interrupt request generation.

ED

This bit specifies:

0 = Level.
1 = Falling edge.

WM

This bit is specified when a high or low level interrupt generation causes an interrupt.

Bits 16–31—Reserved

These bits are reserved and should be set to 0.

12.3.2.4 SIU INTERRUPT VECTOR REGISTER. The 32-bit read-only SIU interrupt vector (SIVEC) register contains an 8-bit code representing an unmasked interrupt source of the highest priority level. The SIVEC register can be read as either a byte, half, or word. When read as a byte, a branch table can be used in which each entry contains one instruction. When read as a half-word, each entry can contain a full routine with a maximum of 256 instructions. The interrupt code is extended with two zero least-significant bits to allow the table to be indexed. 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								RESERVED							
R/W	R								R							
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
R/W	R															

INTC—Interrupt Code

These bits represent the index bits of the current highest unmasked interrupt request routine.

Bits 8–31—Reserved

These bits are reserved and should be set to 0.

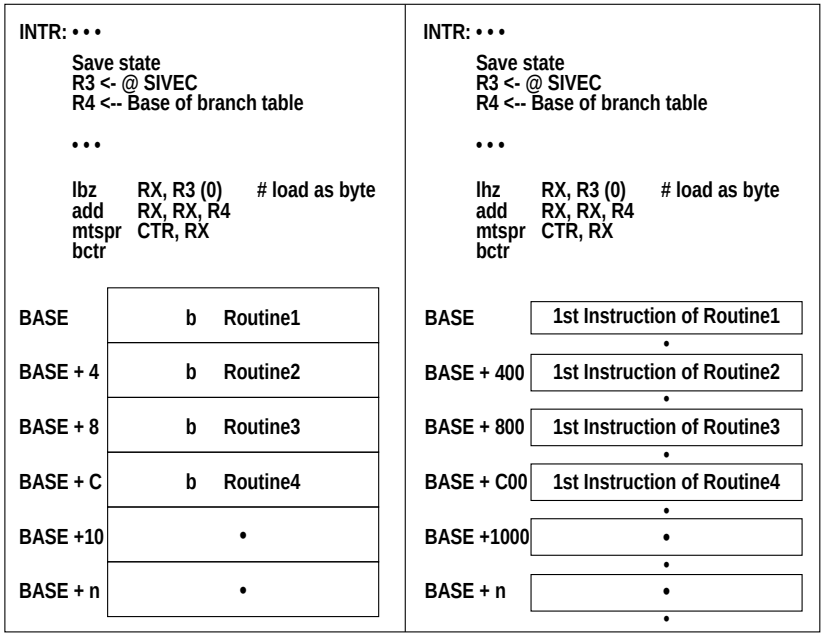


Figure 12-3. Interrupt Table Handling Example

12.4 THE BUS MONITOR

The bus monitor ensures that each bus cycle is terminated within a reasonable period of time. The MPC801 system interface unit provides a bus monitor option that monitors internal and 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, a $\overline{\text{TEA}}$ signal 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.

12.5 THE POWERPC DECREMENTER

The 32-bit PowerPC decrementing counter (DEC) provides a decrementer interrupt. This binary counter is clocked by the same frequency as the timebase. In the MPC801, the decrementer is clocked by the tmbclk clock.

$$T_{dec} = \frac{2^{32}}{(F_{tmbclk})}$$

The state of the decrementer is not affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ and, therefore, should be initialized by the software. The decrementer runs continuously after power-up. It continues counting while $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ are asserted and it is implemented with the following requirements in mind. The decrementer interrupt is also sent to the power-down wake-up logic, thus allowing a pin to be toggled while the rest of the MPC801 is not running.

- The operation of the timebase and decrementer are coherent, which means the counters are driven by the same fundamental timebase.
- The decrementer remains unaffected when it is loading or unloading.
- When storing to the decrementer, the value in the decrementer is replaced with the value in the general-purpose register.
- When Bit 0 (the most-significant bit) of the decrementer changes from 0 to 1, an interrupt request is signaled. If multiple decrementer interrupt requests are received before the first one is reported, only one interrupt is reported.
- If the decrementer is altered by the software and the content of Bit 0 is changed from 0 to 1, an interrupt request is signaled.

A decrementer exception causes a pending decrementer interrupt request in the core. When the decrementer interrupt is taken, the request is automatically cleared. The following chart shows some of the periods available for the decrementer, assuming a 4MHz 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

12.6 THE POWERPC TIMEBASE

The timebase (TB) is a PowerPC architecture defined timer facility. It is a 64-bit free-running binary counter that is incremented at a frequency determined by each implementation of the timebase. There is no interrupt or other indication generated when the count rolls over. The timebase period depends on the driving frequency. For the MPC801, the timebase is clocked by the tmbclk clock and the timebase period is:

$$T_{TB} = \frac{2^{64}}{F_{tmbclk}}$$

The state of the timebase is unaffected by any resets and should be initialized by the software. Reads and writes of the timebase are restricted to special instructions. For the MPC801 implementation, it is not possible to read or write the entire timebase in a single instruction. Therefore, the **mttb** and **mftb** instructions are used to move the lower half of the timebase (TBL) while the **mttbu** and **mftbu** instructions are used to move the upper half of the timebase (TBU). The timebase has four reference registers associated with it. A maskable interrupt is generated when the timebase count reaches the value programmed in one of the four reference registers and the two status bits indicate which of the four reference registers generated the interrupt.

12.7 THE REAL-TIME CLOCK

The real-time clock (RTC) is a 45-bit counter that is clocked by the pitrtclk clock. It is used to provide time-of-day indication to the operating system and application software. The counter is not affected by reset and operates in all low-power modes. It is initialized by the software. The real-time clock can be programmed to generate a maskable interrupt when the time value matches the value programmed in the associated alarm register. It can also be programmed to generate an interrupt once every second. A control and status register is used to enable or disable the different functions and report the interrupt source. The real-time clock related registers are “locked” after power-on reset. To enable a write action to any of these registers, a previously open operation should be taken. For more information refer to **Section 5.10.2 The Key Mechanism**.

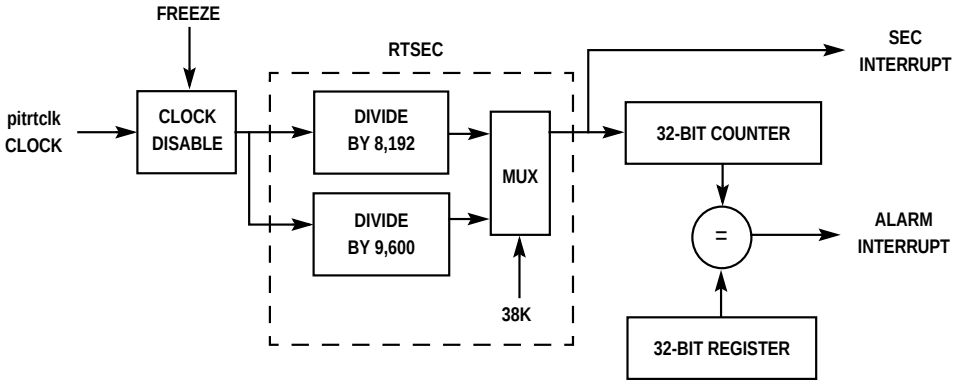


Figure 12-4. RTC Block Diagram

12.8 THE PERIODIC INTERRUPT TIMER

The periodic interrupt timer (PIT) consists of a 16-bit counter clocked by a pitrtclk clock supplied by the clock module. It decrements to zero when loaded with a value from the periodic interrupt timer count (PITC) register and after the timer reaches zero, the PS bit is set and an interrupt is generated if the PIE bit is a logic 1. At the next input clock edge, the value in the PITC register is loaded into the counter and the process starts all over again. When a new value is loaded into the PITC register, the periodic interrupt timer is updated, the divider is reset, and the counter starts counting. If the PS bit is not cleared, it generates an interrupt at the interrupt controller and the interrupt remains pending until it is cleared. If the PS bit is set again, prior to being cleared, the interrupt remains pending until the PS bit is cleared. Any write to the PITC register stops the current countdown and the count resumes with a new value in the PITC. If the PTE bit is not set, the periodic interrupt timer is unable to count and retains the old count value. Reads of the periodic interrupt timer have no effect on it.

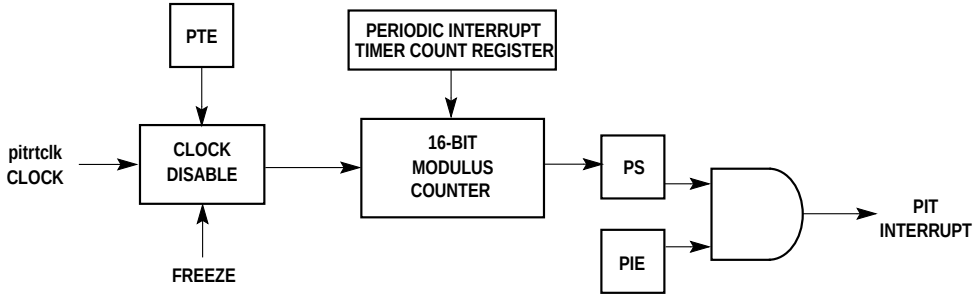


Figure 12-5. Periodic Interrupt Timer Block Diagram

The timeout period is calculated as:

$$PIT_{period} = \frac{PITC + 1}{F_{pitrtclk}} = \frac{PITC + 1}{\left(\frac{ExternalClock}{1 or 128}\right) \div 4}$$

Solving this equation using a 32.768KHz external clock gives:

$$PITperiod = \frac{PITC + 1}{8192}$$

This provides a range from 122 microseconds with a PITC register of $0000, to 8 seconds with a PITC of $FFFF.

12.9 THE SOFTWARE WATCHDOG TIMER

The software watchdog timer option prevents system lockout if the software becomes trapped in loops without a controlled exit. The software watchdog timer is enabled after system reset to cause a system reset if it times out. If you do not need the software watchdog timer, the SWE bit in the system protection control register (SYPCR) must be cleared to disable it. If you do need it, the software watchdog timer requires a special service sequence to be executed on a periodic basis. If this periodic servicing does not occur, the software watchdog timer times out and issues a reset or a nonmaskable interrupt, which is programmed by the SWRI bit in the SYPCR register. Once the software writes the SYPCR register, the state of the SWE bit cannot be changed. Refer to the system configuration and protection registers for more information.

To service the software watchdog timer, write $556C and then $AA39 to the software service register (SWSR). This 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 be repeated. Although the writes must occur in the correct order before a timeout occurs, any number of instructions can be executed between the writes. This allows interrupts and exceptions to occur between the two writes when necessary. Refer to Figure 12-6 for more information.

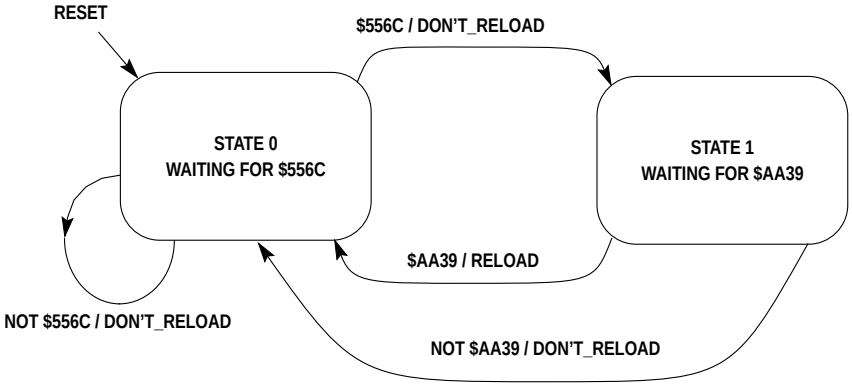


Figure 12-6. Software Watchdog Timer Service State Diagram

Although most software disciplines permit or encourage the watchdog concept, some systems require a selection of timeout periods. For this reason, the software watchdog timer must provide a selectable range for the timeout period. Figure 12-7 illustrates the present method for handling this need. In Figure 12-7 also shows the range that the value SWTC field determines. The value held in the SWTC field is then loaded into a 16-bit decrementer clocked by the system clock. When necessary, an additional divide by 2,048 prescaler is used.

The decrementer begins counting when loaded it is 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. At reset, the value in the SWTC field is set to the maximum value and is again loaded into the software watchdog register, thus starting the process all over again. When a new value is loaded into the SWTC register, the software watchdog timer will not be updated until the servicing sequence is written to the SWSR. If the SWE bit is loaded with the 0 value, the modulus counter will not count.

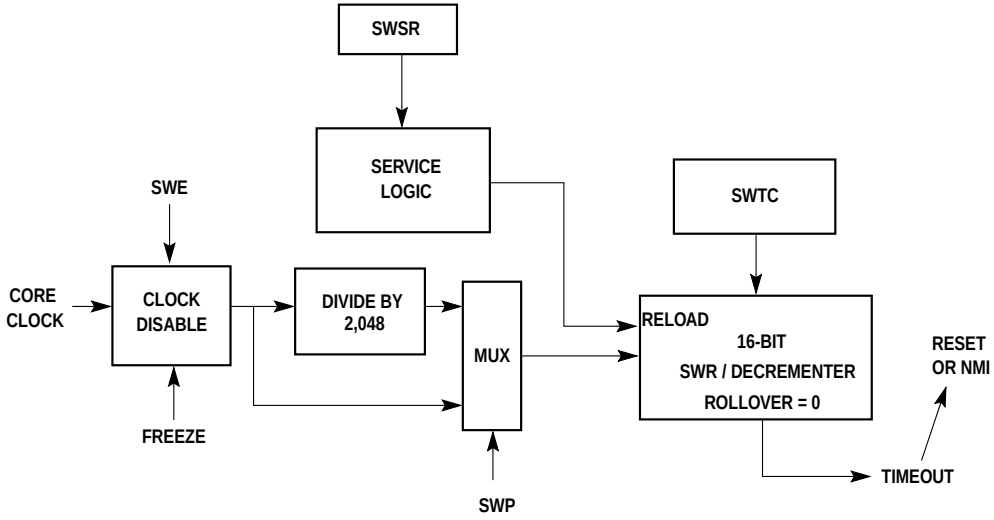


Figure 12-7. Software Watchdog Timer Block Diagram

12.10 FREEZE OPERATION

When the freeze signal is asserted, the clocks to the software watchdog, periodic interrupt timer, real-time clock, timebase counter, and decrementer can be disabled. This is controlled by the associated bits in the control register of each timer. If they are programmed to stop during freeze, the counters maintain their values while freeze is asserted, unless that is changed by the software. The bus monitor is enabled, regardless of this signal.

12.10.1 Low-Power Stop Operation

When the PowerPC core is set to low-power mode, the software watchdog timer is frozen. It remains frozen and maintains its count value until the core exits this mode and continues to execute instructions. The periodic interrupt timer, decrementer, or timebase are not influenced by these low-power modes and they continue to run at their respective frequencies. These timers can generate an interrupt to bring the MPC801 out of the low-power modes.

12.11 MULTIPLEXING THE SYSTEM INTERFACE UNIT PINS

Due to the limited number of pins available in the MPC801 package, some of the functionalities defined in the various sections share pins. The actual MPC801 pinout is illustrated in **Section 2 External Signals**. The following table shows how the functionality is controlled on each pin.

Table 12-2. Multiplexing Control

PIN NAME	PIN CONFIGURATION CONTROL
BDIP/GPLB5 RSV/IRQ2 CR/IRQ3 KR/IRQ4 FRZ/IRQ6	Programmed in the SIUMCR.
WE0/BS_AB0 WE1/BS_AB1 WE2/BS_AB2 WE3/BS_AB3	Dynamically active depending on the machine (GPCM or UPMB) assigned to control the required slave.
GPLA0/GPLB0	Dynamically active depending on the machine (UPMA or UPMB) assigned to control the required slave.
OE/GPLA1/GPLB1	Dynamically active depending on the machine (GPCM, UPMA, or UPMB) assigned to control the required slave.
GPLA[2:3]/GPLB[2:3]/CS[2:3]	GPLA[2:3]/GPLB[2:3]: Dynamically active depending on the machine (UPMA or UPMB) assigned to control the required slave. GPLx[2:3]/CS[2:3]: Programmed in the SIUMCR.
IWP[0:1]/VFLS[0:1] IWP2/VF2 LWP0/VF0 LWP1/VF1 DSCK/AT1 PTR/AT3 TDI/DSDI IRQ5/DSDI TCK/DSCK TDO/DSDO	Programmed in the SIUMCR and hard reset configuration.
MODCK1/STS MODCK2/DSDO	At power-on reset, MODCK[1:2]. Otherwise, it is programmed in the SIUMCR and hard reset configuration.

12.12 PROGRAMMING THE SYSTEM INTERFACE UNIT

12.12.1 System Configuration and Protection Registers

12.12.1.1 SIU MODULE CONFIGURATION REGISTER. The SIU module configuration register (SIUMCR) contains bits that configure various features of the system interface unit module.

SIUMCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EARB	EARP			RESERVED				DSHW	DBGC		DBPC		RES	FRC	DLK
RESET	0	0			0				0	0		0		0	0	0
R/W	R/W	R/W			R/W				R/W	R/W		R/W		R/W	R/W	R/W
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	OPAR	PNCS	DPC	MP RE	MLRC		AEME	SEME	RES	GB5E	B2DD	B3DD	RESERVED			
RESET	0	0	0	0	0		0	0	0	0	0	0	0			
R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W			

EARB—External Arbitration

If this bit is set (1), external arbitration is assumed. If it is cleared (0), internal arbitration is performed. For more information, refer to **Section 13.4.6 Arbitration Phase Signals**.

EARP—External Arbitration Request Priority

This field defines the priority of the external master's arbitration request. This field is valid when EARB is cleared. 000 is the lowest priority level and 111 is the highest. For more information, refer to **Section 13.4.6 Arbitration Phase Signals**.

Bits 4–7 and 13—Reserved

These bits are reserved and should be set to 0.

DSHW—Data Show Cycles

This bit selects the show cycle mode to be applied to data cycles. Instruction show cycles are programmed in the ICTRL register. Refer to **Section 18.5.2 Development Port Registers** for more information. This bit is locked by the DLK bit.

0 = Disable show cycles for all internal data cycles.
1 = Show address and data of all internal data cycles.

DBGC—Debug Pin Configuration

This field configures the debug pins' functionality.

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

10 = Reserved

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

DBPC—Debug Port Pins Configuration

This field configures the pins on which the development port is active.

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

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

10 = Reserved

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

FRC—FRZ Pin Configuration

This bit configures the functionality of the FRZ/$\overline{\text{IRQ6}}$ pin.

0 = FRZ/$\overline{\text{IRQ6}}$ functions as FRZ.
1 = FRZ/$\overline{\text{IRQ6}}$ functions as $\overline{\text{IRQ6}}$.

DLK—Debug Register Lock

If this bit is set (1), bits 8–15 are locked and writes to those bits can no longer be performed. Bits 8–15 can be written in test mode once the internal freeze is asserted, regardless of the state of DLK. This bit is cleared by reset.

OPAR—Odd Parity

This bit is used to program odd or even parity. It can also be used to generate parity errors for testing purposes by writing the memory with OPAR = 1 and reading the memory with OPAR = 0.

PNCS—Parity Enable For Nonmemory Controller Regions

This bit enables parity generation/checking for memory regions not controlled by the memory controller.

DPC—Data Parity Pins Configuration

This bit configures the functionality of the DP[0:3]/$\overline{\text{IRQ}}$[3:6] pins.

0 = DP[0:3]/$\overline{\text{IRQ}}$[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

This bit configures the functionality of the $\overline{\text{RSV}}$/$\overline{\text{IRQ2}}$ and $\overline{\text{CR}}$/$\overline{\text{IRQ3}}$ pins.

0 = $\overline{\text{RSV}}$/$\overline{\text{IRQ2}}$ and $\overline{\text{CR}}$/$\overline{\text{IRQ3}}$ function as $\overline{\text{RSV}}$ and $\overline{\text{CR}}$ accordingly. Reservation protocol is enabled.
1 = $\overline{\text{RSV}}$/$\overline{\text{IRQ2}}$ and $\overline{\text{CR}}$/$\overline{\text{IRQ3}}$ function as $\overline{\text{IRQ2}}$ and $\overline{\text{IRQ3}}$ accordingly. Reservation protocol is disabled.

MLRC—Multi-Level Reservation Control

This bit configures the functionality of the $\overline{\text{KR}}$/$\overline{\text{IRQ4}}$ pins.

00 = $\overline{\text{KR}}$/$\overline{\text{IRQ4}}$ functions as $\overline{\text{IRQ4}}$.
01 = Reserved.
10 = $\overline{\text{KR}}$/$\overline{\text{RETRY}}$/$\overline{\text{IRQ4}}$ functions as $\overline{\text{KR}}$/$\overline{\text{RETRY}}$.
11 = Reserved.

AEME—Asynchronous External Master Enable

This bit configures how the memory controller refers to external asynchronous masters initiating a transaction. If this bit is set, the memory controller interprets any assertion on the $\overline{\text{AS}}$ pin as an external asynchronous master initiating a transaction. If it is 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 this bit is set, the memory controller interprets any $\overline{TS}$ assertion that the external bus does not own as an external synchronous master initiating a transaction. If it is reset, the memory controller ignores the value of the $\overline{TS}$ pin when it does not own the external bus.

Bits 24 and 28–31—Reserved

These bits are reserved and should be set to 0.

GB5E—GPLB(5) Enable

If this bit is set (1), then the $\overline{GPLB5}$ of the memory controller functionality is active. If it is cleared (0), the $\overline{BDIP}$ signal functionality is active.

B2DD—Bank 2 Double Drive

If this bit is set (1), then the $\overline{CS2}$ signal is reflected onto $\overline{GPLA2}$/$\overline{GPLB2}$.

B3DD—Bank 3 Double Drive

If this bit is set (1), then the $\overline{CS3}$ signal is reflected onto $\overline{GPLA3}$/$\overline{GPLB3}$.

12.12.1.2 INTERNAL MEMORY MAP REGISTER. The internal memory map register (IMMR) is located within the PowerPC special register space. It contains the identification of a specific device as well as a base for the internal memory map. Based on the value read from this register, the software can deduce the availability and location of any on-chip system resource. The contents of this register can be read by the **mfspr** instruction and the ISB field can be written by the **mtspr** instruction. However, the PARTNUM and MASKNUM fields are mask programmed and cannot be changed for any device.

IMMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	ISB															
RESET	0															
R/W	R/W															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	PARTNUM								MASKNUM							
RESET	10								00							
R/W	R								R							

ISB—Internal Space Base

This read/write field defines the base address of the internal memory space. The initial value of this field can be configured at reset to one of four addresses and changed to any value by the software. The number of programmable bits in this field and the resolution of the location of internal space depends on the internal memory space of the specific implementation.

In the MPC801, 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 system interface unit is located. It is intended to help factory test and user code that is sensitive to part refinements. This field changes as the part number changes. For example, it would change if a new module is added or if the size of the memory module is revised. However, it would not change if the part is revised to fix a bug in an existing module. The MPC801 has an ID of $10.

MASKNUM—Mask Number

This read-only field is mask programmed with a code corresponding to the mask number of the part on which the system interface unit is located. It is intended to help factory test and user code that is sensitive to part refinements. It is programmed in a frequently changed layer and should be revised for all mask set modifications. The MPC801 Rev 0 has an ID of $00.

12.12.1.3 SYSTEM PROTECTION CONTROL REGISTER. The system protection control register (SYPCR) controls the system monitors, software watchdog period, and bus monitor timing. This register can be read at any time, but can only be written once after system reset.

SYPCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SWTC															
RESET	1															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	BMT								BME	RESERVED			SWF	SWE	SWRI	SWP
RESET	1								0	0			0	1	1	1

SWTC—Software Watchdog Timer Count

This field contains the count value for the software watchdog timer.

BMT—Bus Monitor Timing

This field defines the timeout period, in 8 system clock resolution, for the bus monitor.

BME—Bus Monitor Enable

This bit controls the operation of the bus monitor when an internal to external bus cycle is executed.

0 = Disable the bus monitor.
1 = Enable the bus monitor.

Bits 25–27—Reserved

These bits are reserved and should be set to 0.

SWF—Software Watchdog Freeze

If this bit is asserted (1), the software watchdog timer stops when freeze indicator is asserted (debug mode). Otherwise, it is free running.

SWE—Software Watchdog Enable

This bit enables the software watchdog timer. To disable the software watchdog timer, it should be cleared by the software after a system reset.

SWRI—Software Watchdog Reset/Interrupt Select

When this bit is cleared, the software watchdog timer causes a nonmaskable interrupt to the core. When it is set, the software watchdog timer causes a system reset.

SWP—Software Watchdog Prescale

This bit controls the divide-by-2,048 software watchdog timer prescaler. If it is cleared, the software watchdog timer is not prescaled and if it is set, the software watchdog timer clock is prescaled.

12.12.1.4 SOFTWARE SERVICE REGISTER. The software service register (SWSR) is the location the software watchdog timer servicing sequence writes to. To prevent a software watchdog timer timeout, a write of $556C followed by $AA93 should be written to this register. The SWSR can be written at any time, but returns all zeros when read.

12.12.1.5 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. Canceled speculative accesses that do not cause an interrupt allow these bits to be set. The register has two identical sets of bit fields–one is associated with instruction transfers and the other with data transfers.

TESR

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

Bits 0–17 and 24–25—Reserved

These bits are reserved and should be set to 0.

IEXT—Instruction External Transfer Error Acknowledge

This bit is set if the cycle is terminated by an externally generated $\overline{\text{TEA}}$ signal when an instruction fetch is initiated.

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.

IPB—Instruction Parity Error on Byte

There are four parity error status bits for each 8-bit lane. One of these is set for the byte that had a parity error when an instruction was fetched. Parity check for a memory region that is not under memory controller control is enabled by the PNCS bit in SIUMCR.

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.

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.

DPB—Data Parity Error On Byte

There are four parity error status bits for each 8-bit lane. One of these is set for the byte that had a parity error when a data load was requested by an internal master. Parity check for a memory region that is not controlled by the memory controller is enabled by the PNCS bit in the SIUMCR.

12.12.2 System Timer Registers

System timers are powered by keep alive power, which preserves their value when the main power supply is off. Refer to **Section 5.10.2 The Key Mechanism** for details on the required actions needed to guarantee data retention.

12.12.2.1 DECREMENTER REGISTER. The 32-bit PowerPC decrementer (DEC) register contains values that the down counter uses to cause decrementer interrupts. The decrementer causes an interrupt whenever Bit 0 changes from logic 0 to logic 1. A read of this register always returns the current count value from the down counter. The contents of this register can be read or written to by a **mfspr** or **mtspr** instruction. This register is not affected by reset. It uses standby power and continues counting when standby power is applied.

DEC

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

12.12.2.2 TIMEBASE REGISTER. The 64-bit PowerPC timebase (TB) register contains a 64-bit integer that is periodically incremented. Since there is no automatic initialization of the TB register, the system software must perform the initialization. The contents of the register can be written by a **mttbl** or **mttbu** instruction.

Table 12-3. 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.

12.12.2.3 TIMEBASE REFERENCE REGISTERS. There are two 32-bit, read/write timebase reference registers—TBREFF0 and TBREFF1—associated with the lower part of the timebase. When there is a match between the contents of the timebase and reference registers, a maskable interrupt is generated.

TBREFF0 AND TBREFF1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TBREFF0															
RESET	1															
R/W	R/W															
ADDR	204															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TBREFF1															
RESET																
R/W	R/W															
ADDR	208															

12.12.2.4 TIMEBASE CONTROL AND STATUS REGISTER. The 16-bit, read/write timebase control and status register (TBSCR) controls timebase count enable and interrupt generation. It is also used to report interrupt sources. A status bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. This register can be read at any time.

TBSCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TBIRQ								REFA	REFB	RESERVED		REFAE	REFBE	TBF	TBE
RESET	0								0	0	0		0	0	0	0
R/W	R/W								R/W	R/W	R/W		R/W	R/W	R/W	R/W

TBIRQ—Timebase Interrupt Request

This field determines the interrupt priority level of the timebase. To specify a certain level, the appropriate bit should be set. Refer to **Section 12.3.1 Configuring the Interrupt** for more information.

REFA and REFB—Reference Interrupt Status A and B

If set, these bits indicate that a match has been detected between the corresponding reference register (TBREFF0 for REFA and TBREFF1 for REFB) and the timebase low register. The bit can be cleared by writing a 1.

Bits 10–11—Reserved

These bits are reserved and should be set to 0.

REFAE and REFBE—Reference Interrupt Enable A and B

If these bits are asserted (1), the timebase generates an interrupt when the REFA or REFB bits are asserted. Otherwise, interrupt is disabled.

TBF—Timebase Freeze

If this bit is asserted (1), the timebase and decrementer stops while freeze is asserted.

TBE—Timebase Enable

If this bit is asserted (1), the timebase and decrementer are enabled.

12.12.2.5 REAL-TIME CLOCK STATUS AND CONTROL REGISTER. The real-time clock status and control register (RTCSC) is used to enable the different real-time clock functions and to report interrupt sources. A status bit is cleared by writing a 1 (writing a zero has no effect) and more than one status bit can be cleared at a time. This register can be read at any time.

RTCSC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RTCIRQ								SEC	ALR	RES	38K	SIE	ALE	RTF	RTE
RESET	0								0	0	0		0	0	0	
R/W	R/W								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

RTCIRQ—Real-Time Clock Interrupt Request

This field controls the real-time clock interrupt priority level. Refer to **Section 12.3.1 Configuring the Interrupt** 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 real-time clock 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 32.768KHz to generate the second pulse and if it is asserted, the real-time clock assumes 38.4KHz. 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

If this bit is set, the real-time clock timers are enabled. This bit is not affected by reset.

12.12.2.6 REAL-TIME CLOCK REGISTER. The 32-bit read/write real-time clock (RTC) register contains the current value of the real-time clock.

RTC

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

12.12.2.7 REAL-TIME CLOCK ALARM REGISTER. When the 32-bit read/write real-time clock alarm (RTCAL) register has a value equal to the value programmed in the alarm register, a maskable interrupt is generated.

RTCAL

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

ALARM

The alarm interrupt will be generated as soon as there is a match between the RTCAL and RTC register value. The resolution of the alarm is 1 second.

12.12.2.8 PERIODIC INTERUPT STATUS AND CONTROL REGISTER. The read/write periodic interrupt status and control register (PISCR) contains the interrupt request level and interrupt status bits. It also controls the 16 bits to be loaded into a modulus counter.

PISCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PIRQ								PS	RESERVED				PIE	PITF	PTE
RESET	0								0	0				0	0	1
R/W	R/W								R/W	R/W				R/W	R/W	R/W

PIRQ—Periodic Interrupt Request Level

This field determines the interrupt request level that will be asserted when a periodic interrupt is generated.

PS—Periodic Interrupt Status

This bit is asserted if the periodic interrupt timer issues an interrupt. An interrupt can be issued after the modulus counter counts to zero. This bit is negated by writing a 1 (zero has no effect).

Bits 9–12—Reserved

These bits are reserved and should be set to 0.

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 periodic interrupt timer counting. When the timer is disabled, it maintains its old value. When the counter is enabled, it continues counting using the previous value.

0 = Disable counter.
1 = Enable counter.

12.12.2.9 PERIODIC INTERRUPT TIMER COUNT REGISTER. The read/write periodic interrupt timer count (PITC) register contains the 16 bits that are to be loaded in a modulus counter.

PITC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PITC															
R/W	R/W															

PITC—Periodic Interrupt Timing Count

This field contains the periodic timer count. If it is loaded with a $FFFF value, the maximum count period is selected.

12.12.2.10 PERIODIC INTERRUPT TIMER REGISTER. The read-only periodic interrupt timer register (PITR) 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 effect on the counter.

PITR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PIT															
R/W	R															

PIT—Periodic Interrupt Timing Count

This field contains the current count remaining in the periodic timer. Writes have no effect on this field.

SECTION 13
EXTERNAL BUS INTERFACE

The MPC801 bus is synchronous, burstable, and supports multiple masters. Signals driven on this bus are required to make the setup and hold time relative to the bus clock's rising edge. The bus's MPC801 architecture supports byte, half-word, and word operands that allow access to 8-,16-, and 32-bit data ports by using synchronous cycles controlled by the size outputs. Accesses to 16- and 8-bit ports are made for slaves controlled by the memory controller.

13.1 FEATURES

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

- 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 Asynchronous and Burstable Memory Types
- Asynchronous DRAM Interface Support
- Flash ROM Programming Support
- Compatible with PowerPC™ Architecture
- Simple Interface to Slave Devices
- Bus is Synchronous (All Signals are Referenced to the Rising Edge of Bus Clock)
- Data Parity Support

13.2 TRANSFER SIGNALS

The bus transfers information between the MPC801 and the external memory or peripheral device. External devices can accept or provide 8,16, and 32 bits in parallel and must follow the handshake protocol described later in this section. The maximum number of bits accepted or provided during a bus transfer is defined as the port width.

The MPC801 contains an address bus that specifies the transfer's address and a data bus that transfers the data. Control signals indicate the beginning and type of the cycle, as well as the address space and size of the transfer. The selected device then controls the length of the cycle with the signal(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 MPC801 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 MPC801 latches the level of the input during a sample window around the rising edge of the clock signal. This window is illustrated in Figure 13-1, where tsu and tho are the input setup and hold times, respectively. To ensure that an input signal is recognized on a specific falling edge of the clock, the input must be stable during the sample window. If an input makes a transition during the window time period, the level recognized by the MPC801 is not predictable. However, the MPC801 always resolves the latched level to either a logic high or low before using it. In addition to meeting input setup and hold times for deterministic operation, all input signals must obey the protocols described in this section.

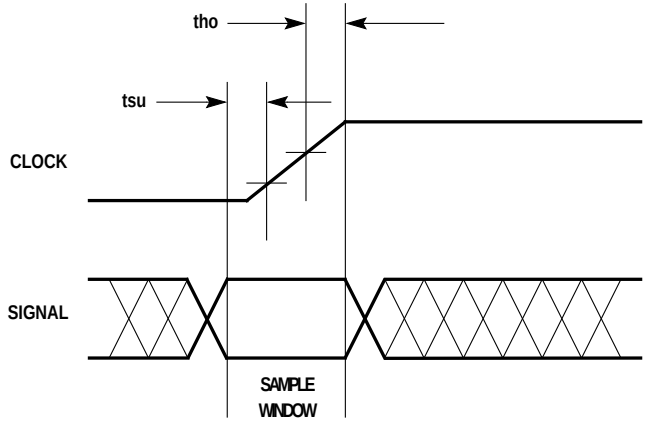


Figure 13-1. Input Sample Window

13.2.1 Control Signals

The MPC801 initiates a bus cycle by driving the address, size, address type, cycle type, and read/write outputs. At the beginning of a bus cycle, the TSIZ0 and TSIZ1 signals are driven with the AT signals. TSIZ0 and TSIZ1 indicate the number of bytes to be transferred during an operand cycle that consists of one or more bus cycles. These signals are valid at the rising edge of the clock in which the $\overline{TS}$ signal is asserted. The RD/$\overline{WR}$ signal determines the direction of the transfer during a bus cycle. Driven at the beginning of a bus cycle, RD/$\overline{WR}$ is valid at the rising edge of the clock in which the transfer start signal is asserted. However, RD/$\overline{WR}$ only transitions when a write cycle is preceded by a read cycle or vice versa. The signal may remain low for consecutive write cycles.

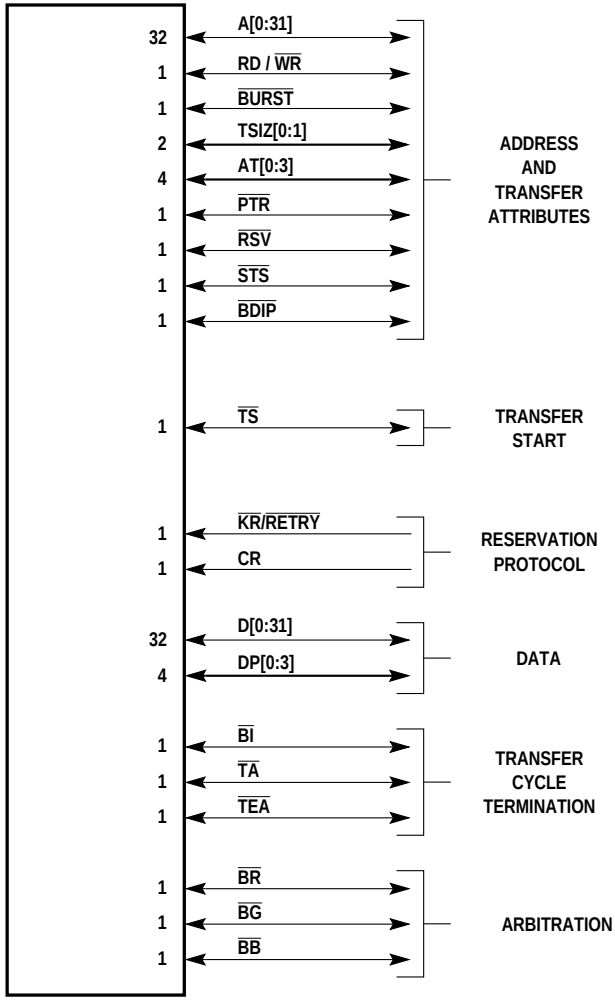


Figure 13-2. MPC801 Bus Signals

13.3 SIGNAL DESCRIPTIONS

The following table describes each bus interface signal. More detailed descriptions of these signals can be found in subsequent sections of this manual.

Table 13-1. MPC801 System Interface Unit Signals

MNEMONIC	PINS	ACTIVE	I/O	DESCRIPTION
ADDRESS AND TRANSFER ATTRIBUTES				
A[6:31]	32	High	O	**Address Bus**—Driven by the MPC801 when it owns the external bus. It specifies the physical address of the bus transaction. These signals can change during a transaction when controlled by the memory controller.
			I	Used only for testing purposes. Sampled by the MPC801 when the external master initiates a transaction and the memory controller is configured to handle external masters.
RD/$\overline{WR}$	1	High	O	**Read/Write**—Driven by the MPC801 along with the address when it owns the external bus. Driven high indicates that a read access is in progress and driven low indicates that a write access is in progress.
			I	Used only for testing purposes. Sampled by the MPC801 when the external master initiates a transaction and the memory controller is configured to handle external masters.
$\overline{BURST}$	1	High	O	**Burst Transfer**—Driven by the MPC801 along with the address when it "owns" the external bus. Driven low indicates that a burst transfer is in progress and driven high indicates that the current transfer is not a burst.
			I	Used only for testing purposes. Sampled by the MPC801 when the external master initiates a transaction and the memory controller is configured to handle external masters.
TSIZ[0:1]	2	High	O	**Transfer Size**—Driven by the MPC801 along with the address when it owns the external bus. It specifies the data transfer size for the transaction.
			I	Used only for testing purposes. Sampled by the MPC801 when the external master initiates a transaction and the memory controller is configured to handle external masters.
AT[0:3]	3	High	O	**Address Type**—Driven by the MPC801 along with the address when it owns the external bus. It provides additional information about the address on the current transaction.
			I	Used only for testing purposes.
$\overline{RSV}$	1	High	O	**Reservation Transfer**—Driven by the MPC801 along with the address when it owns the external bus. It provides additional information about the address on the current transaction.
			I	Used only for testing purposes.

Table 13-1. MPC801 System Interface Unit Signals (Continued)

MNEMONIC	PINS	ACTIVE	I/O	DESCRIPTION
$\overline{\text{PTR}}$	1	High	O	**Program Trace**—Driven by the MPC801 along with the address when it owns the external bus. It provides additional information about the address on the current transaction.
			I	Used only for testing purposes.
$\overline{\text{BDIP}}$	1	Low	O	**Burst Data In Progress**—Driven by the MPC801 when it owns the external bus. It is part of the burst protocol. Asserted indicates that the second beat in front of the current one is requested by the master. This signal is negated prior to the end of a burst to terminate the burst data phase early.
			I	Used only for testing purposes.
TRANSFER START				
$\overline{\text{TS}}$	1	Low	O	**Transfer Start**—Driven by the MPC801 when it owns the external bus.It indicates the start of a transaction on the external bus.
			I	$\overline{\text{TS}}$ is input for testing purposes. Sampled by the MPC801 when the external master initiates a transaction and the memory controller is configured to handle external masters.
$\overline{\text{STS}}$	1	Low	O	**Special Transfer Start**—Driven by the MPC801 when it owns the external bus. It indicates the start of a transaction on the external bus or an internal transaction in show cycle mode.
RESERVATION PROTOCOL				
CR	1	Low	I	**Cancel Reservation**—Each core has its own signal. If it is asserted, it instructs the bus master to clear its reservation. Some other master has touched its reserved space. This is a pulsed signal.
$\overline{\text{KR}}/\overline{\text{RETRY}}$	1	Low	I	**Kill Reservation/Retry**—When a bus cycle is initiated by a **stwcx** instruction that was issued by the core to a nonlocal bus on which the storage reservation has been lost, this signal is used by the nonlocal bus interface to back-off the cycle. Refer to **Section 13.4.9 Storage Reservation Protocol** for details. For a regular transaction, this signal is driven by the slave device to indicate that the MPC801 has to relinquish ownership of the bus and retry the cycle.

Table 13-1. MPC801 System Interface Unit Signals (Continued)

MNEMONIC	PINS	ACTIVE	I/O	DESCRIPTION
DATA				
D[0:31]	32	High		**Data Bus**—The data bus has the following byte lane assignments: Data Byte — Byte Lane D[0:7] — 0 D[8:15] — 1 D[16:23] — 2 D[24:31] — 3
			O	Driven by the MPC801 when it owns the external bus and has initiated a write transaction to a slave device. For single beat transactions, if A[30:31] and TSIZ[0:1] do not select the byte lanes for transfer, they will not supply valid data.
			I	Driven by the slave in a read transaction. For single beat transactions, if A[30:31] and TSIZ[0:1] do not select the byte lanes for transfer, they will not be sampled by the MPC801. Sampled by the MPC801 when the external master initiates a transaction and the memory controller is configured to handle external masters.
DP[0:3]	4	High		**Parity Bus**—Each parity signal corresponds to each one of the data bus lanes: Data Bus Byte — Parity Line D[0:7] — DP0 D[8:15] — DP1 D[16:23] — DP2 D[24:31] — DP3
			O	Driven by the MPC801 when it owns the external bus and has initiated a write transaction to a slave device. Each parity signal has the parity value (even or odd) of the corresponding data bus byte. For single beat transactions, if A[30:31] and TSIZ[0:1] do not select the byte lanes for transfer, they will not have a valid parity line.
			I	Driven by the slave in a read transaction. Each parity signal is sampled by the MPC801 and checked (if enabled) against the expected value parity value (even or odd) of the corresponding data bus byte. For single beat transactions, if A[30:31] and TSIZ[0:1] do not select the byte lanes for transfer, they will not be sampled by the MPC801 and its parity signals will not be checked.
TRANSFER CYCLE TERMINATION				
$\overline{TA}$	1	Low	I	**Transfer Acknowledge**—Driven by the slave device the current transaction was addressed to. It indicates that the slave has received the data on the write cycle or returned the data on the read cycle. If the transaction is a burst, $\overline{TA}$ should be asserted for each one of the transaction beats.
			O	Driven by the MPC801 when the slave device is controlled by the on-chip memory controller.

Table 13-1. MPC801 System Interface Unit Signals (Continued)

MNEMONIC	PINS	ACTIVE	I/O	DESCRIPTION
$\overline{TEA}$	1	Low	I	**Transfer Error Acknowledge**—Driven by the slave device the current transaction was addressed to. It indicates that an error condition has occurred during the bus cycle.
			O	Driven by the MPC801 when the internal bus monitor detects an erroneous bus condition.
$\overline{BI}$	1	Low	I	**Burst Inhibit**—Driven by the slave device the current transaction was addressed to. It indicates that the current slave does not support burst mode.
			O	Driven by the MPC801 when the slave device is controlled by the on-chip memory controller.
ARBITRATION				
$\overline{BR}$	1	Low	I	**Bus Request**—When the internal arbiter is asserted, it indicates that an external master is requesting the bus.
			O	Driven by the MPC801 when the internal arbiter is disabled and the chip is not parked.
$\overline{BG}$	1	Low	O	**Bus Grant**—When the internal arbiter is enabled, the MPC801 asserts this signal to indicate that an external master can assume ownership of the bus and begin a bus transaction. The $\overline{BG}$ signal should be qualified by the master requesting the bus to ensure it is the bus owner: Qualified $\overline{BG}$ = $\overline{BG}$ & ~ $\overline{BB}$
			I	When the internal arbiter is disabled, the $\overline{BG}$ is sampled and properly qualified by the MPC801 when an external bus transaction is to be executed by the chip.
$\overline{BB}$	1	Low	O	**Bus Busy**—When the internal arbiter is enabled, the MPC801 asserts this signal to indicate that it is the current owner of the bus. When the internal arbiter is disabled, it will assert this signal after the external arbiter grants the chip ownership of the bus and it is ready to start the transaction.
			I	When the internal arbiter is enabled, the MPC801 samples this signal to get an indication of when the external master ended its bus tenure ($\overline{BB}$ negated). When the internal arbiter is disabled, the $\overline{BB}$ is sampled to properly qualify the $\overline{BG}$ signal when an external bus transaction is to be executed by the chip.

NOTES:
O = Output from the MPC801.
I = Input to the MPC801.

13

13.4 OPERATIONS ON THE BUS

The MPC801 generates a system clock output (CLKOUT) signal that sets the frequency of operation for the bus interface. Internally, the MPC801 uses a phase-lock loop (PLL) circuit to generate a master clock for all of the core circuitry, including the bus interface that is phase-locked to the CLKOUT output signal.

All signals for the MPC801 bus interface are specified with respect to the rising-edge of the external CLKOUT signal and are guaranteed to be sampled as inputs or changed as outputs with respect to that edge. Since the same clock edge is referenced for driving or sampling the bus signals, the possibility of clock skew could exist between various modules in a system because of routing or using multiple clock lines. It is the responsibility of the system to handle any clock skew problems that could occur as a result of layout, lead-length, and physical routing.

13.4.1 Basic Transfer Protocol

The basic transfer protocol defines the sequence of actions that must occur on the MPC801 bus to perform a complete bus transaction. A simplified scheme of the basic transfer protocol is illustrated in Figure 13-3.

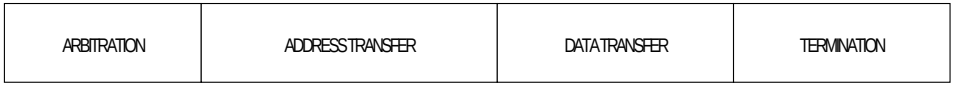


Figure 13-3. Basic Transfer Protocol

This protocol provides an arbitration phase and an address and data transfer phase. The arbitration phase specifies the master that initiates the next transaction. The address phase specifies the address for the transaction and the transfer attributes that describe the transaction. The data phase performs the transfer of data if any is to be transferred. It can transfer a single beat of data (4 bytes or less) for nonburst operations, a 4-beat burst of data (4×4 bytes), an 8-beat burst of data (8×2 bytes), or a 16-beat burst of data (16×1 bytes).

13.4.2 Single Beat Transfers

During the data transfer phase, 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 before the cycle following the address transfer phase. To avoid electrical contention, the master considers the “one dead clock cycle” when switching between drivers. The master can stop driving the data bus as soon as it samples the $\overline{TA}$ signal asserted on the rising edge of the CLKOUT signal. On a read cycle the master accepts the data bus contents as valid at the rising edge of the CLKOUT signal in which the $\overline{TA}$ signal is sample asserted.

13.4.2.1 SINGLE BEAT READ FLOW

The basic read cycle begins with a bus arbitration, followed by the address and data transfers. The handshakes as they apply to a fixed transaction protocol are illustrated in the following diagrams.

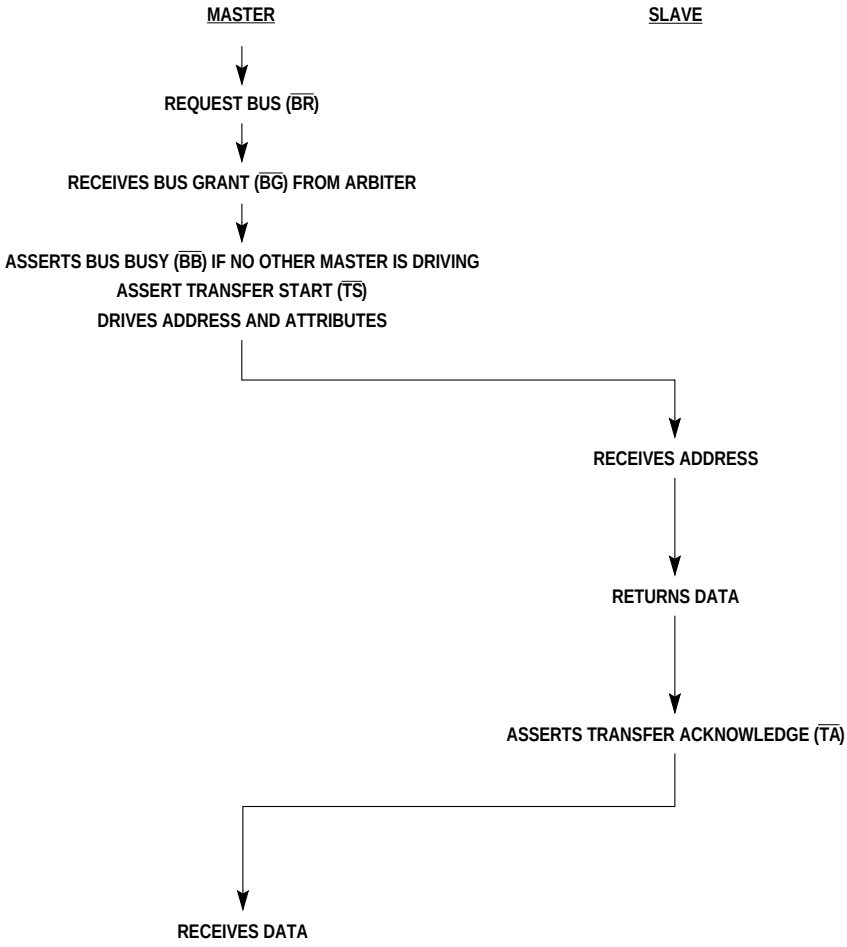


Figure 13-4. Simplified Diagram of a Single Beat Read Cycle

13

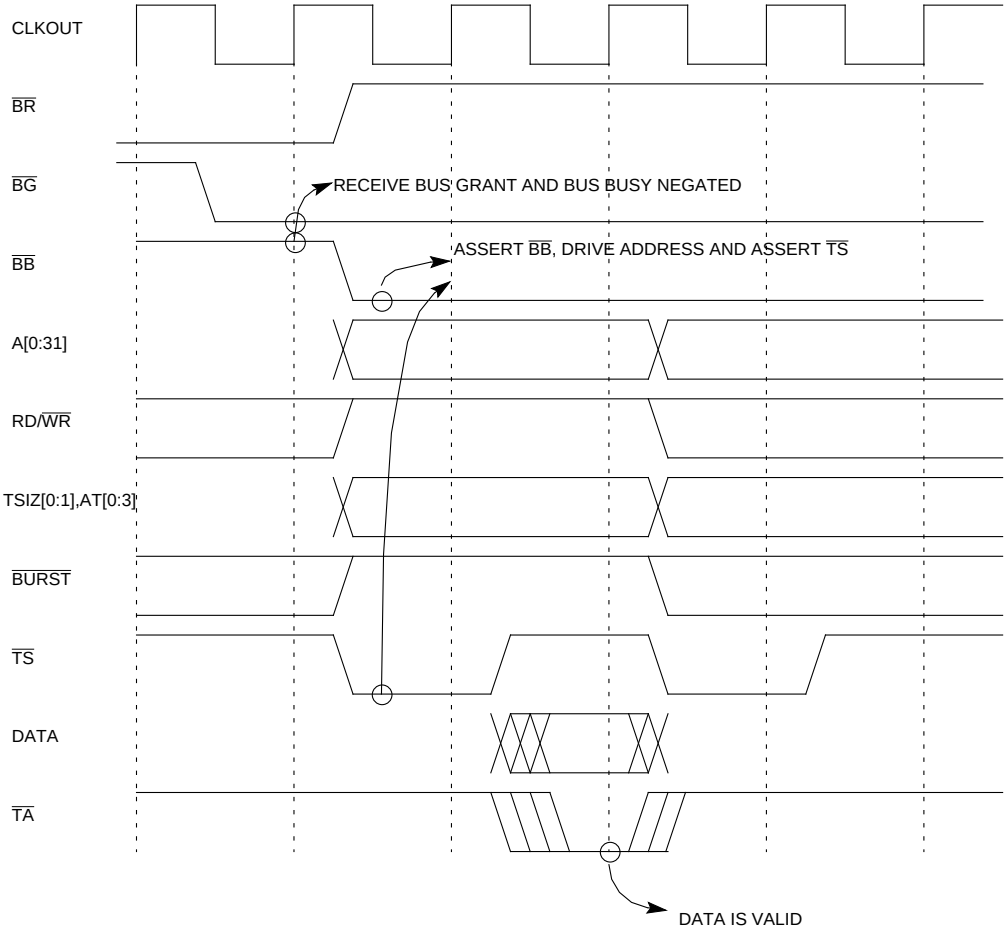


Figure 13-5. Single Beat Read Cycle–Basic Timing–Zero Wait States

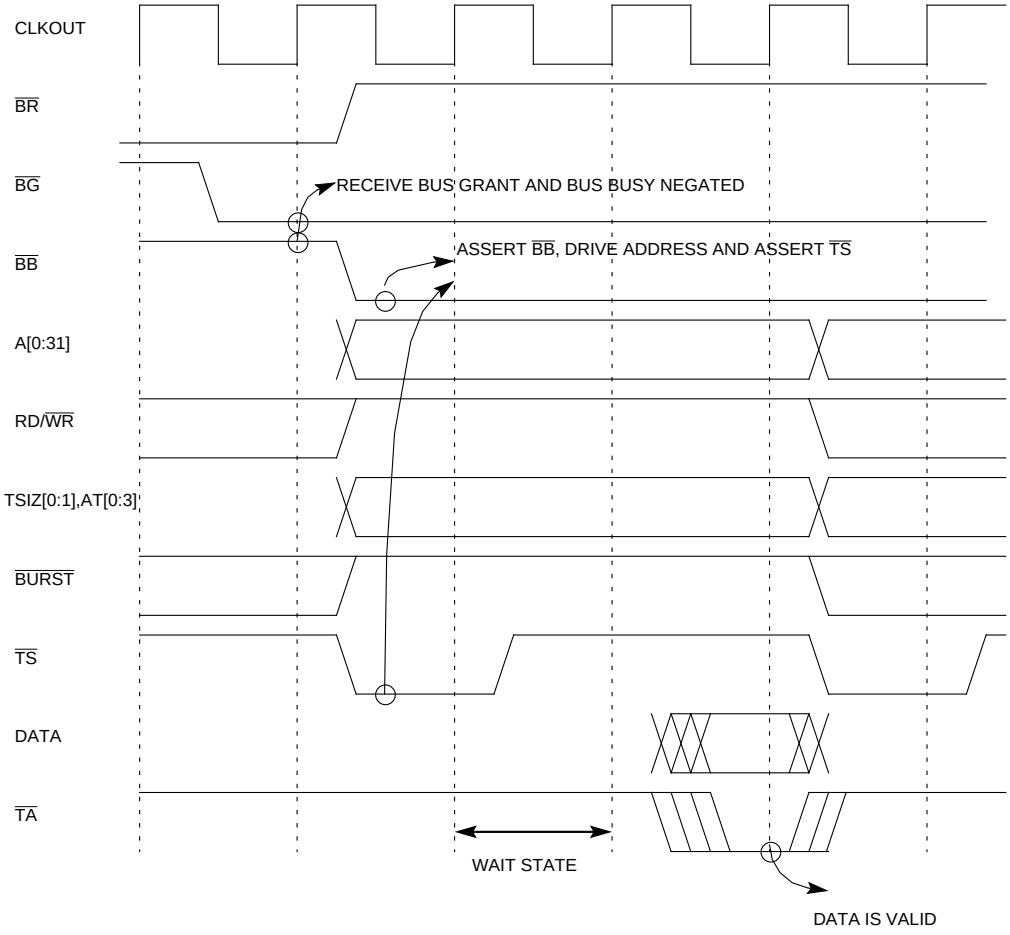


Figure 13-6. Single Beat Read Cycle–Basic Timing–One Wait State

13.4.2.2 SINGLE BEAT WRITE FLOW

The basic write cycle begins with a bus arbitration, followed by the address data transfers. The handshakes are illustrated in the following diagrams as they apply to a fixed transaction protocol.

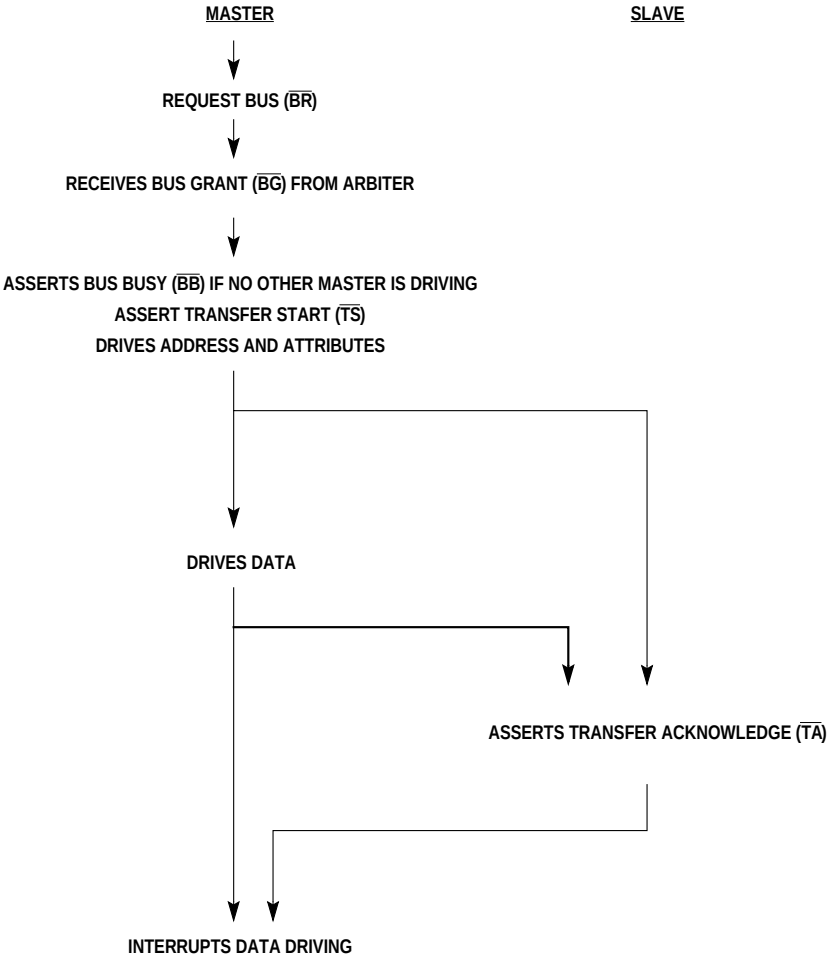


Figure 13-7. Simplified Flow Diagram of a Single Beat Write Cycle

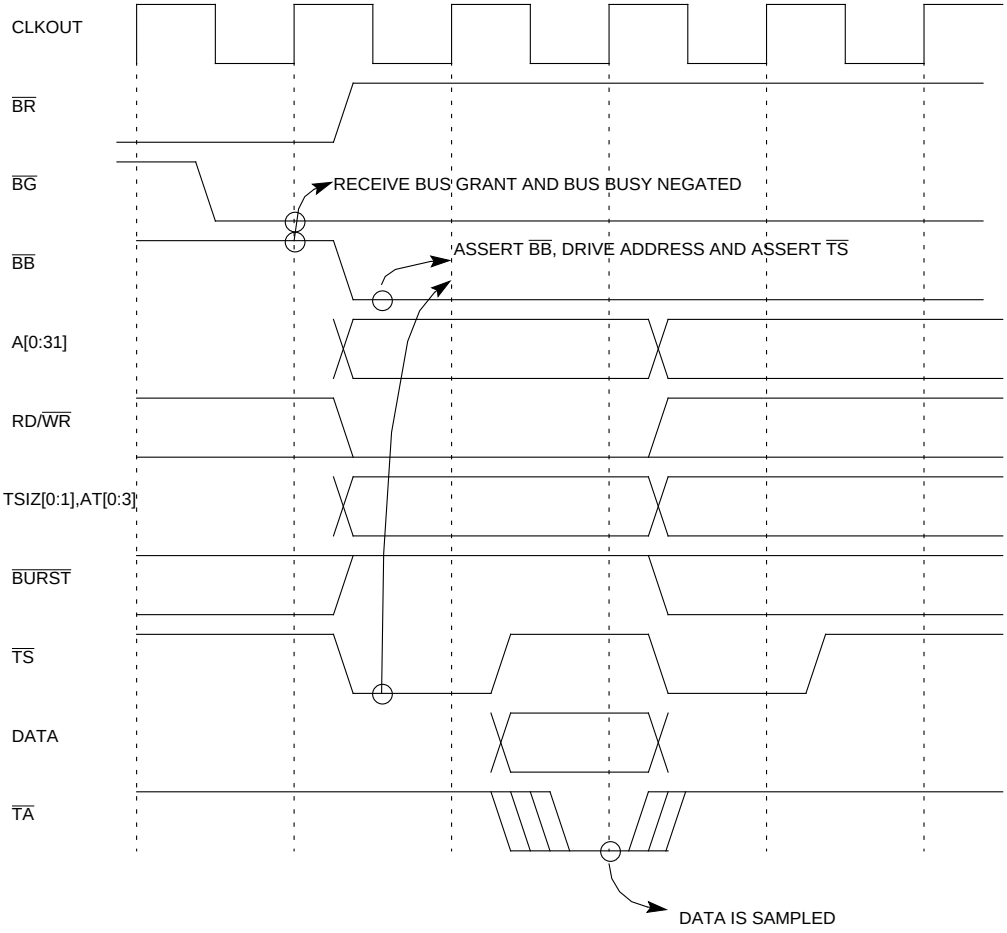


Figure 13-8. Single Beat Write Cycle–Basic Timing–Zero Wait States

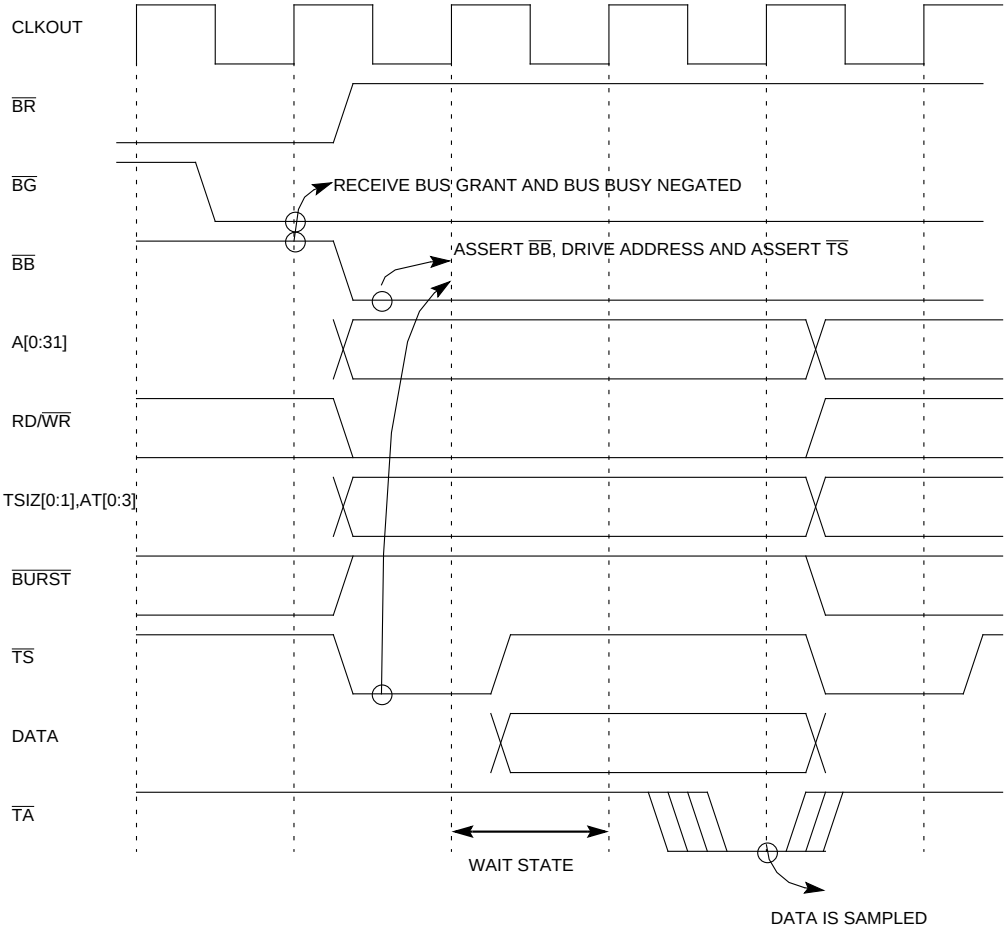


Figure 13-9. Single Beat Write Cycle–Basic Timing–One Wait State

A typical single beat transfer assumes that the external memory has a 32-bit port size. The MPC801 provides an effective mechanism for interfacing with 16- and 8-bit port size memories, which allows transfers to these devices when they are controlled by the internal memory controller.

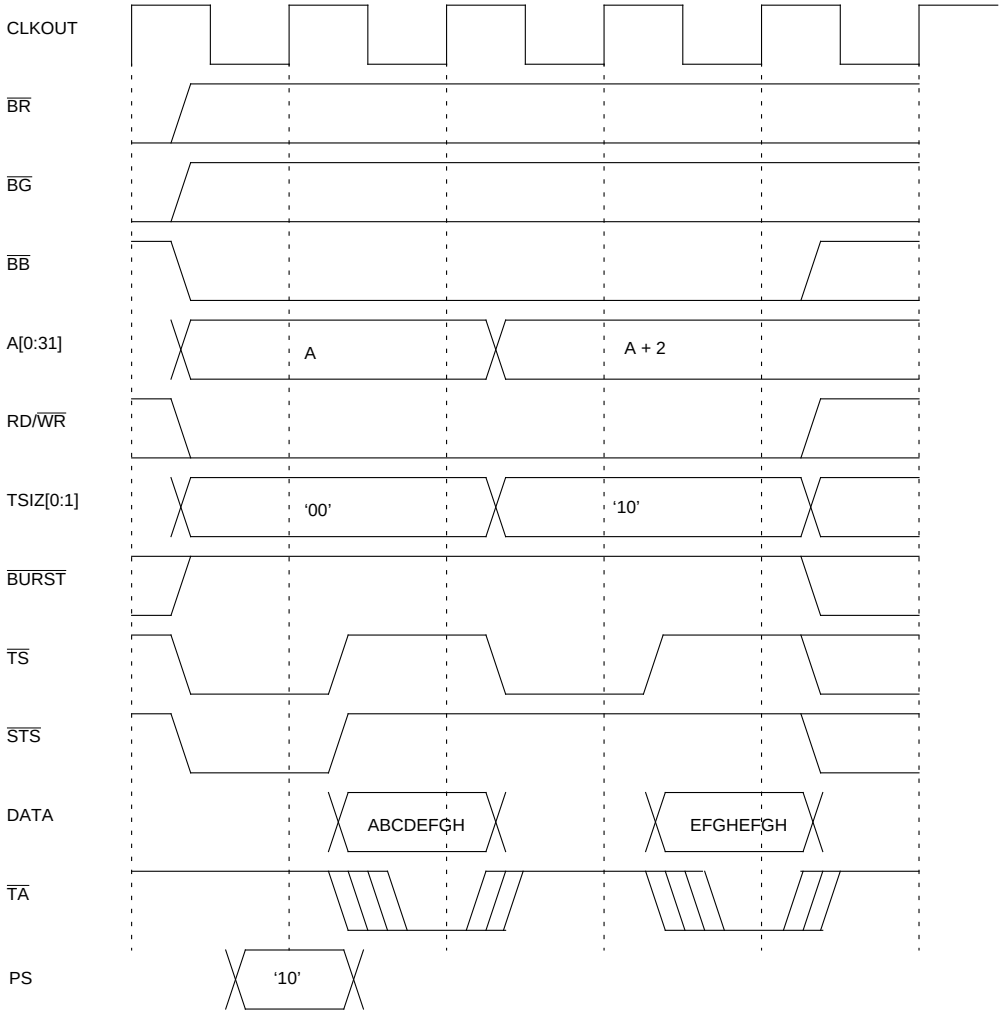


Figure 13-10. Single Beat–32-Bit Data–Write Cycle–16-Bit Port Size Basic Timing

13.4.3 Burst Transfers

The MPC801 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 requires the memory device to sequentially drive or sample each word on the data bus. The selected slave device must internally increment the A[28] and A[29] (A[30] for a 16-bit port size slave device) bits of the supplied address for each transfer, thus causing the address to wrap around at the end of the four words block.

The address and transfer attributes supplied by the MPC801 remain stable during the transfers and the selected device terminates each transfer by driving or sampling the word on the data bus and asserting the $\overline{TA}$ signal. The MPC801 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 or samples the first word the MPC801 points to and asserts the $\overline{BI}$ signal with $\overline{TA}$ for the first transfer of the burst access. The MPC801 responds by terminating the burst and accessing the remainder of the 16-byte block, thus using three read/write cycle buses (each one for a word) for a 32-bit port width slave, seven read/write cycle buses for a 16-bit port width slave, or fifteen read/write cycle buses for a 8-bit port width slave.

Typical burst transfers assume that the external memory has a 32-bit port size. The MPC801 provides an effective mechanism for interfacing with 16- and 8-bit port size memories that allow burst transfers to these devices when they are controlled by the internal memory controller. In this case, the MPC801 tries to initiate a burst transfer as normal. If the slave device responds a cycle before the transfer acknowledge to the first beat, its port size is 16-/8-bits and that the burst is accepted, the MPC801 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-beat burst is considered an atomic transaction, so the MPC801 will not allow other unrelated master accesses or bus arbitration to intervene between the transfers.

13.4.4 The Burst Mechanism

The MPC801 burst mechanism consists of a signal indicating that the cycle is a burst cycle, another indicating the duration of the burst data, and a signal indicating whether the slave is burstable. These signals are in addition to the basic signals of the bus. At the start of the burst transfer, the master drives the address, its attributes, and the $\overline{BURST}$ 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 $\overline{BI}$ signal. If the slave cannot burst, it asserts the burst inhibit signal. During the data phase of a burst write cycle the master drives the data. It also asserts the $\overline{BDIP}$ signal if it intends to drive the data beat after the current data beat. When the slave has received the data, it asserts the signal transfer acknowledge to let the master know it is ready for the next data transfer. The master again drives the next data and asserts or negates the $\overline{BDIP}$ signal. If the master does not intend to drive another data beat after the current one, it negates the $\overline{BDIP}$ to let the slave know the next subsequent data beat transfer is the last data of the burst write transfer. During the data phase of a burst read cycle, the master receives data from the addressed slave. If the master needs more than one data, it asserts the $\overline{BDIP}$ signal. When the data is received before the last data, the master deasserts the $\overline{BDIP}$ signal and the slave stops driving new data after it receives the negation of the $\overline{BDIP}$ signal at the rising edge of the clock.

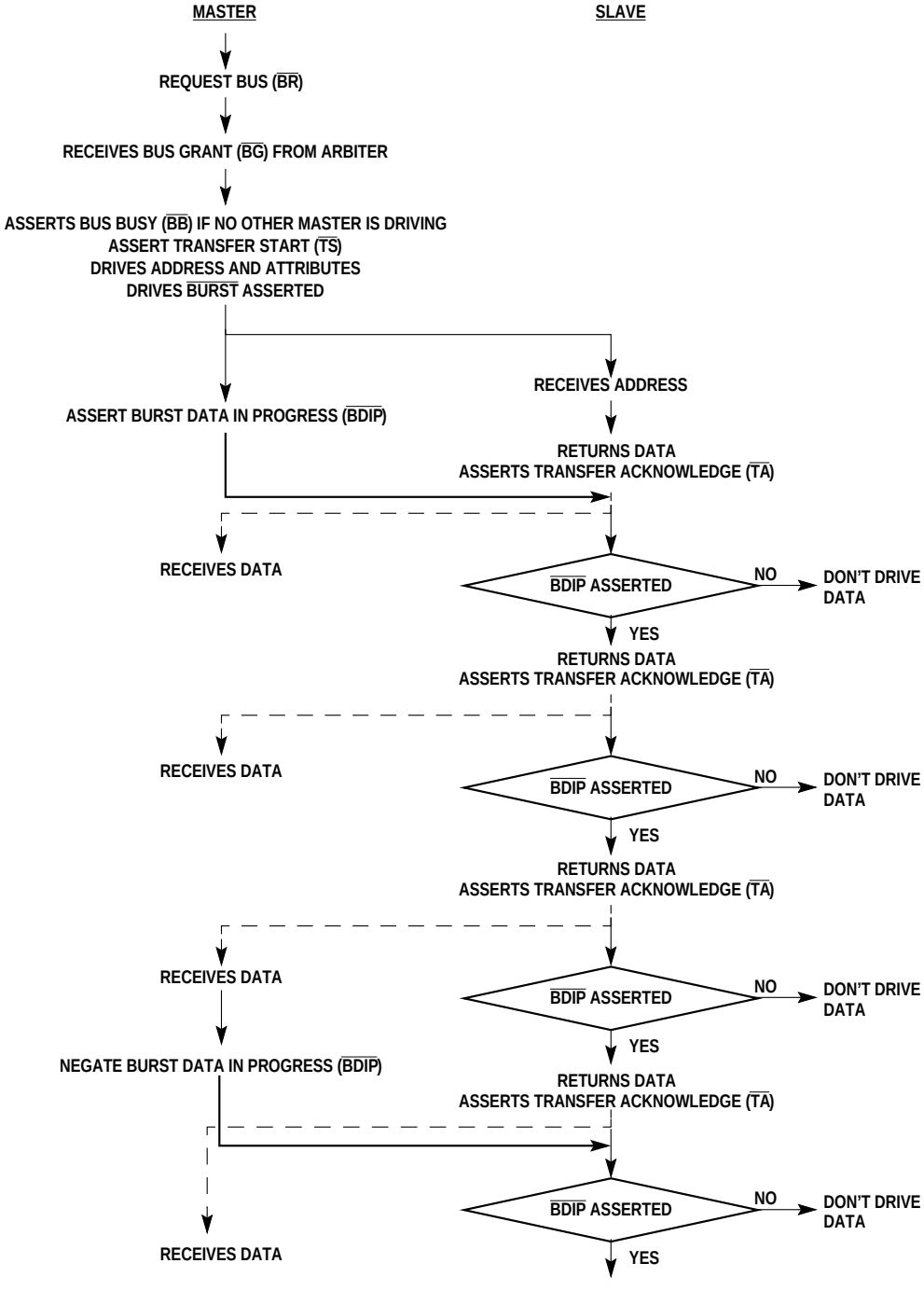


Figure 13-11. Simplified Flow Diagram Of A Burst Read Cycle

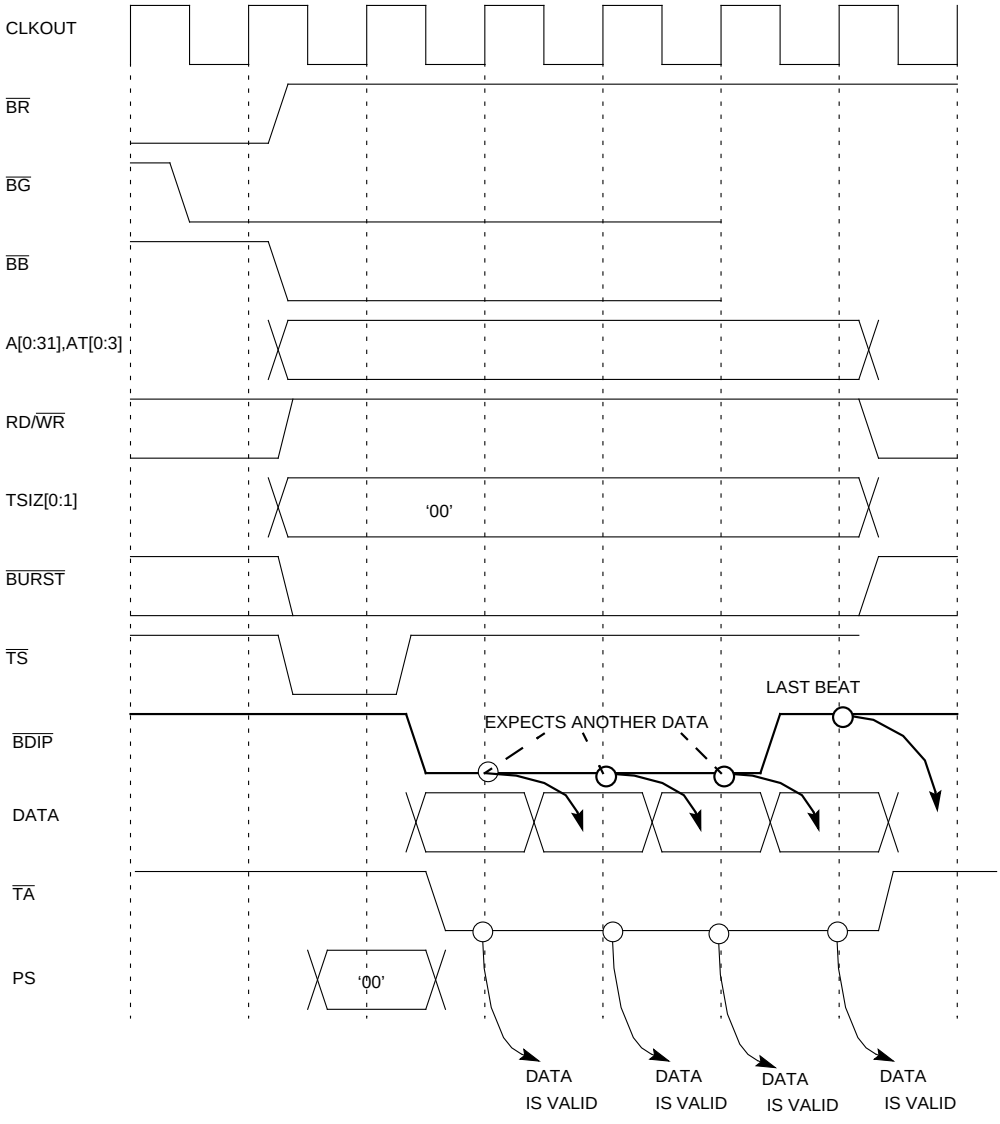


Figure 13-12. Burst-Read Cycle–32-Bit Port Size–Zero Wait State

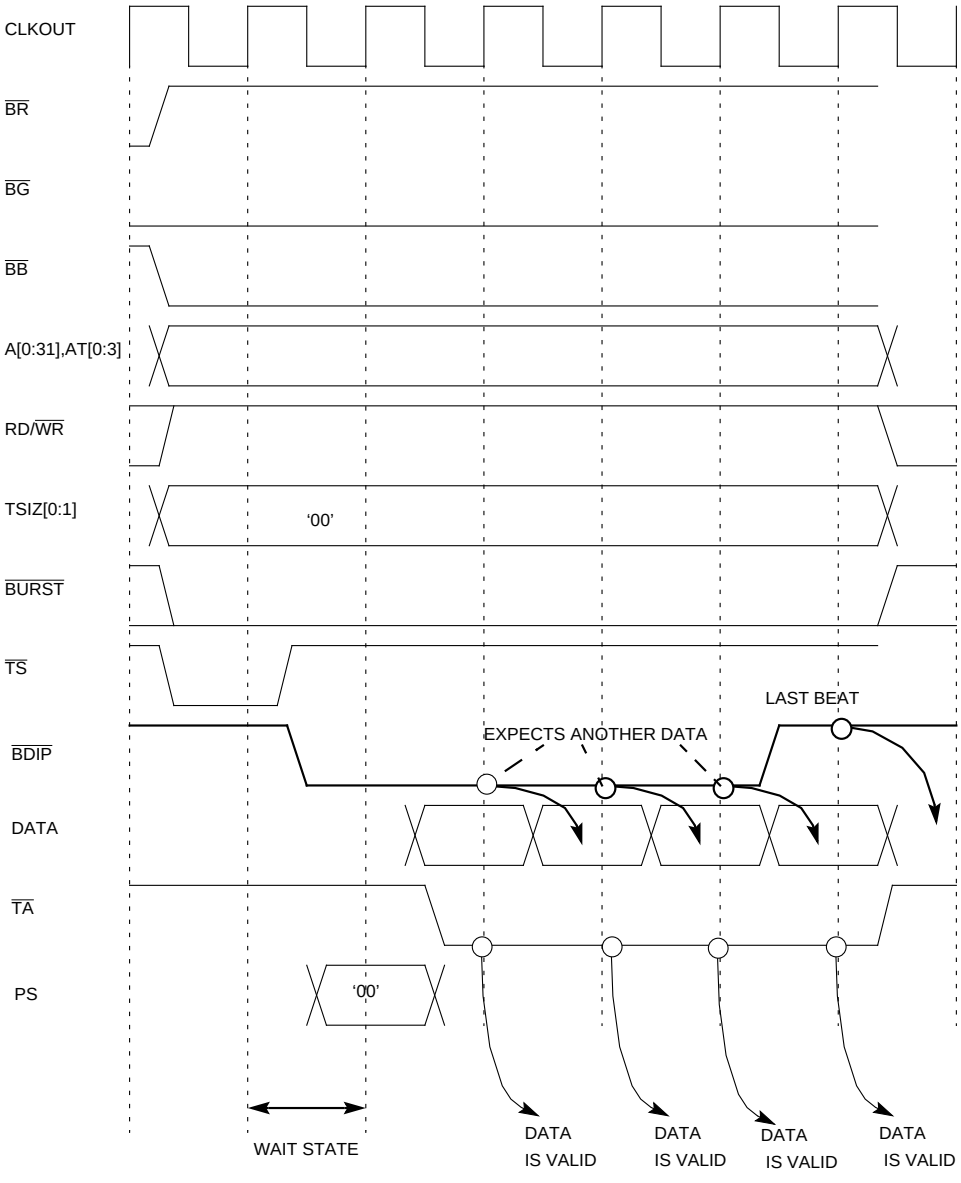


Figure 13-13. Burst-Read Cycle–32-Bit Port Size–One Wait State

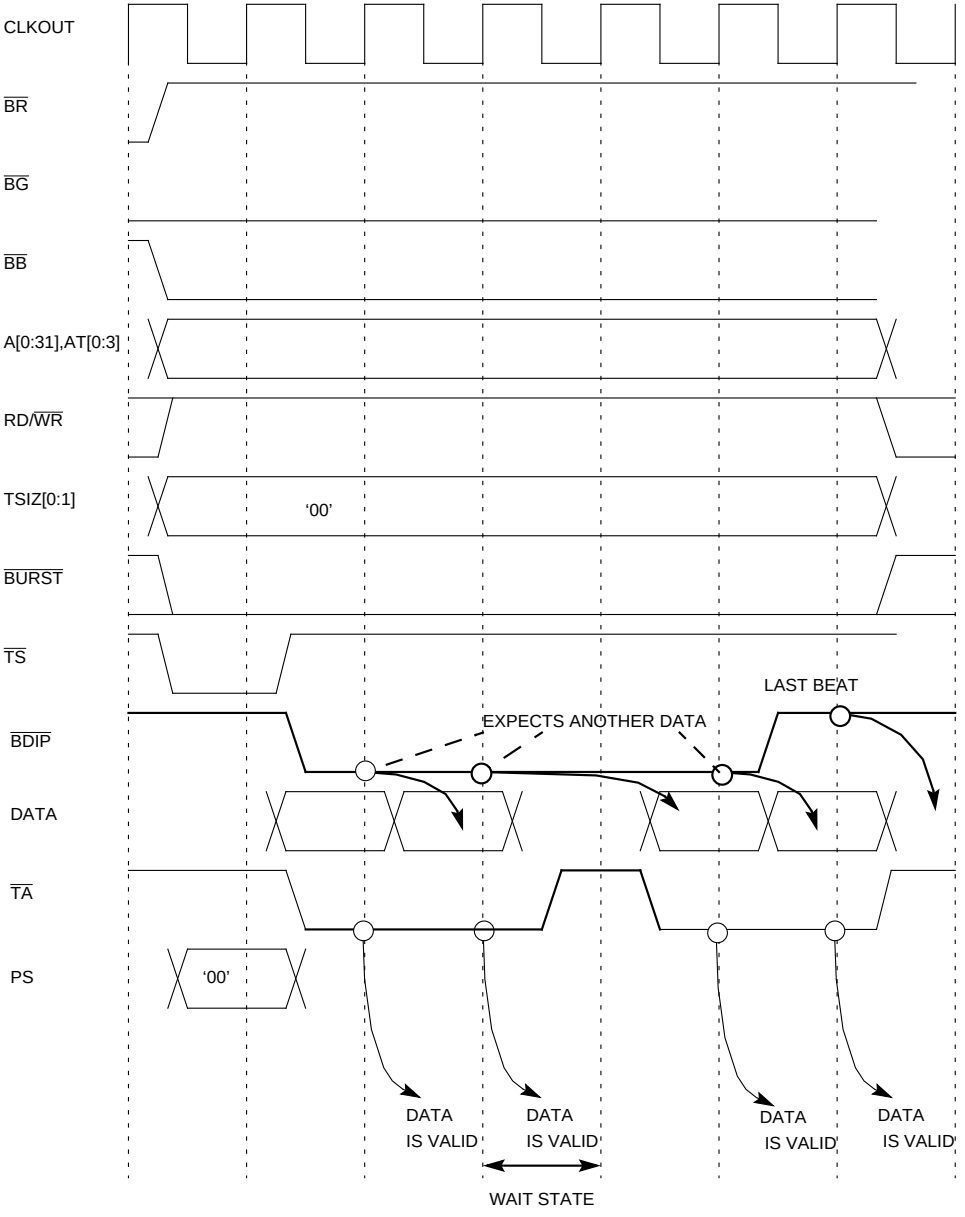


Figure 13-14. Burst-Read Cycle–32-Bit Port Size–Wait States Between Beats

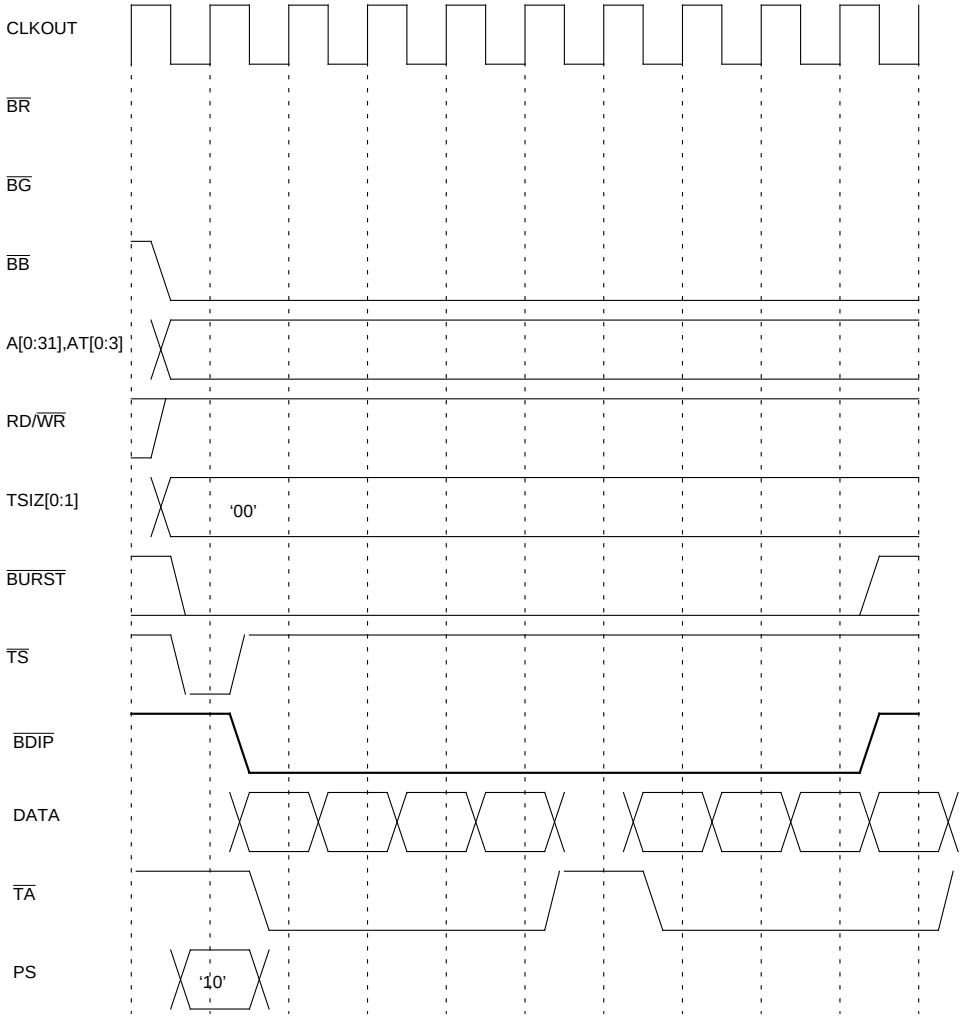


Figure 13-15. Burst-Read Cycle–16-Bit Port Size–One Wait State Between Beats

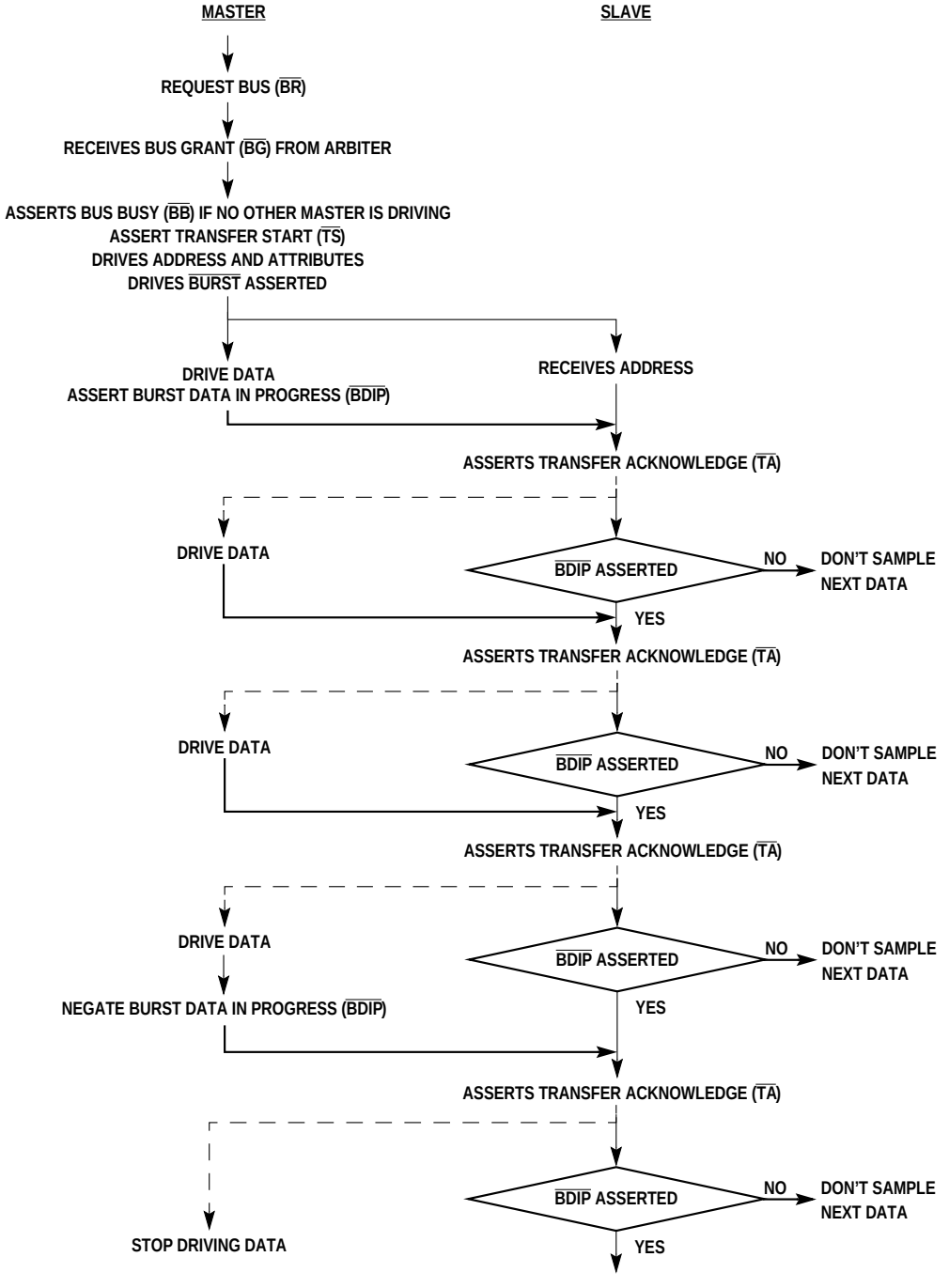


Figure 13-16. Simplified Flow Diagram of a Burst Write Cycle

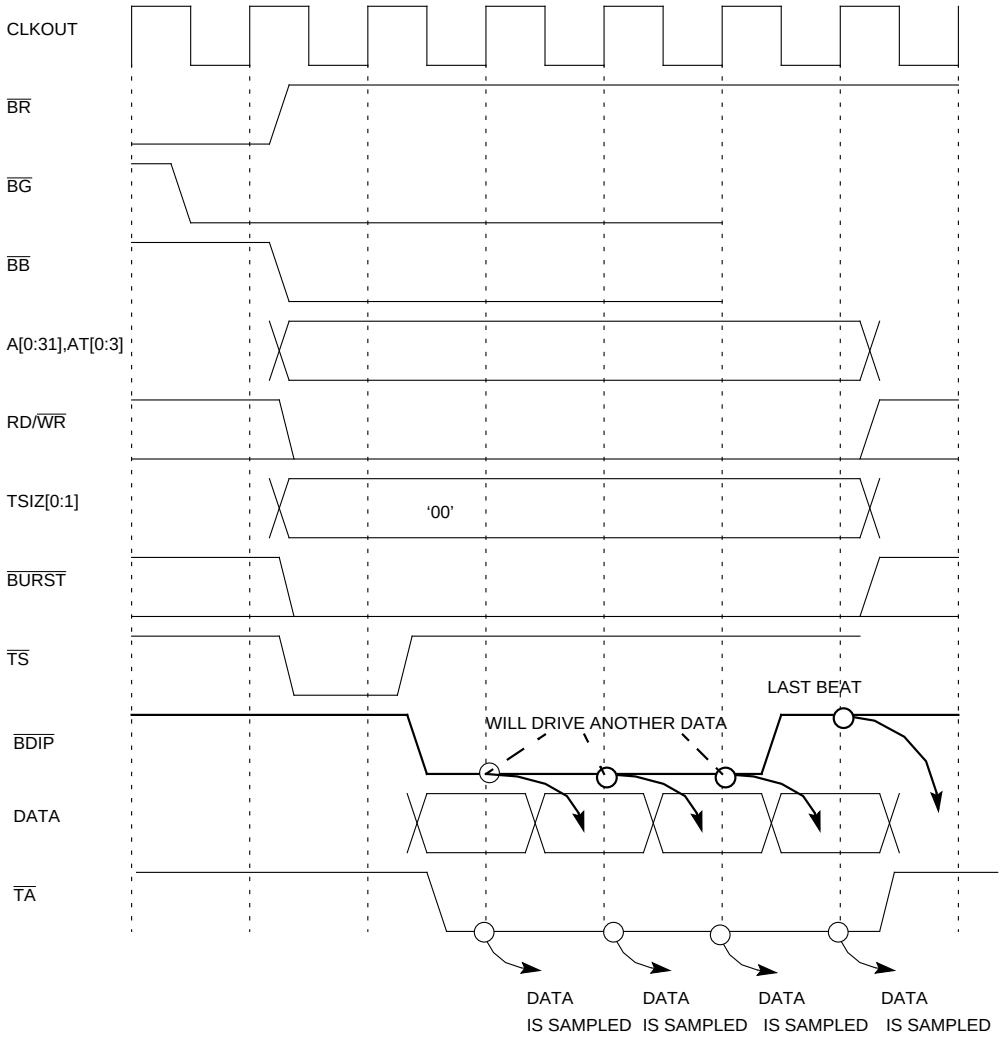


Figure 13-17. Burst Write Cycle–32-Bit Port Size–Zero Wait States

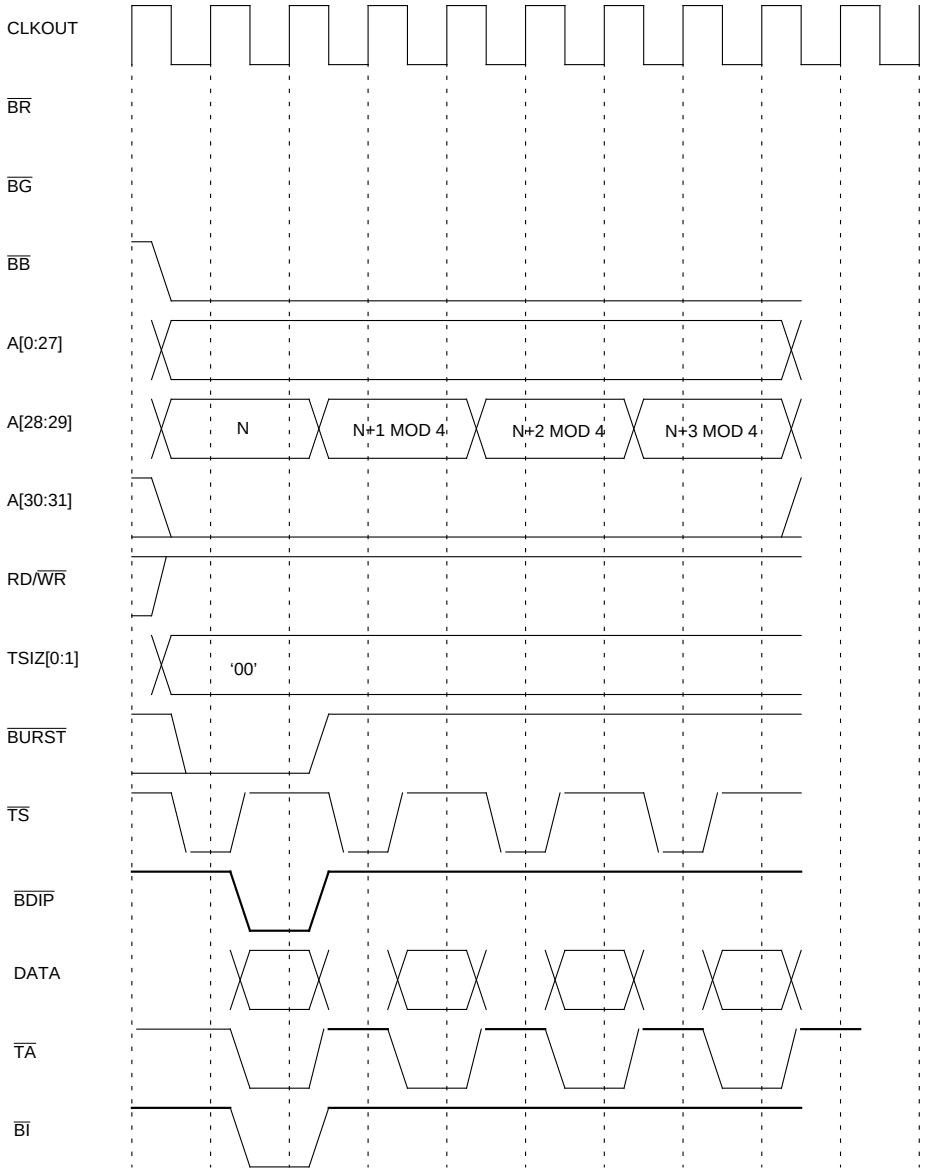


Figure 13-18. Burst-Inhibit Cycle–32-Bit Port Size

13.4.5 Transfer Alignment and Packaging

The MPC801 external bus only supports natural address alignment:

- Byte access can have any address alignment.
- Half-word access must have address bit 31 equal to 0.
- Word access must have address bits 30–31 equal to 0.
- For burst access must have address bits 30–31 equal to 0.

The MPC801 can perform operand transfers through its 32-bit data port. If the transfer is controlled by the internal memory controller, the MPC801 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 MPC801 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 Figures 13-19 and 13-20 and Tables 13-2 and 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.

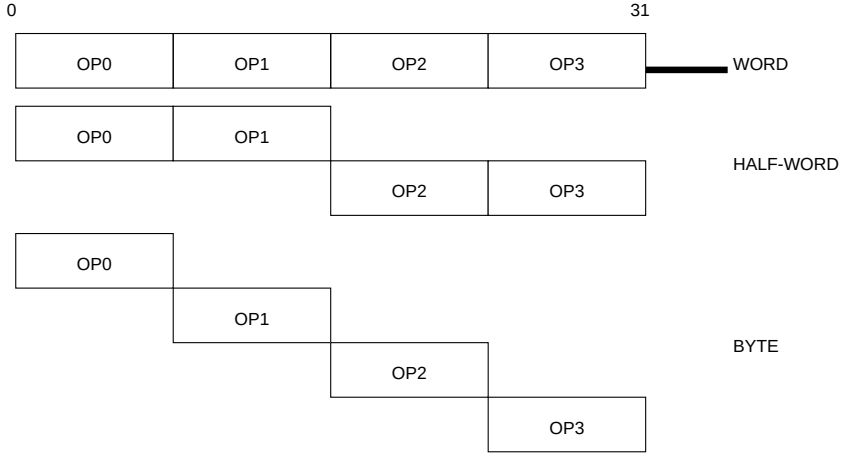


Figure 13-19. Internal Operand Representation

Figure 13-20 illustrates the device connections on the data bus.

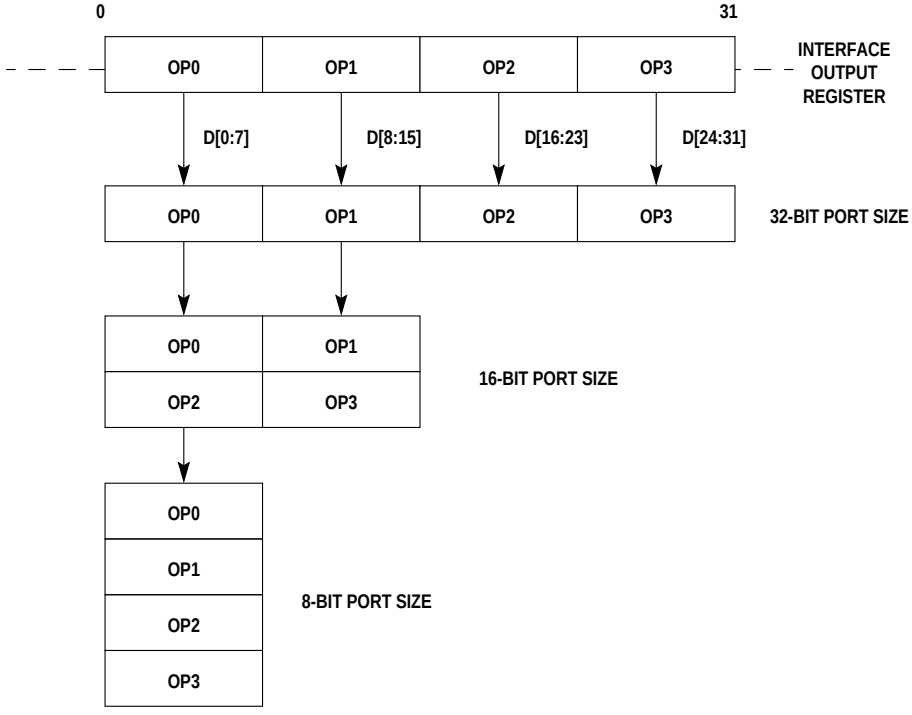


Figure 13-20. Interface To Different Port Size Devices

Table 13-2 lists the bytes required for read cycles on the data bus.

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
			A[30]	A[31]	D[0:7]	D[8:15]	D[16:23]	D[24:31]	D[0:7]	D[8:15]	D[0:7]
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 MPC801.

Table 13-3. Data Bus Contents for Write Cycles

TRANSFER SIZE	TSIZE		ADDRESS		EXTERNAL DATA BUS PATTERN			
			A[30]	A[31]	D[0:7]	D[8:15]	D[16:23]	D[24:31]
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.

13.4.6 Arbitration Phase Signals

The external bus design provides for a single bus master, either the MPC801 or an external device. One or more of the external devices on the bus has the capability of becoming bus master for the external bus. Bus arbitration can 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 MPC801. The external or internal arbitration configuration is set at system reset. See **Section 15.3 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 $\overline{BR}$ signal. The device then waits for the arbiter to assert a $\overline{BG}$ signal. In addition, the new master must look at the $\overline{BB}$ signal to ensure that no other master is driving the bus before it can assert bus busy and assume ownership of the bus. If 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 made. The MPC801, however, guarantees data coherency for access to a small port size and decomposed bursts. This means that the MPC801 will not release the bus before the transactions that are considered atomic complete. Figure 13-21 describes the basic protocol for bus arbitration. See **Section 12.12.1.1 SIU Module Configuration Register** for details.

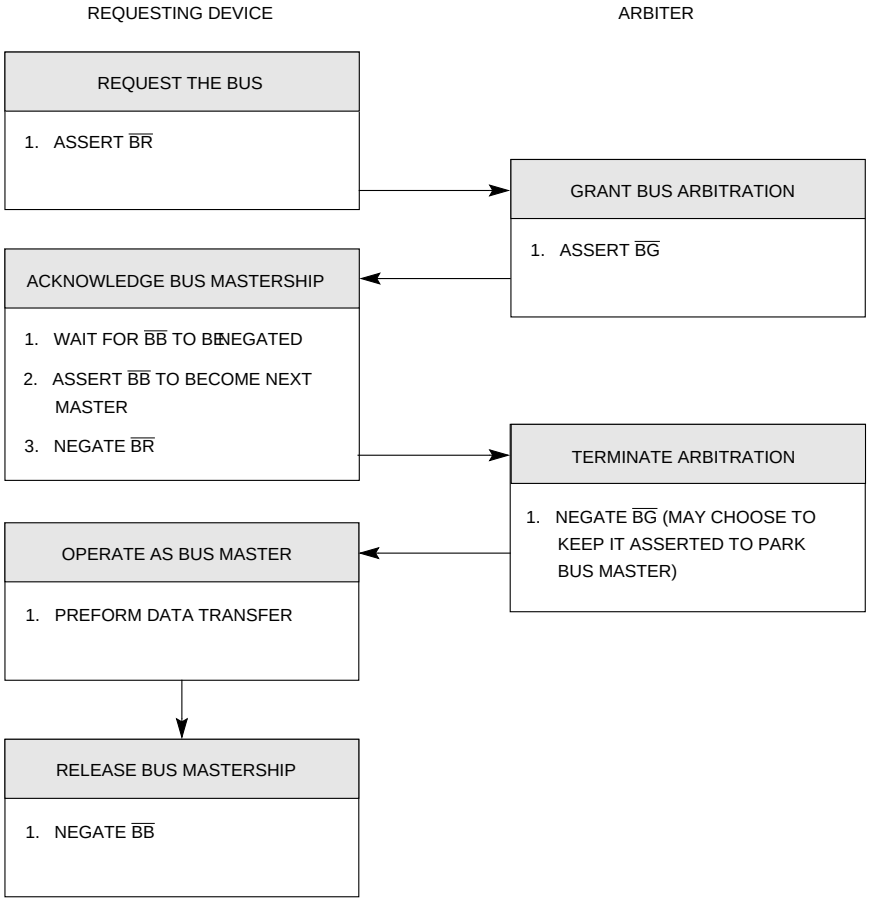


Figure 13-21. Bus Arbitration Flowchart

13.4.6.1 BUS REQUEST

The potential bus master asserts the $\overline{BR}$ signal to request bus mastership. $\overline{BR}$ should be negated once the bus is granted, the bus is not busy, and the new master can drive the bus. If more requests are pending, the master can keep asserting its bus request as long as needed. When configured for external central arbitration, the MPC801 drives this signal when it requests bus mastership. When the internal on-chip arbiter is used, this signal is an input to the internal arbiter and should be driven by the external bus master.

13.4.6.2 BUS GRANT

The $\overline{BG}$ signal is asserted by the arbiter to indicate that the bus is granted to the requesting device. This signal can be negated after the $\overline{BR}$ signal is negated or kept asserted for the current master to park the bus. When configured for external central arbitration, $\overline{BG}$ is an input signal to the MPC801 from the external arbiter. When the internal on-chip arbiter is used, this signal is an output from the internal arbiter to the external bus master.

13.4.6.3 BUS BUSY

The $\overline{BB}$ signal indicates that the current bus master is using the bus. New masters should not begin transferring until this signal is deasserted. The bus owner should not relinquish or negate this signal until its transfer is complete. To avoid contention on the $\overline{BB}$ signal, masters should three-state this signal when it gets a logical ‘1’ value. This situation implies that the connection of an external pull-up resistor is needed to ensure that a master acquiring the bus can recognize the negated $\overline{BB}$ signal, regardless of how many cycles have passed since the previous master relinquished the bus. Refer to Figure 13-22 for more information.

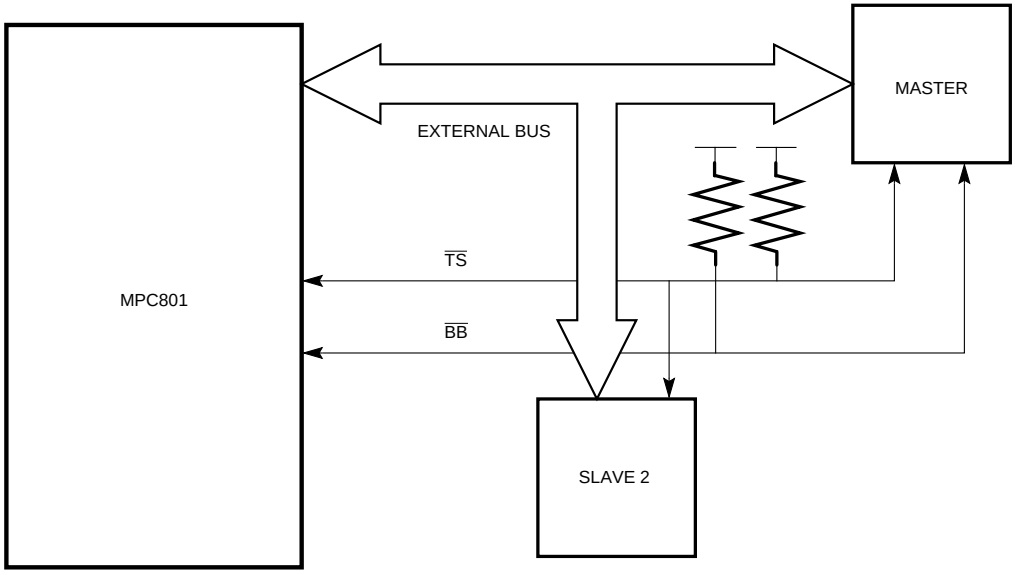


Figure 13-22. Basic Connection of the Master Signal

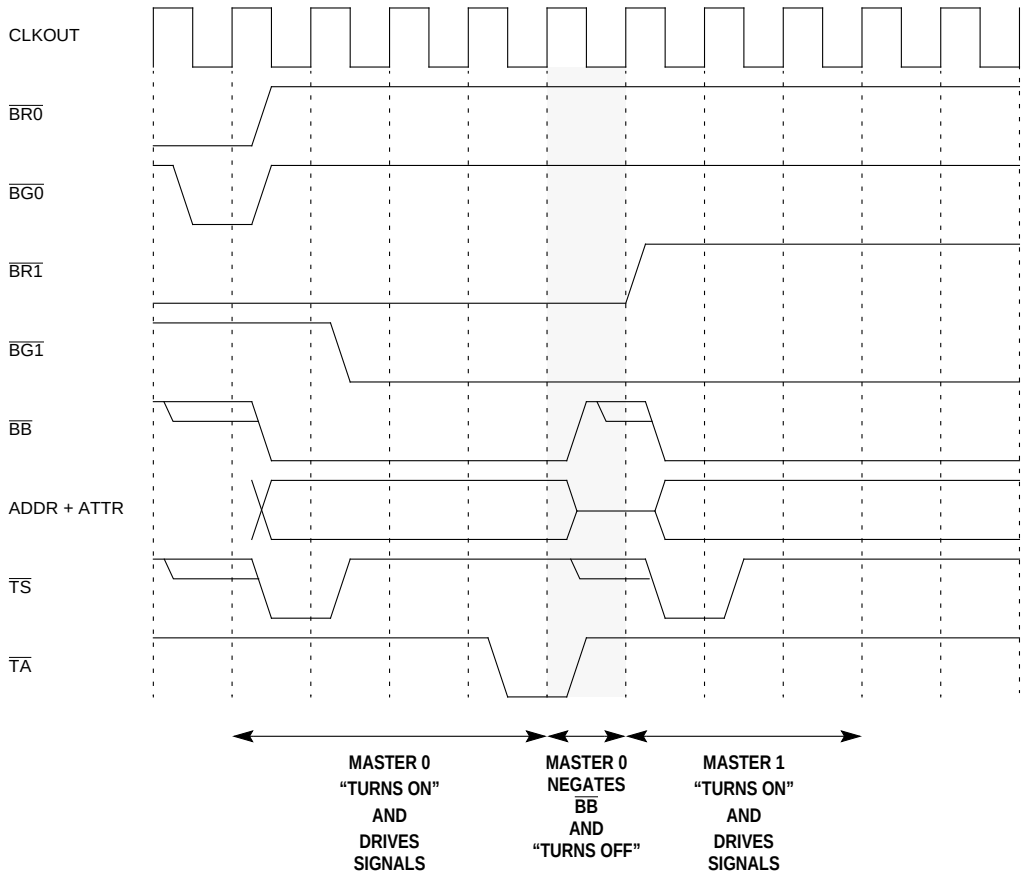


Figure 13-23. Bus Arbitration Timing Diagram

At system reset, the MPC801 can be configured to use the internal bus arbiter and it will be parked on the bus. The priority of the external device relative to the internal MPC801 bus masters is programmed in the SIU module configuration register. If the external device requests the bus and the MPC801 does not need it or the external device has priority over the current internal bus master, the MPC801 grants the bus to the external device. Figure 13-24 illustrates the internal finite state machine that implements the arbiter protocol.

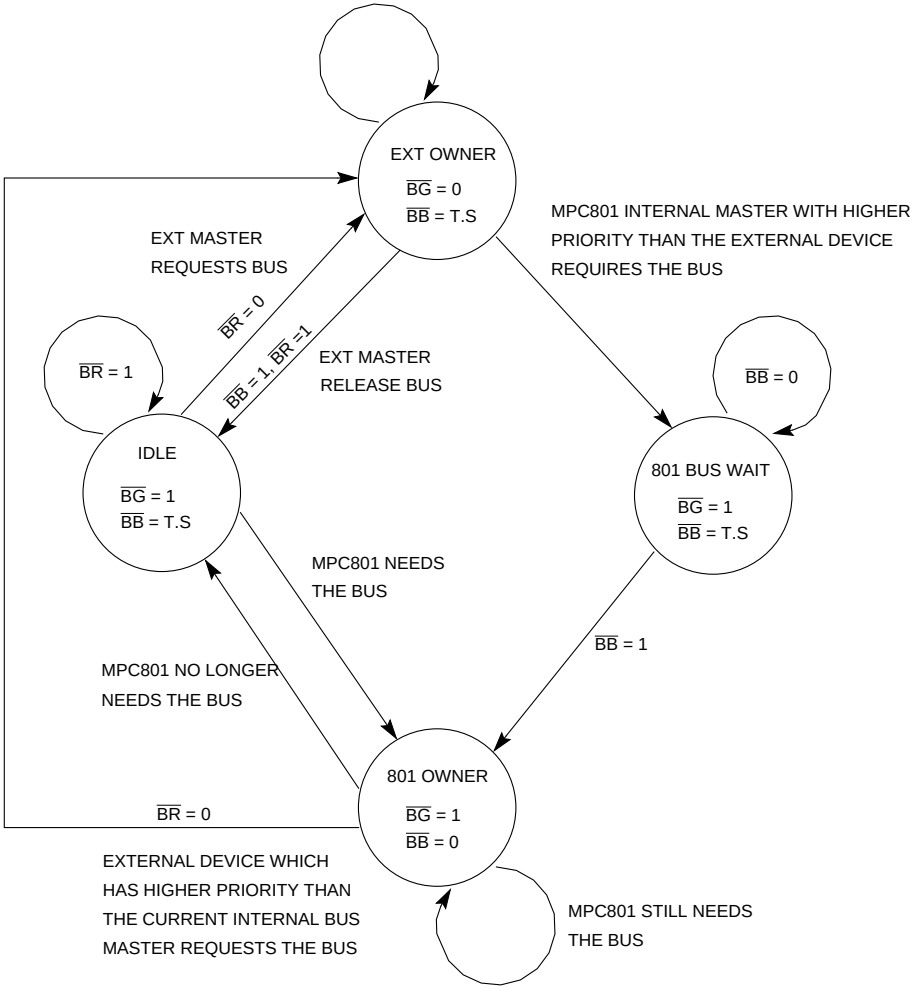


Figure 13-24. Internal Bus Arbitration State Machine

13

13.4.7 Address Transfer Phase-Related Signals

13.4.7.1 TRANSFER START

The $\overline{TS}$ signal indicates the beginning of a transaction on a bus that is addressing a slave device. This signal should be asserted by a master only after ownership of the bus is granted by the arbitration protocol. It 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 signal. This situation indicates that an external pull-up resistor should be connected to the $\overline{TS}$ signal to keep a slave from recognizing this asserted signal when no master drives it. Refer to Figure 13-22 for more information.

13.4.7.2 ADDRESS BUS

The 32-bit address bus consists of address bits 0–31 and 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 stay valid until the bus master received a signal transfer acknowledge from the slave. To distinguish the individual byte, the slave device must observe the TSIZ signals.

13.4.7.3 TRANSFER ATTRIBUTES

The transfer attributes include the RD/$\overline{WR}$, $\overline{BURST}$, TSIZ[0:1], AT[0:3], $\overline{STS}$, and $\overline{BDIP}$ signals. These signals, except $\overline{BDIP}$, are available at the same time as the address bus.

13.4.7.3.1 Read/Write. When the RD/$\overline{WR}$ signal is high, it indicates a read access and low indicates a write access.

13.4.7.3.2 Burst Indicator. When the $\overline{BURST}$ signal is driven by the bus master at the beginning of the bus cycle to indicate that the transfer is a burst transfer. The burst size is always 16 bytes long. For a 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 MPC801 bus supports critical data first access for fixed-size burst. The order of the wraparound wraps back to 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

13.4.7.3.3 Transfer Size. The TSIZ signals indicate the size of the requested data transfer. The TSIZ signals can be used with $\overline{BURST}$ 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 the TSIZ signal is always 00.

Table 13-4. $\overline{\text{BURST}}$/TSIZE Encoding

$\overline{\text{BURST}}$	TSIZ	TRANSFER SIZE
N	01	Byte
N	10	Half-Word
N	11	x
N	00	Word
A	00	Burst (16 bytes)

13.4.7.3.4 Address Types. The A[0:3], $\overline{\text{PTR}}$, and $\overline{\text{RSV}}$ signals 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, and instruction/data types. The address type signals are valid at the rising edge of the clock in which the $\overline{\text{STS}}$ signal is asserted.

Address type signals reflect the current status of the master originating the access, not necessarily the status in which the original access to this location has occurred. An example of this situation is when a copyback of a dirty line in the data cache occurs after the privilege state of the processor has been changed since the last access to the same line. Functional usage of the AT[0:3], $\overline{\text{PTR}}$ and $\overline{\text{RSV}}$ signals is for the reservation protocol described in **Section 13.4.9 Storage Reservation Protocol**. Table 13-5 provides the space definition encoded by the $\overline{\text{STS}}$, $\overline{\text{TS}}$, AT[0:3], $\overline{\text{PTR}}$, and $\overline{\text{RSV}}$ signals.

Table 13-5. Definitions of Address Types

STS	TS	AT0	AT1	AT2	AT3	PTR	RSV	ADDRESS SPACE DEFINITIONS
			PROBLEM STATE/ PRIVILEGE STATE	INSTRUCTION/ DATA	RESERVATION/ PROGRAM TRACE	PROGRAM TRACE	RESERVATION	
1	x	x	x	x	x	1	1	No transfer or no first transaction of a transfer
0	x	x	x	x	x	x	x	Start of a transaction
x	0	0	0	0	0	0	1	Core, normal instruction, program trace, privilege state
					1	1	1	Core, normal instruction, privilege state
				1	0	1	0	Core, reservation data, privilege state
					1	1	1	Core, normal data, privilege state
			1	0	0	0	1	Core, normal instruction, program trace, problem trace
					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)
		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 core's internal bus devices. These accesses are driven externally for emulation, visibility, and debugging purposes. A show cycle can have one address phase and one data phase or just an address phase for the instruction show cycles. The cycle can be a write or read access and the data for both the read and write accesses should be driven by the bus master. This is different than the normal bus read and write accesses. The address of the show cycle should be valid on the bus for one clock and the data of the show cycle should be valid on the bus for one clock. The data phase should not require a transfer acknowledge to terminate the bus-show cycle. In a burst-show cycle only the first data beat will be shown externally.

13.4.7.3.5 Burst Data in Progress. The $\overline{\text{BDIP}}$ signal is sent from the master to the slave to indicate that there is a data beat following the current data beat. The master uses this signal to give the slave an advanced warning of the remaining data in the burst. This signal can also be used to terminate the burst cycle early during a burst. Refer to **Section 13.4.2 Single Beat Transfers** and **Section 13.4.4 The Burst Mechanism** for more information.

13.4.8 Termination Signals

13.4.8.1 TRANSFER ACKNOWLEDGE

The $\overline{\text{TA}}$ signal indicates that a bus transfer has completed normally. During a burst cycle, the slave asserts this signal with every data beat returned or accepted.

13.4.8.2 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 permanently tied to a low.

13.4.8.3 TRANSFER ERROR ACKNOWLEDGE

The $\overline{\text{TEA}}$ signal 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.

13.4.8.4 PROTOCOL FOR TERMINATION SIGNALS

The transfer protocol was defined to avoid electrical contention on signals that can be driven by various sources. To do that, a slave should not drive signals associated with the data transfer until the address phase is completed and it recognizes the address as its own. The slave should disconnect from signals immediately after it has acknowledged the cycle and no later than the termination of the next address phase cycle. This indicates that the termination signals should be connected to the power through a pull-up resistor to avoid a situation in which a master samples an undefined value in any of these signals when no real slave is addressed. Refer to Figures 13-25 and 13-26 for more information. Table 13-6 summarizes how the MPC801 recognizes the termination signals provided by the slave device that the initiated transfer addressed.

Table 13-6. Termination Signal Protocol

$\overline{\text{TEA}}$	$\overline{\text{TA}}$	$\overline{\text{RETRY}}$/$\overline{\text{KR}}$	RESULT
Asserted	X	X	Transfer Error Termination
Negated	Asserted	X	Normal Transfer Termination
Negated	Negated	Asserted	Retry Transfer Termination / Kill Reservation

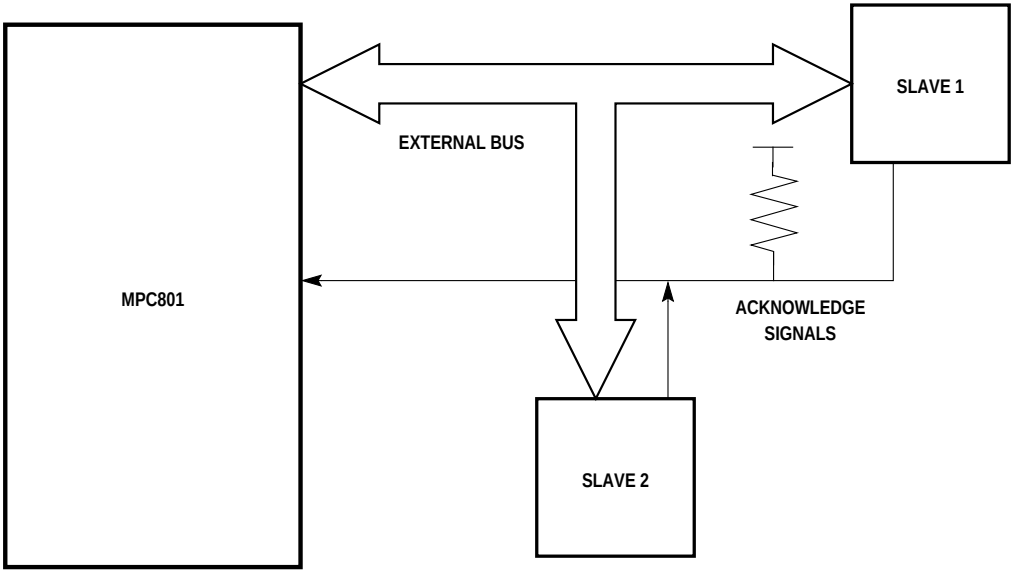


Figure 13-25. Termination Signals Protocol Basic Connection

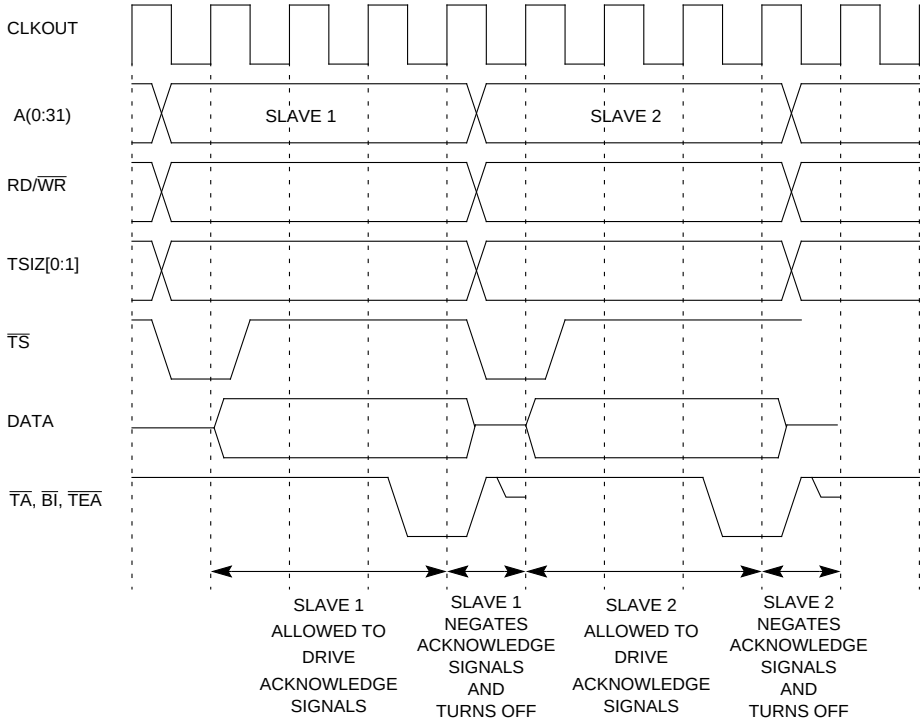


Figure 13-26. Termination Signals Protocol Timing Diagram

13.4.9 Storage Reservation Protocol

The MPC801 storage reservation protocol supports multilevel bus structure. For each local bus, storage reservation is handled by the local reservation logic. The protocol tries to optimize reservation cancellation so that a PowerPC processor is notified of storage reservation loss on a remote bus only when it has issued a **stwcx** cycle to that address. In other words, the reservation loss indication comes as part of the **stwcx** cycle. This method avoids the need to have fast storage reservation loss indication signals routed from every remote bus to every PowerPC master. The storage reservation protocol makes the following assumptions:

- Each “processor” has, at most, one reservation “flag”.
- **lwarx** sets the reservation “flag”.
- **lwarx** by the same processor clears the reservation “flag” related to a previous **lwarx** instruction and again sets the reservation “flag”.
- **stwcx** by the same processor clears the reservation “flag”.
- Store by the same processor does not clear the reservation “flag”.

- Some other processor (or other mechanism) store to the same address as an existing reservation clears the reservation “flag”.
- If the storage reservation is lost, it is guaranteed that **stwcx** will not modify the storage.

The reservation protocol for a single-level (local) bus is illustrated in Figure 13-27. 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.

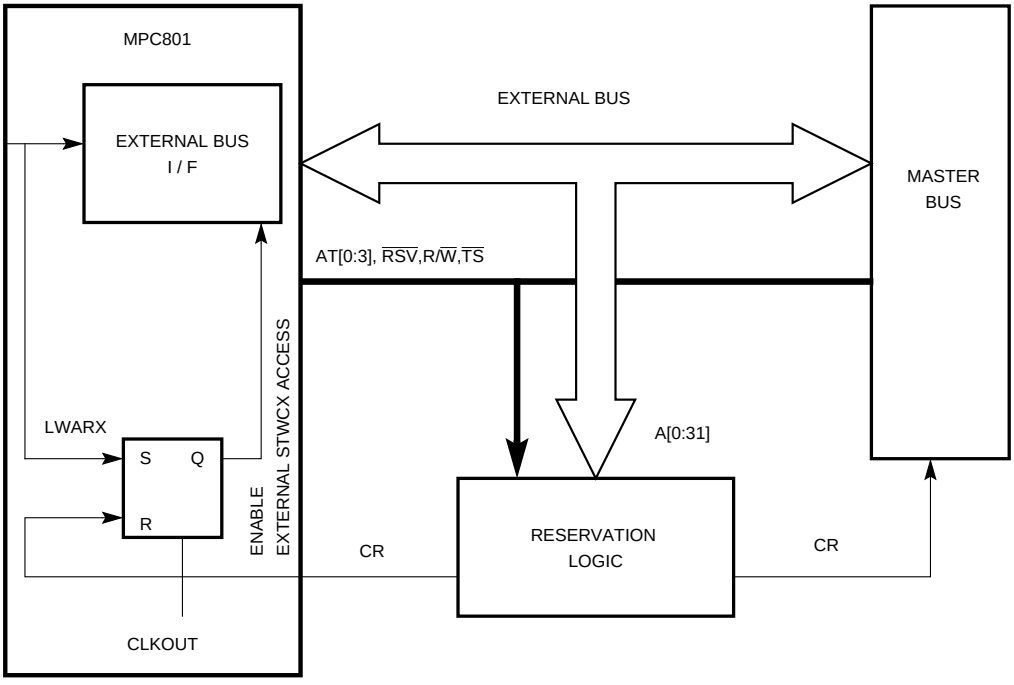


Figure 13-27. Reservation On A Local Bus

The CR signal is sampled by the MPC801 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 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 core.

The reservation protocol for a multi-level (local) bus is illustrated in Figure 13-28. The system describes the situation in which the reserved location is sited in the remote bus.

In this case, the bus's interface block implements a reservation "flag" for the local bus master. The reservation "flag" is set by the bus's interface when a load with reservation is issued by the local bus master and the reservation address is located on the remote bus. The "flag" is reset when an alternative master on the remote bus accesses the same location in a write cycle. If the MPC801 begins a memory cycle to the previously reserved address (located in the remote bus) as a result of a **stwcx** instruction, the following two situations can occur:

- If the reservation "flag" is set, the bus's interface acknowledges the cycle in a normal way and if it is reset, the bus's interface should assert the $\overline{KR}$.
- The bus's interface should either perform the remote bus write access or abort it if the remote bus supports aborted cycles. In this situation, failure of the **stwcx** instruction is reported to the core.

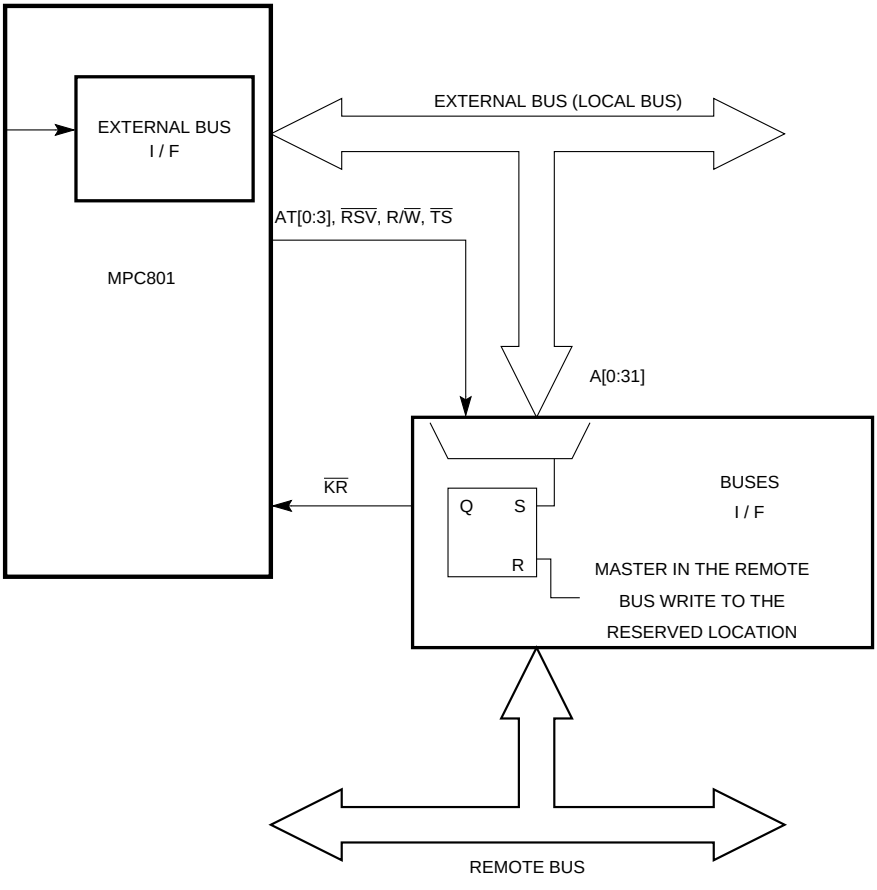


Figure 13-28. Reservation On Multilevel Bus Hierarchy

13.4.10 Exception Control Cycles

The MPC801 bus architecture requires the $\overline{TA}$ signal to be asserted from an external device to indicate bus cycle completion. $\overline{TA}$ is not asserted when one of the following conditions occur:

- The external device does not respond
- Other application-dependent errors occur

The external circuitry can provide $\overline{TEA}$ when no device responds by asserting $\overline{TA}$ within an appropriate period of time after the MPC801 initiates the bus cycle. This allows the cycle to terminate and the processor to enter exception processing for the error condition, whereas each of the internal masters causes an internal interrupt. To properly control the termination of a bus cycle for a bus error, the $\overline{TEA}$ signal must be asserted simultaneously or before $\overline{TA}$ is asserted. $\overline{TEA}$ should be negated before the second rising edge after it was sample-asserted to avoid an error for the next initiated bus cycle. $\overline{TEA}$ is an open-drain pin that allows the “wire-OR” of any different error generation sources.

13.4.10.1 RETRY

When an external device asserts the $\overline{RETRY}$ signal during a bus cycle, the MPC801 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 (for a write cycle). Figure 13-29 illustrates the behavior of the MPC801 when the $\overline{RETRY}$ signal is found as a termination of a transfer. In the figure, it is illustrated that when the internal arbiter is enabled, MPC801 negates the $\overline{BB}$ signal and asserts the $\overline{BG}$ signal in the clock cycle following the retry detection. This allows any external master to gain bus ownership. In the next clock cycle, a normal arbitration procedure occurs again. The figure also illustrates that the external master did not use the bus, so the MPC801 initiates a new transfer with the same address and attributes as before. In Figure 13-30 the same situation is illustrated to show that the MPC801 is working with an external arbiter. If the clock cycle after the $\overline{RETRY}$ signal is asserted, the $\overline{BR}$ signal is negated together with the $\overline{BB}$ signal. One clock cycle later, the normal arbitration procedure occurs again.

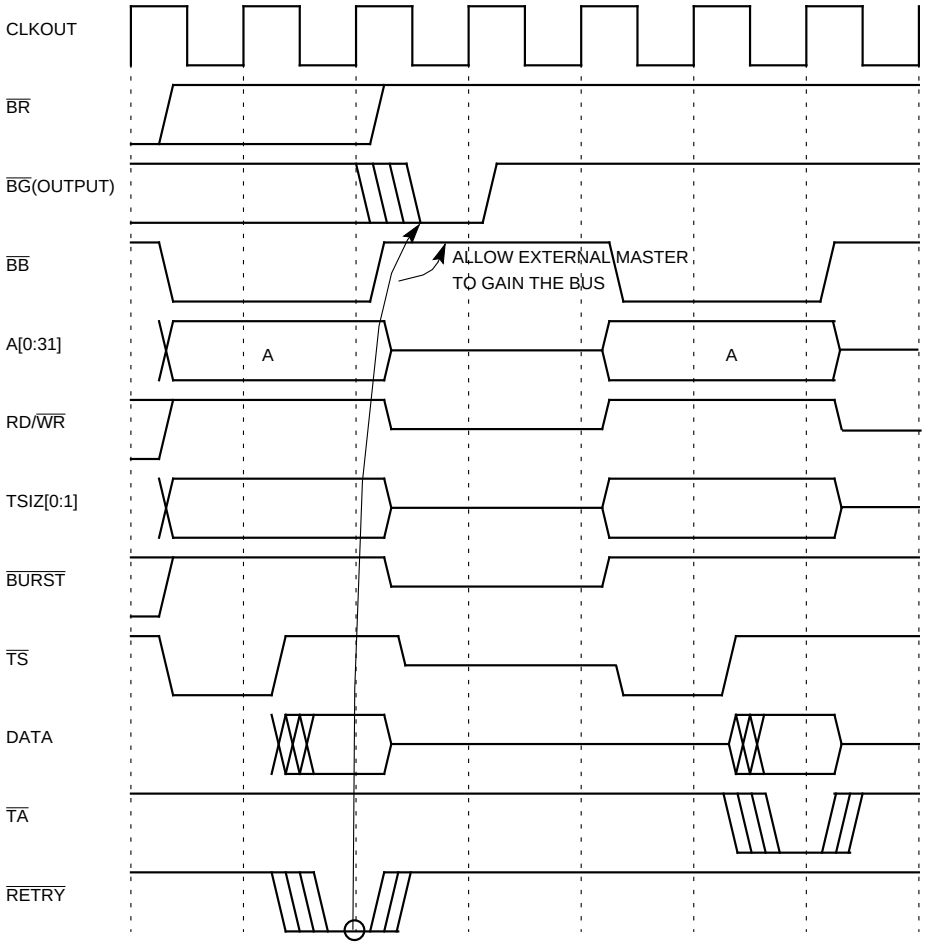


Figure 13-29. RETRY Transfer Timing–Internal Arbiter

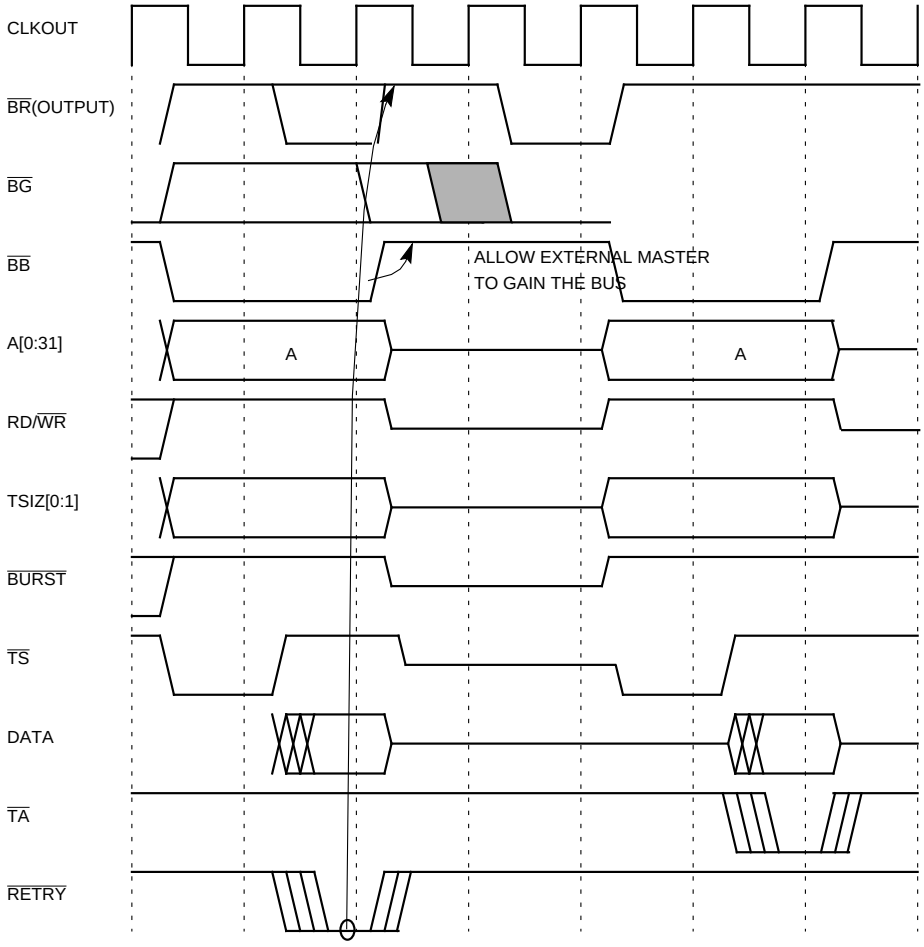


Figure 13-30. RETRY Transfer Timing–External Arbiter

When the MPC801 initiates a burst access, the bus interface only recognizes the RETRY assertion as a retry termination if it detects it before the first data beat is acknowledged by the slave device. When the RETRY signal is asserted as a termination signal on the second or third data beat of the access (being the first data beat acknowledged by a normal TA assertion), the MPC801 recognizes it as a transfer error acknowledge.

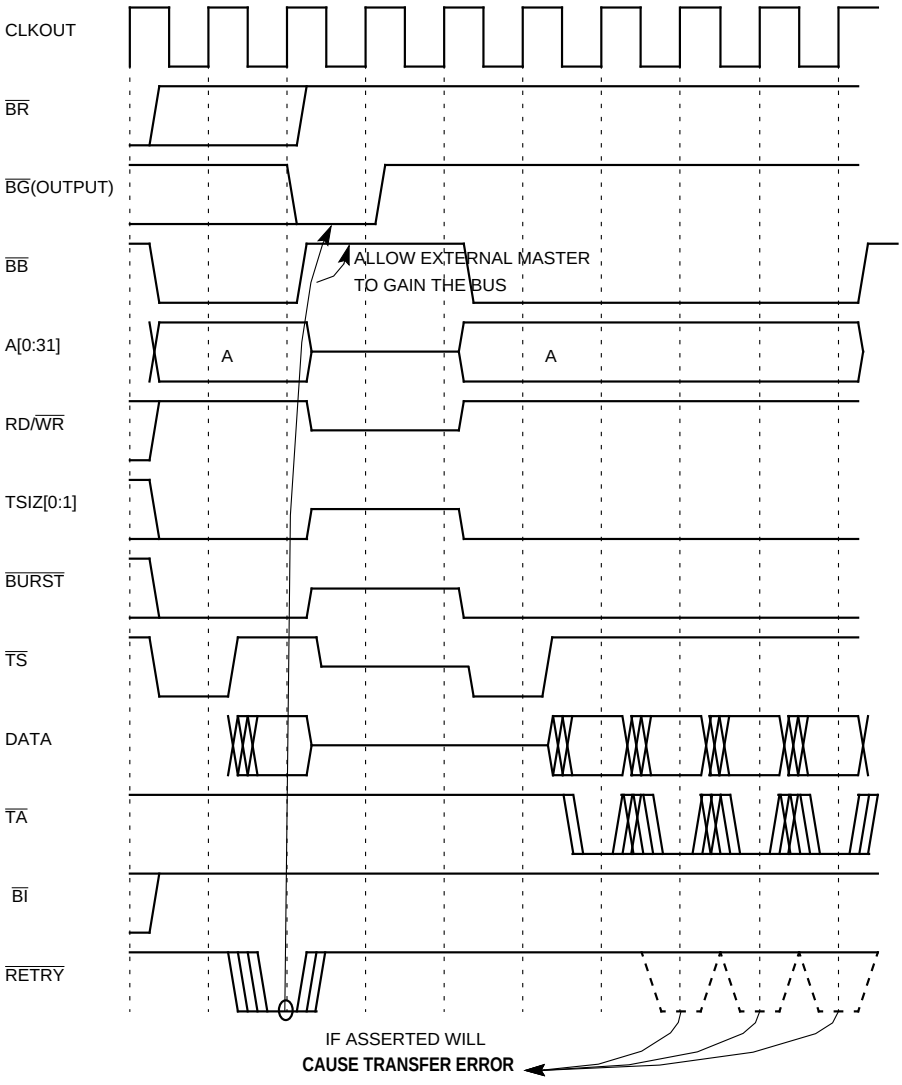


Figure 13-31. RETRY On Burst Cycle

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 MPC801 to complete the 16-byte transfers recognize the RETRY signal assertion as a transfer error acknowledge.

SECTION 14
ENDIAN MODES

A general description of the different endian modes can be found in the *PowerPC™ Microprocessor Family: The Programming Environments* (MPCFPE/AD) manual that is available from Motorola. The MPC801 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 MPC801 bus. The MPC801 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 Individually 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 MPC801.

To program the configurations, refer to the table below.

Table 14-2. Endian Mode Programming For Core Data Structures

MODE	MSR_{LE} (AND MSR_{ILE})	$DCCST_{LES}$
Big-Endian Mode	0	0
Little-Endian Mode	0	1
PowerPC Little-Endian Mode	1	0
Reserved	1	1

The following hardware operations support the different endian modes:

- Address munging in the core is controlled by the MSR_{LE} bit.
- The MPC801 internal bus signal is driven by the master that informs the system interface unit to swap and perform address demunging or leave the current access as it is. Table 13-1 defines the $DCCST_{LES}$ bit for core and cache accesses.

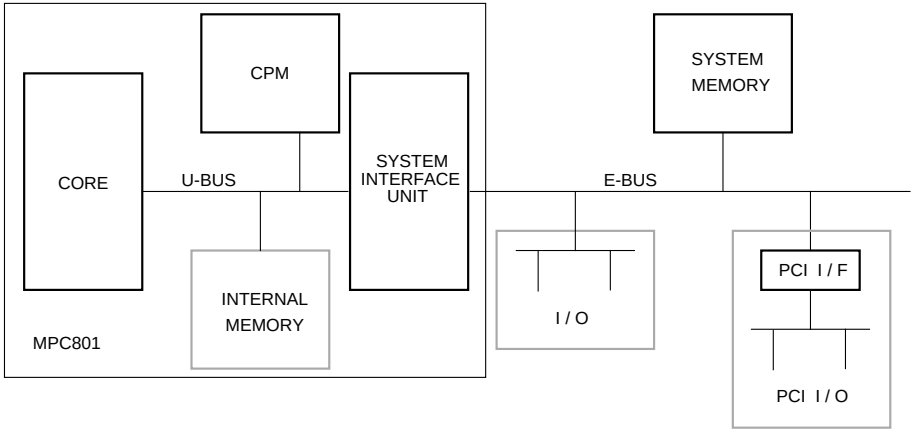


Figure 14-1. General MPC801 System Diagram

NOTE

A PCI bridge cannot be used in the little-endian system.

14.1 LITTLE-ENDIAN SYSTEM FEATURES

The following is a list of the little-endian system's main features.

- System memory organization and E-bus format is little-endian
- U-bus data, instruction cache, data cache, and internal memory format is big-endian
- Contains data access constraints, according to the PowerPC little-endian rules.
- Same byte order between the media and system memory
- For core accesses, swap and address demunging are performed by the system interface unit on the U-bus ↔ system path
- The core load/store unit swapper uses munged addresses to put the data on the right byte lanes when an access of half-word or byte is performed

The following tables describe little-endian program/data in the little-endian system that is built around the MPC801 for various port sizes.

Table 14-3. Little-Endian Program/Data Path Between the Register and 32-Bit Memory

FETCH LOAD STORE TYPE	LITTLE-ENDIAN ADDR	U-BUS AND CACHES ADDR	EXTERNAL BUS ADDR	DATA IN THE REGISTER				U-BUS AND CACHES FORMAT				E-BUS FORMAT				LITTLE-ENDIAN PROGRAM/DATA			
				0	1	2	3	0	1	2	3	0	1	2	3	3	2	1	0
Word	0	0	0	11	12	13	14	11	12	13	14	14	13	12	11	11	12	13	14
Half-word	0	2	0			21	22			21	22	22	21					21	22
Half-word	2	0	2			31	32	31	32					32	31	31	32		
Byte	0	3	0				‘a’				‘a’	‘a’							‘a’
Byte	1	2	1				‘b’			‘b’			‘b’					‘b’	
Byte	2	1	2				‘c’		‘c’					‘c’			‘c’		
Byte	3	0	3				‘d’	‘d’							‘d’	‘d’			

Table 14-4. Little-Endian Program/Data Path Between the Register and 16-Bit Memory

FETCH/ LOAD STORE TYPE	LITTLE-ENDIAN ADDR	U-BUS AND CACHES ADDR	EXTERNAL BUS ADDR	DATA IN THE REGISTER				U-BUS AND CACHES FORMAT				E-BUS FORMAT				LITTLE-ENDIAN PROGRAM/DATA			
				0	1	2	3	0	1	2	3	0	1	2	3	3	2	1	0
Word	0	0	0 2	11	12	13	14	11	12	13	14	14 12	13 11					13 11	14 12
Half-word	0	2	0			21	22			21	22	22	21					21	22
Half-word	2	0	2			31	32	31	32			32	31					31	32
Byte	0	3	0				‘a’				‘a’	‘a’							‘a’
Byte	1	2	1				‘b’			‘b’			‘b’					‘b’	
Byte	2	1	2				‘c’		‘c’			‘c’							‘c’
Byte	3	0	3				‘d’	‘d’					‘d’					‘d’	

Table 14-5. Little-Endian Program/Data Path Between the Register and 8-Bit Memory

FETCH/ LOAD STORE TYPE	LITTLE-ENDIAN ADDR	U-BUS AND CACHES ADDR	EXTERNAL BUS ADDR	DATA IN THE REGISTER				U-BUS AND CACHES FORMAT				E-BUS FORMAT				LITTLE-ENDIAN PROGRAM/DATA			
				0	1	2	3	0	1	2	3	0	1	2	3	3	2	1	0
Word	0	0	0 1 2 3	11	12	13	14	11	12	13	14	14 13 12 11							14 13 12 11
Half-word	0	2	0 1			21	22			21	22	22 21							22 21
Half-word	2	0	2 3			31	32	31	32			32 31							32 31
Byte	0	3	0				‘a’				‘a’	‘a’							‘a’
Byte	1	2	1				‘b’			‘b’		‘b’							‘b’
Byte	2	1	2				‘c’		‘c’			‘c’							‘c’
Byte	3	0	3				‘d’	‘d’				‘d’							‘d’

14.2 BIG-ENDIAN SYSTEM FEATURES

The following is a list of the big-endian system’s main features:

- Caches, U-bus, E-bus, system memory, and I/O organization format is big-endian
- Same byte order between the media and system memory
- The PCI bridge can operate in big-endian mode

14.3 POWERPC LITTLE-ENDIAN SYSTEM FEATURES

The following is a list of the PowerPC little-endian system’s main features:

- Caches, U-bus, E-bus, system memory, and E-bus attached I/O organization format is big-endian.
- PCI bus format is little-endian
- Contains data access constraints, according to the PowerPC little-endian rules.
- Address munging in the core, according to Table 14-1.
- 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 mechanisms of the instruction and data caches operate less efficiently when address munging is performed on the cache accesses. Therefore, you should expect some performance degradation when you are working in this mode.

14.4 SETTING THE ENDIAN MODE OF OPERATION

Dynamic switching between the endian modes is not effectively supported. The mode should be set early in the reset routine and remain that way. The MPC801 core is in big-endian mode 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 that resides (preferably) in a physical address ended by 3'b100. The instruction executed next is fetched from this address plus 8. If the instruction resides in an address ending with 3'b000, then this instruction is executed twice because of address munging.

The instruction to transfer the system back to the big-endian resides in an address ended by 3'b000. The next instruction is fetched from this address plus 12. Transferring to the little-endian mode (setting of bit $DCCST_{LES}$ in the data cache) should be performed by the **mtspr** instruction that resides at an address ending with 3'b000. Further instructions should reside in the little-endian format of the external system memory and in the big-endian format of the internal memory if it exists.

SECTION 15
MEMORY CONTROLLER

The memory controller controls a maximum of 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. The flexibility of the memory controller allows you to implement memory systems with very specific timing requirements. It supports external address multiplexing, periodic timers, and timing generation for row and column address strobes to create a glueless interface to DRAM devices. The periodic timers allow refresh cycles to be initiated while the address muxing provides row and column addresses.

You can define different timing patterns for the control signals that govern a memory device. These patterns show how the external control signals behave in read-access, write-access, burst read-access, burst write-access requests, or when the periodic timers reach the maximum programmed value for refresh operation.

15.1 FEATURES

The following list summarizes the main features of the memory controller:

- Supports a Maximum of Eight Memory Banks
- Provides a General-Purpose Chip-Select Machine
- Provides Two User-Programmable Machines

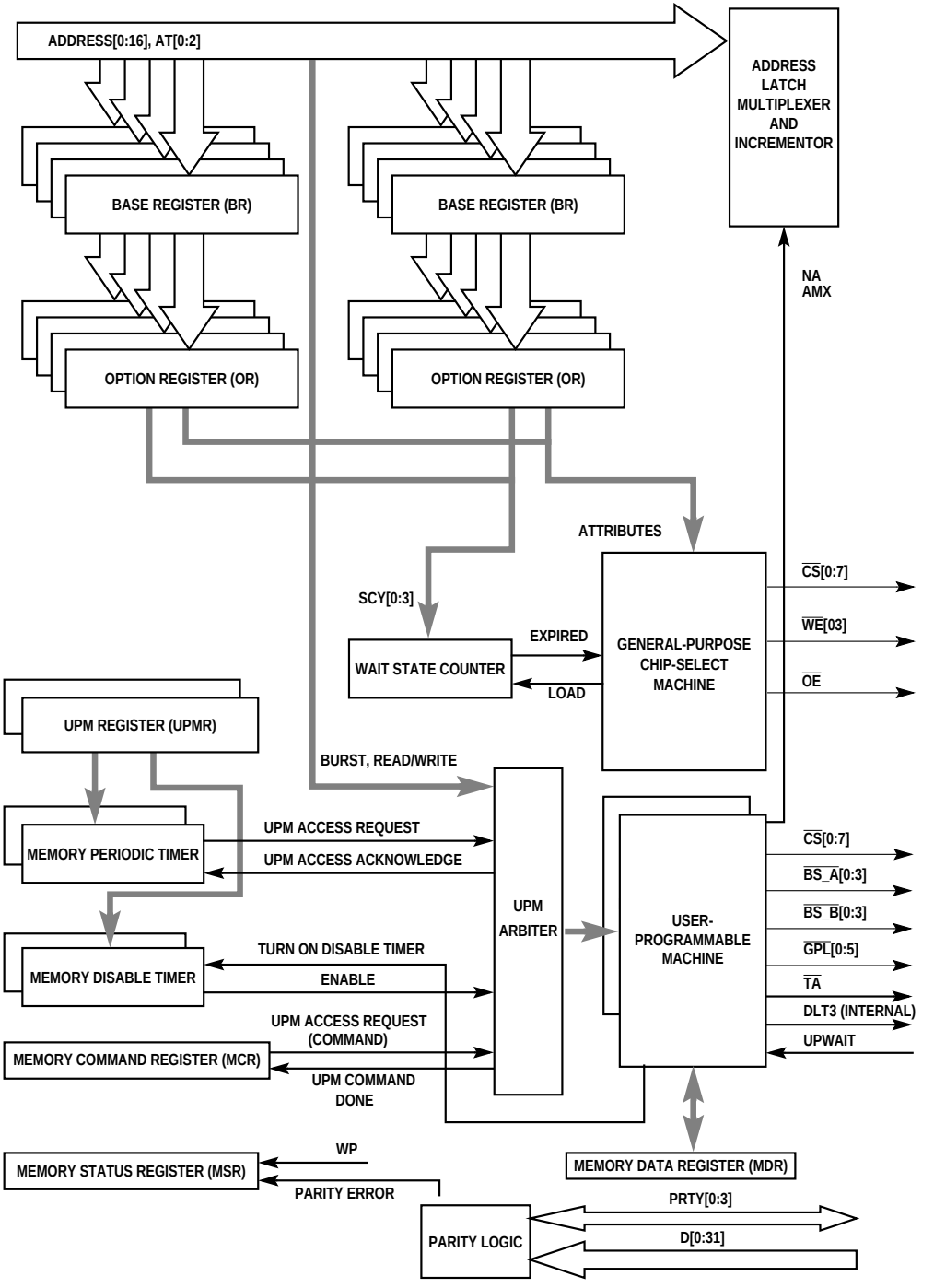


Figure 15-1. Memory Controller Block Diagram

15.2 BASIC ARCHITECTURE

The memory controller consists of three machines:

- General-purpose chip-select machine
- User-programmable machine A
- User-programmable machine B

Each bank can be assigned to any one of these machines via the MS bits in the base register as illustrated in Figure 15-3. When an access to one of the memory banks is initiated, the corresponding machine takes ownership of the external signals that control access until the cycle terminates. The general-purpose chip-select machine provides a glueless interface to EPROM, SRAM, Flash EPROM, and other peripherals. The general-purpose chip-selects are available on the $\overline{CS}$[0:7] signals. $\overline{CS0}$ is also the global (boot) chip-select that is used to access the boot EPROM. The chip-select allows 0 to 30 wait states.

Some features are common to all eight memory banks. The full 32-bit decode is available, even if all 32 address bits are not visible outside the MPC801. Since there is only a 26-bit address bus, the memory controller related to the external master address has a 32-bit address whose six most-significant bit s are equal to 0. The memory controller only uses 17 most-significant bit addresses for address decoding. Each memory bank includes a variable block size of 32K or 64K at a maximum of 256M. Parity can be generated and checked for any memory bank and each memory bank can be selected for read-only or read/write operation. For system protection purposes, accessing a memory bank can be restricted to certain address type codes. For additional flexibility, address type comparison provides a mask option.

The memory controller functionality allows the design of MPC801-based systems with little or no glue logic required. In Figure 15-2, $\overline{CS0}$ is used as the 16-bit boot EPROM and $\overline{CS1}$ is used for the 32-bit DRAM as the $\overline{RAS}$ signal. The $\overline{BS\_A}$[0:3] signals are used as the $\overline{CAS}$ signals on the DRAM.

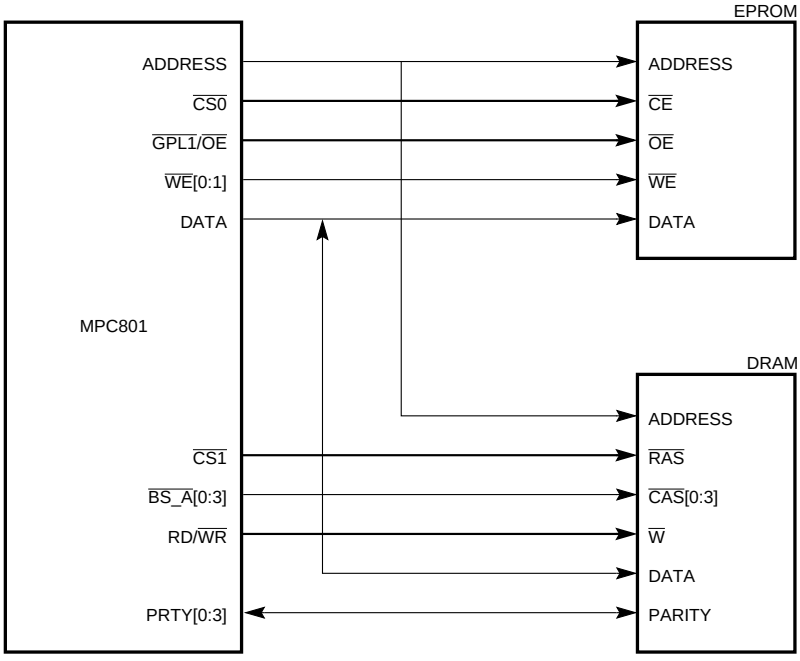


Figure 15-2. MPC801 Simple System Configuration

The two user-programmable machines—UPMA and UPMB—in the memory controller provide a flexible interface to many types of memory devices. Each one can simultaneously control the address multiplexing necessary to access DRAM devices, the timing of the $\overline{BS}$ signals, and the timing of the $\overline{GPL}$ signals. Each memory bank can be assigned to any user-programmable machine, so that each one controls eight $\overline{CS}$ signals.

Each user-programmable machine (UPM) is a RAM-based machine controlled by the software. The software toggles the memory controller external signals when an external single word read/write access or an external burst read/write access is initiated by an internal or external master. The user-programmable machine also controls address multiplexing, address increment, and 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.

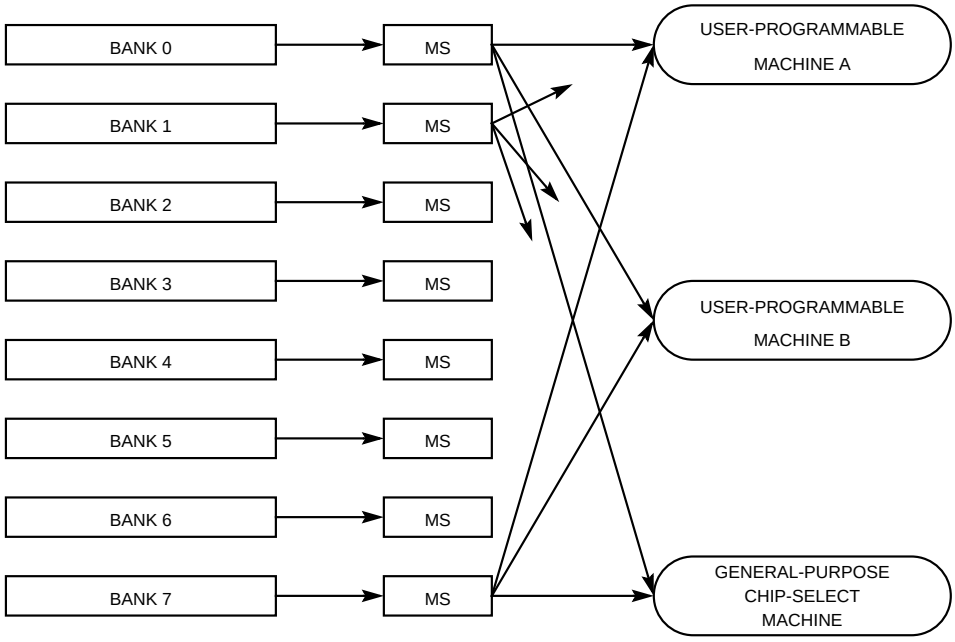


Figure 15-3. Memory Controller Machine Selection

When an address match is found in one of the memory bank's chip-select ranges, the base register's MS bits define the machine that handles the memory access. See Figure 15-4 for details.

The memory controller provides four PRTY signals, one for each data byte on the MPC801 system bus.The parity on the bus is only checked if the memory bank accessed in the current transaction has parity enabled. Parity checking/generation can be enabled for a specific memory bank in the base register. The type of parity is defined in the SIU module configuration register. Also, system protection is provided by defining each memory bank as read-only or read/write.

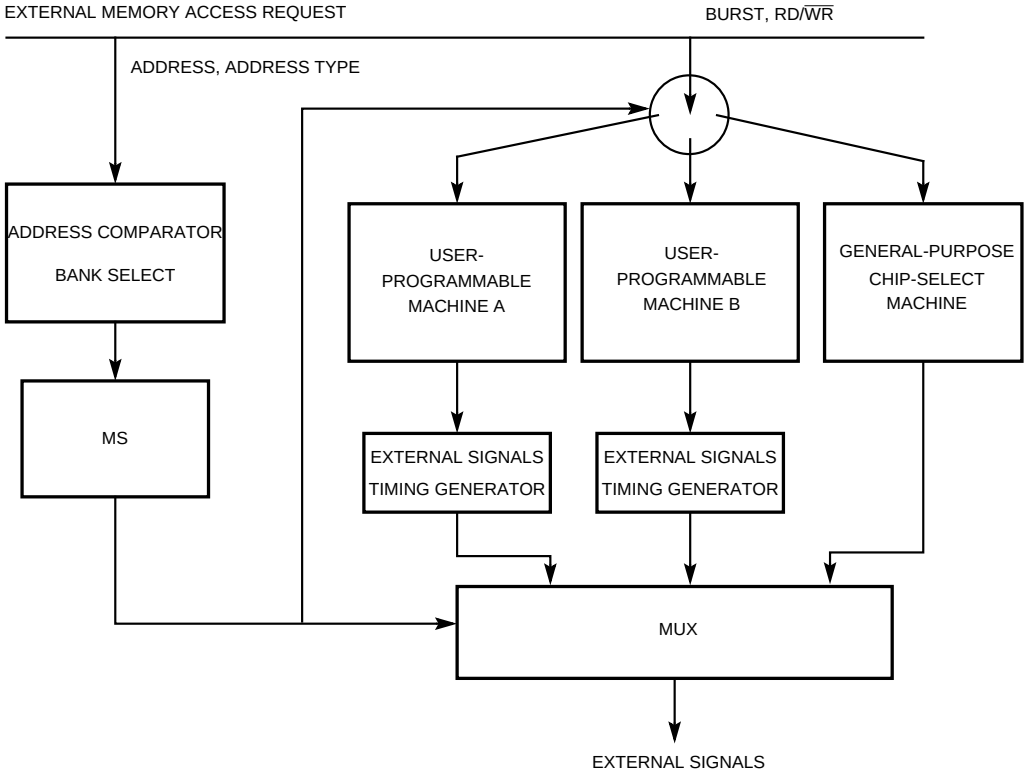


Figure 15-4. Memory Controller Basic Operation

15.2.1 Registers Associated with the Memory Controller

The status bits for each one of the memory banks are in the memory controller status (MSTAT) register and there is only one MSTAT for the entire memory controller. Each memory bank has a base register (BR) and an option register (OR). The MSTAT register reports write-protect violations that occur and parity errors for every bank. The BR*x* and OR*x* registers are specific memory to bank *x*. The BR contains a V bit that indicates when there is valid register information for that chip-select.

Each of the option registers define the attributes for the general-purpose chip-select machine when the corresponding bank is accessed. The option registers also define the initial address multiplexing for a memory cycle controlled by a user-programmable machine. The machine A mode register (MAMR) and machine B mode register (MBMR) define most of the global features for the user-programmable machines.

The memory command register (MCR) and memory data register (MDR) are used to initialize the user-programmable machine's RAM and to specify which pattern the software must run. The memory address register (MAR) allows a specific pattern to output this register's data to the address pins.

15.2.1.1 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 or even bytes and even half-words on data bus bits 0 through 15. Predefined 32-bit ports can be accessed as odd bytes, even bytes, odd half-words, even half-words, or words on word boundaries. The port size is specified by the PS bits in the base register.

15.2.1.2 WRITE-PROTECT CONFIGURATION . The WP bit of the 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 register.

15.2.1.3 ADDRESS AND ADDRESS SPACE CHECKING . The defined base address is written to the base register. The address mask bits for that address are written to the option register. The address type access value, if preferred, is written to the AT bits in the base register. The ATM bits in the option register can be used to mask this selection. If address type checking is not preferred, the ATM bits should be programmed 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 the base and option registers 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.

NOTE

When external masters access slaves on the bus, the internal AT[0:2] signals to the memory controller are forced to ‘100’.

15.2.1.4 PARITY GENERATION AND CHECKING . Parity can be configured for any bank. It is generated and checked on a per-byte basis using the PRTY[0:3] signals for the bank if the PARE bit is set in the base register. The OPAR bit determines the type of parity—odd or even. Any parity error results in the assertion and interrupt generation of the associated PER*x* bit in the MSTAT register. Refer to **Section 12.12.1.5 Transfer Error Status Register** for details.

15.2.1.5 TRANSFER ERROR ACKNOWLEDGE GENERATION . An internal transfer error indication signal is asserted by the memory controller when a parity error occurs or by the bus monitor of the system interface unit as the result of a write-protect violation.

15.2.2 The General-Purpose Chip-Select Machine

The general-purpose chip-select machine (GPCM) allows a glueless and flexible interface between the MPC801 and SRAM, EPROM, FEPROM, ROM devices, and external peripherals. If the MS bits in the BR*x* of the selected bank select the general-purpose chip-select machine, the attributes for the memory cycle initiated are taken from the OR*x* register. These attributes include the CSNT, ACS, SCY, TRLX, EHTR, and SETA fields.

Anywhere from 0 to 30 wait states can be programmed for $\overline{TA}$ generation. The $\overline{WE}$ signals are available for each byte that is written to memory. An $\overline{OE}$ signal is provided to eliminate external glue logic. The memory banks selected to operate with the general-purpose chip-select machine have features unique to that machine. On system reset, a global (boot) chip-select is available to provide a boot ROM chip-select before the system is fully configured. Next, the banks selected to operate with the general-purpose chip-select machine support an option to output the $\overline{CS}$ signal at different timings with respect to the external address bus. $\overline{CS}$ can be output in one of the following configurations:

- Simultaneous with an 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 general-purpose chip-select machine allows you to connect to devices that have long disconnect times on data by delaying new bus transactions addressing other memory banks for additional clock cycles. Finally, the banks selected to operate with the general-purpose chip-select machine support external cycle termination by sensing the $\overline{TA}$ signal that is asserted by the addressed external slave.

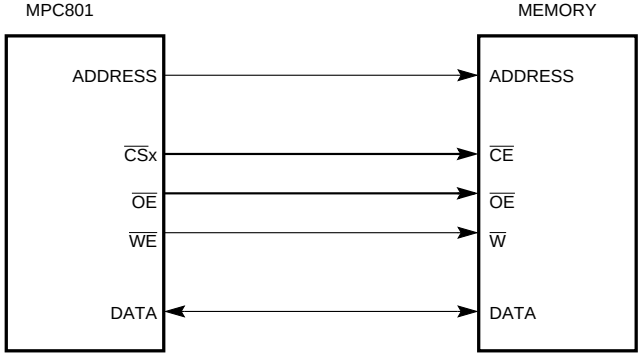


Figure 15-5. MPC801 GPCM–Memory Devices Interface

Figure 15-5 describes the basic connection between the MPC801 and a “static” memory device. In this case, $\overline{CS}x$ is connected directly to the $\overline{CE}$ signal of the memory device. The $\overline{WE}$ signals are connected to the respective $\overline{W}$ pin in the memory device where each $\overline{WE}$ signal corresponds to a different data byte. As illustrated in Figure 15-6, the $\overline{CS}x$ timing is the same as the address lines output. The strobes for the transaction are supplied by the $\overline{OE}$ or $\overline{WE}$ signals, depending on the transaction direction (read or write). This $\overline{CS}$ timing is generated when the ACS bits in the corresponding OR*x* are set to ‘00’.

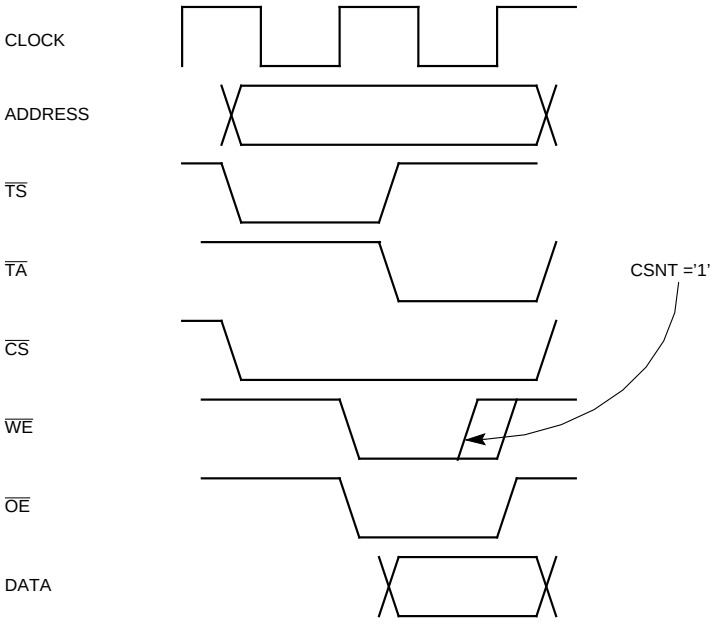


Figure 15-6. MPC801 GPCM–Memory Device Basic Timing (ACS = 00,TRLX = 0)

Figure 15-7 illustrates the basic connection between the MPC801 and an external peripheral device. In this case CS*x* is connected directly to the CE signal of the memory device and the R/W signal is connected to the respective R/W in the peripheral device. In this situation, the CS*x* signal is the strobe output for the memory access.

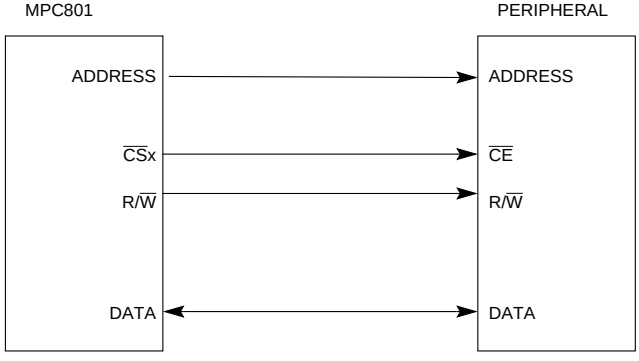


Figure 15-7. MPC801 GPCM–Peripheral Device Interface

Figure 15-8 illustrates the $\overline{CSx}$ timing as defined by the setup time required between the address and $\overline{CE}$ signals. The MPC801 memory controller allows you to specify the $\overline{CS}$ timing to meet this requirement through the ACS field of the option register.

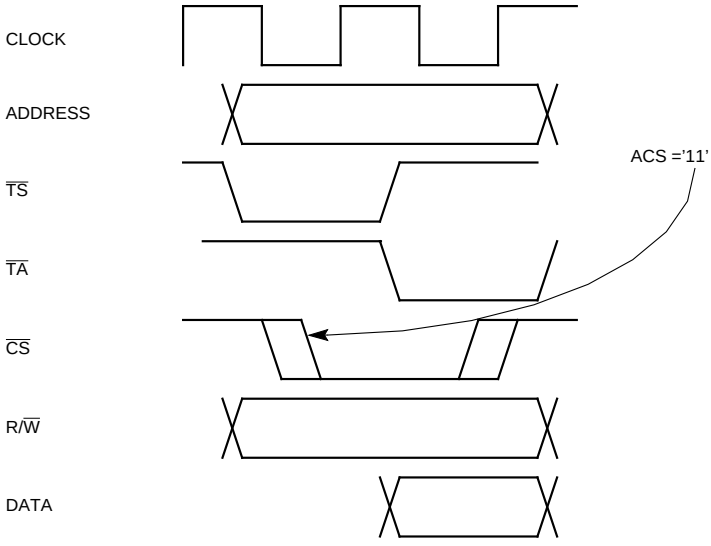


Figure 15-8. MPC801 GPCM–Peripheral Device Basic Timing (ACS = 10, ACS = 11,TRLX = 0)

The general-purpose chip-select machine also provides an attribute that controls the negation timing of the appropriate strobe in write cycles. When this attribute is asserted, the strobe is negated one quarter of a clock before the normal case. For example, when ACS '00' and CSNT == '1', $\overline{WE}$ is negated one quarter of a clock earlier and when ACS <> '00' and CSNT == '1', $\overline{WE}$ and $\overline{CS}$ are negated one quarter of a clock earlier. For more information refer to Figures 15-6 and 15-8.

The TRLX field is provided for memory systems that require more relaxed timing between signals. When TRLX is set and ACS <> 00, an additional cycle between the address and strobes is inserted by the MPC801 memory controller. See Figure 15-9 for more information. When TRLX is set and CSNT == '1' in a write-memory access, the strobe lines ($\overline{WE}$ and $\overline{CS}$, if ACS <> '00') are negated one clock earlier than in the normal case. Refer to Figures 15-10 through 15-12 for details. When a bank is selected to operate with external transfer acknowledge (SETA == '1') and TRLX == '1', the memory controller does not support external devices providing $\overline{TA}$ to complete the transfer with zero wait states. The minimum access duration in this case is 3 clock cycles.

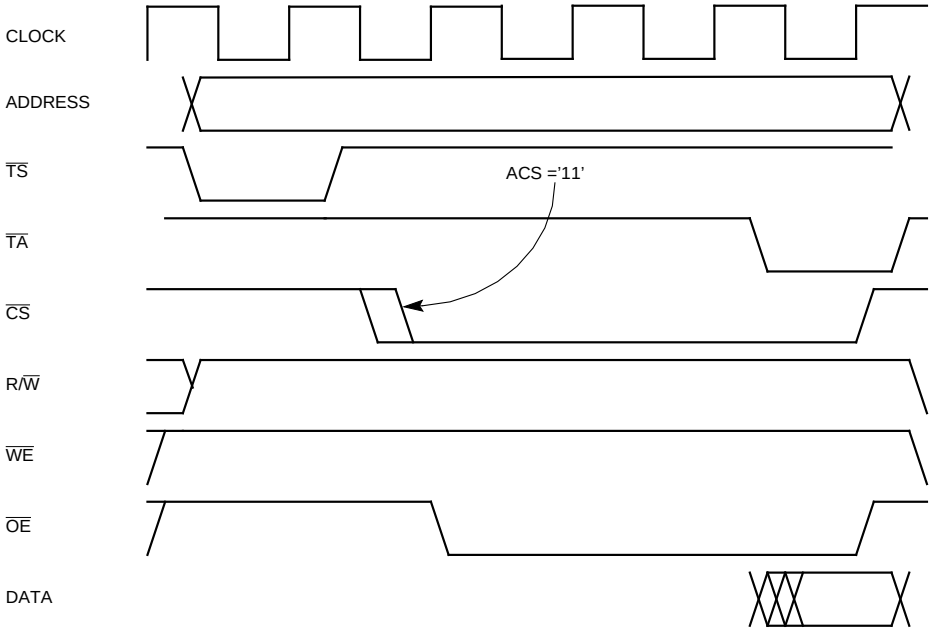


Figure 15-9. MPC801 GPCM–Relaxed Timing–Read Access (ACS = 10, ACS = 11, SCY = 1, TRLX =1)

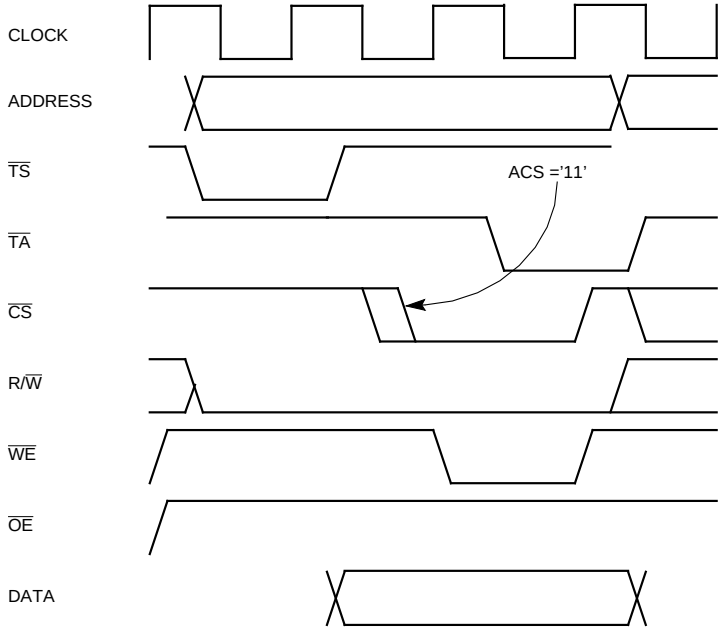


Figure 15-10. MPC801 GPCM–Relaxed Timing–Write Access (ACS = 10, ACS = 11, SCY = 0, CSNT = 0, TRLX =1)

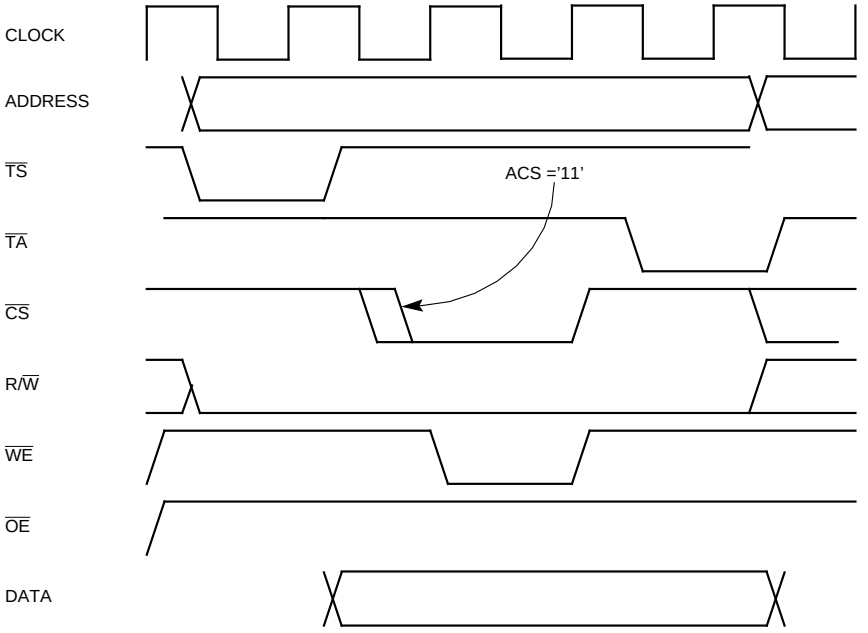


Figure 15-11. MPC801 GPCM–Relaxed Timing–Write Access (ACS = 10, ACS = 11, SCY = 0, CSNT = 1, TRLX =1)

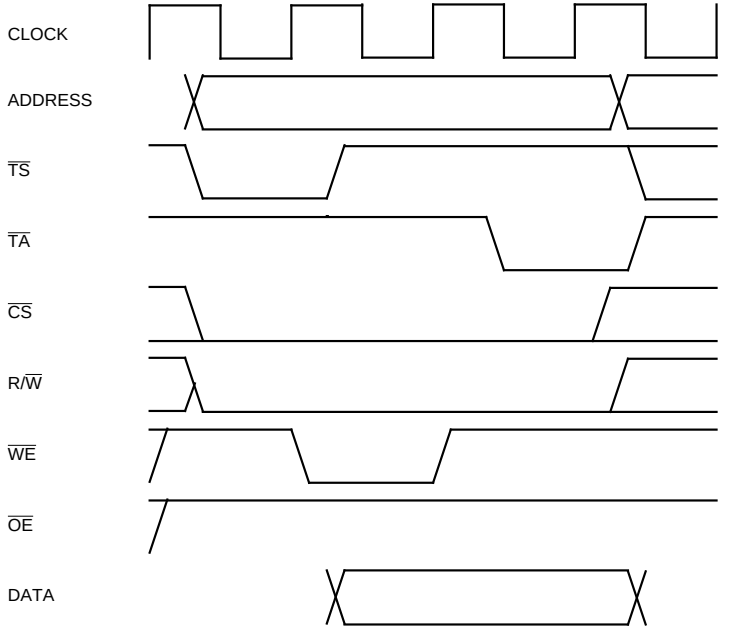


Figure 15-12. MPC801 GPCM–Relaxed Timing–Write Access (ACS = 00, SCY = 0, CSNT = 1, TRLX =1

15.2.2.1 PROGRAMMABLE WAIT STATE CONFIGURATION . The general-purpose chip-select machine supports internal $\overline{TA}$ generation. It allows “fast” accesses to external memory through an internal bus master and a maximum of 17 clock accesses. This can be done by programming the SCY bits in the option register. The internal $\overline{TA}$ generation mode is enabled if the SETA bit in the option register is negated. If the $\overline{TA}$ pin is externally asserted at least two clock cycles before the wait state counter has expired, the current memory cycle is terminated. When TRLX is set, the number of wait states inserted by the memory controller is defined by NumberofWaitStates = 2 x SCY.

15.2.2.2 EXTENDED HOLD TIME ON READ ACCESSES . Slow memory devices that require a long delay on data read accesses should set the EHTR bit in the corresponding option register. Any MPC801 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.

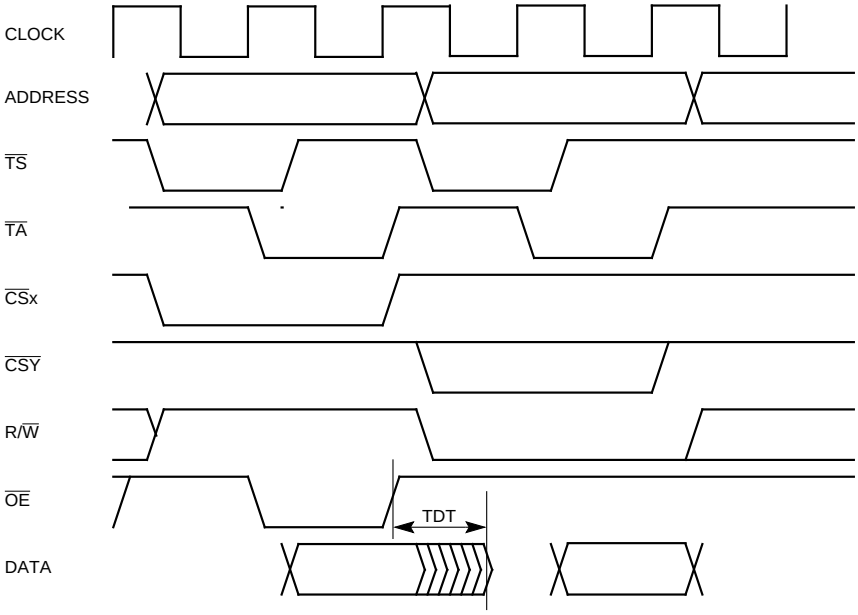


Figure 15-13. MPC801 Consecutive Accesses Write After Read–(ORx-EHTR = 0)

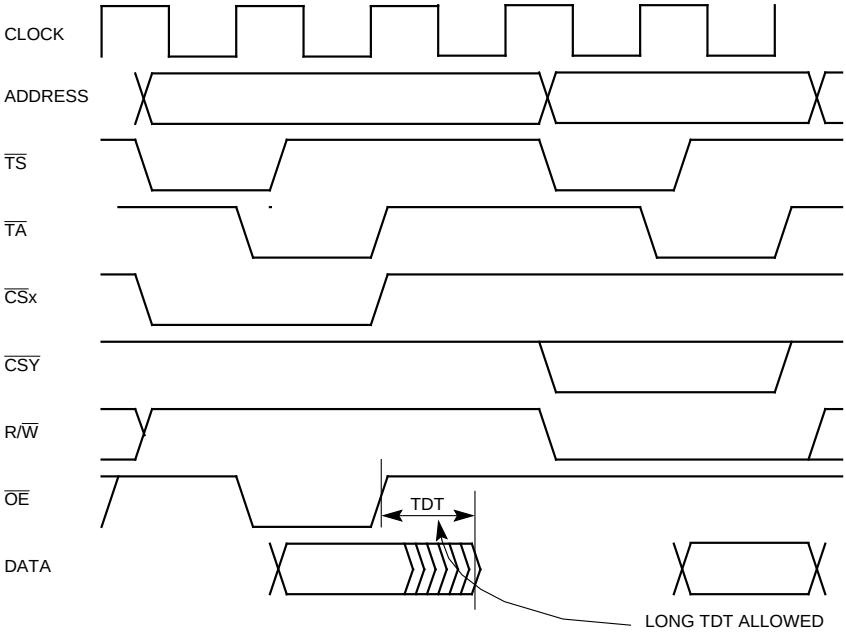


Figure 15-14. MPC801 Consecutive Accesses Write After Read–(ORx-EHTR = 1)

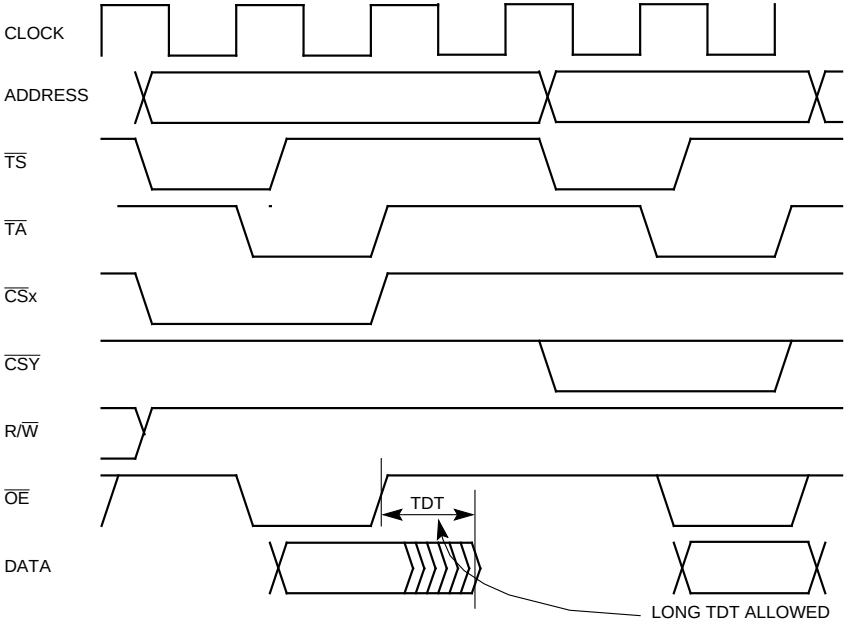


Figure 15-15. MPC801 Consecutive Accesses Read After Read From Different Banks–(ORx-EHTR = 1)

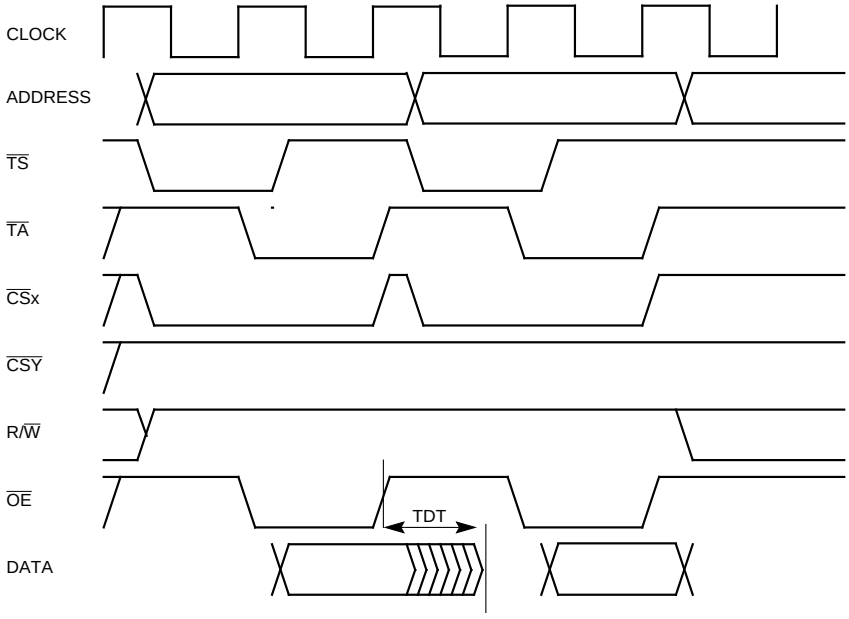


Figure 15-16. MPC801 Consecutive Accesses Read After Read From Same Bank– (ORx-EHTR = 1)

15.2.2.3 GLOBAL CHIP-SELECT OPERATION . Global (boot) chip-select operation allows address decoding for a boot ROM before system initialization occurs. The CS0 signal is the global chip-select output and its operation differs from the other external chip-select outputs on system reset. When the MPC801 internal core begins accessing memory at system reset, 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 RSTCONF and D[0:31] pins. See **Section 4.3 How to Configure Reset** for more information. Setting these 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. CS0 operates this way until the first write to the CS0 option register (OR0) is made and it can be programmed to continue decoding a range of addresses once the preferred address range is loaded into base register 0. After the first write to 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 Field 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
$\overline{BI}$	1
SCY[0:3]	1111
SETA	0
TRLX	1
EHTR	0

15.2.2.4 SRAM INTERFACE . Figure 15-17 illustrates a simple connection between an SRAM device and the MPC801.

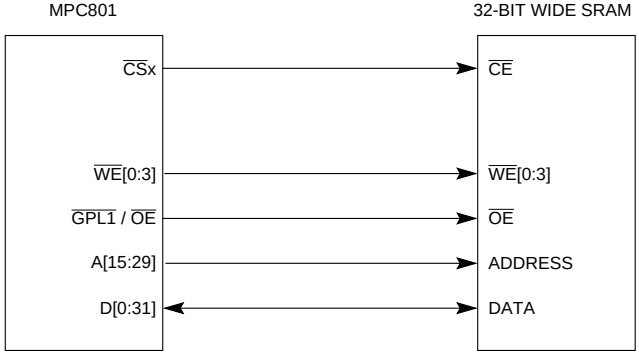


Figure 15-17. MPC801–Simple 128K SRAM Configuration

15.2.2.5 GPCM EXTERNAL ASYNCHRONOUS MASTER SUPPORT . Figure 15-18 illustrates the basic interface between an asynchronous external master and the MPC801 to allow connection to “static RAM” memory.

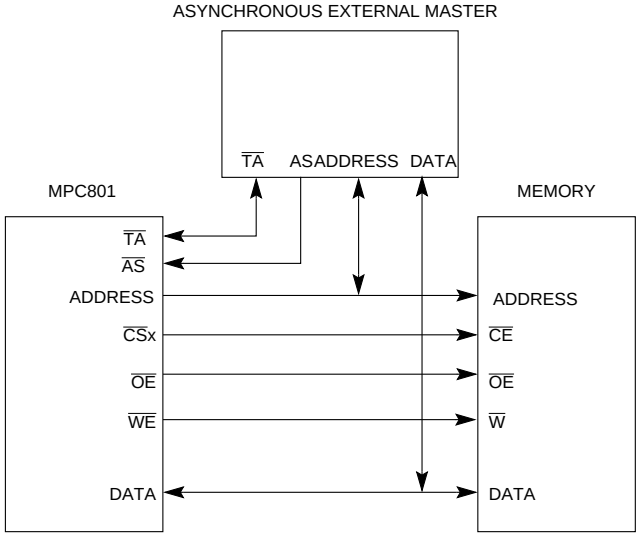


Figure 15-18. MPC801–Asynchronous External Master Configuration For GPCM–Handled Memory Devices

Figure 15-19 illustrates the timing for TRLX = 0 when an external asynchronous master accesses SRAM. The $\overline{TA}$ signal remains asserted with the $\overline{WE}$ and $\overline{OE}$ signals until $\overline{AS}$ is negated by the external master.

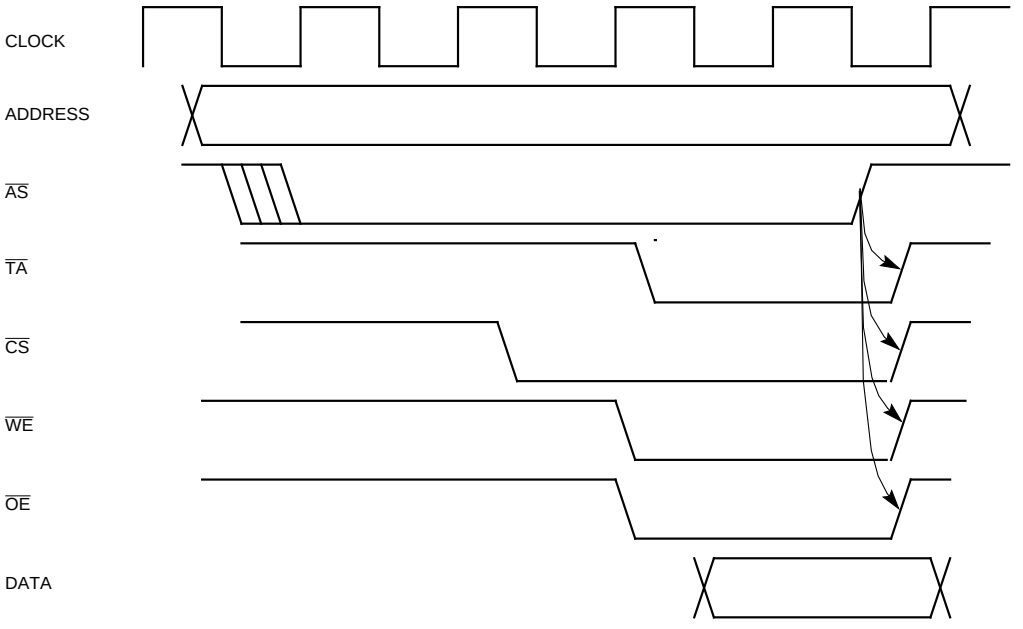


Figure 15-19. Asynchronous Master GPCM–Memory Devices Basic Timing (TRLX = 0)

When an external asynchronous master performs an access to a memory device via the general-purpose chip-select machine in the MPC801 memory controller, the CSNT bit in the option register is configured as “don’t care”.

15.2.3 User-Programmable Machines

The user-programmable machine (UPM) is a flexible interface that connects to a wide range of memory devices. At the heart of the user-programmable machine 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 a resolution of one quarter the system clock period on byte- and chip-select lines. There are three possible ways to initiate a UPM cycle:

- When an internal or external master requests an external memory access.
- When an internal periodic timer expires, thus requesting a transaction.
- When a valid command is written to the memory command register.

Figure 15-20 illustrates basic user-programmable machine operation.

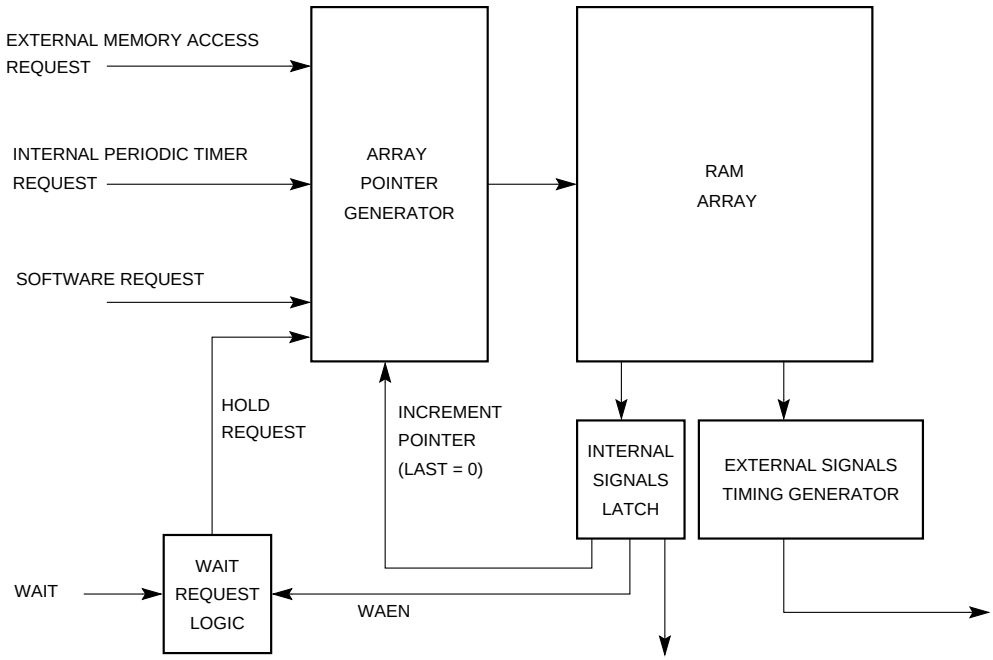


Figure 15-20. General Description of a UPM

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 of its base register selects the user-programmable machine that will handle the memory access. A service request from the selected user-programmable machine is required for this access.

When a request is initiated, the first location pointed to in the RAM array can be one of the following fixed addresses that is determined by the requested cycle's attributes.

- 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 user-programmable machine has a machine mode register (MAMR and MBMR) that defines the general attributes for operation. The PTA bits of the MAMR and the PTB bits of the 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. When the periodic interrupt timer request is serviced, the first location pointed to in the RAM array 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.

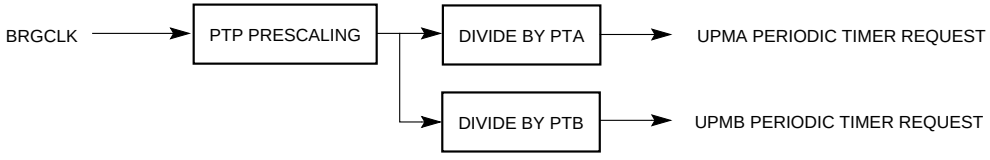


Figure 15-21. Memory Periodic Timer Request Block Diagram

The software can request a special service from the user-programmable machine by writing a valid command to the memory command register (MCR) and memory data register (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 user-programmable machine and about specific strobes that control internal memory controller resources.

When the 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. A memory disable timer (MDTA or MDTB) is associated with each user-programmable machine. This timer counts down to zero starting at the value programmed in the DSA/DSB field of the MAMR/MBMR. The one-shot timer trigger is controlled by the TODT in the RAM array. When an access to a memory bank controlled by UPM*x* has the memory disable timer turned on, a new user-programmable machine access to this bank is held off until the timer expires. In general, the disable timer is a simple way to assure that a $\overline{\text{RAS}}$ 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 the MAMR and AMB[0:2] in the MBMR) allows each of the user-programmable machines to select an address multiplexing configuration. The AMX bits in the RAM array controls the multiplexing/nonmultiplexing value of the address pins on 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 of 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. Figures 15-22 and 15-23 illustrate the clock scheme of the user-programmable machines in the memory controller. Table 20-1 shows the value of the external signals that can be changed if specified in the RAM after any of the edges of GCLK1 and GCLK2 and a circuit delay time.

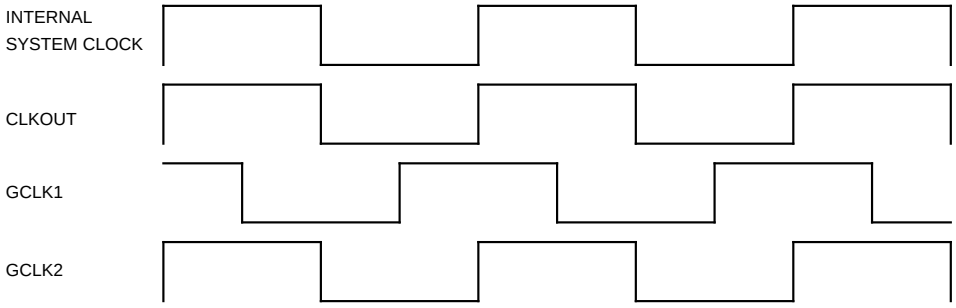


Figure 15-22. Memory Controller UPM Clock Scheme (For System_To CLKOUT Division Factor 1–EBDF = 00)

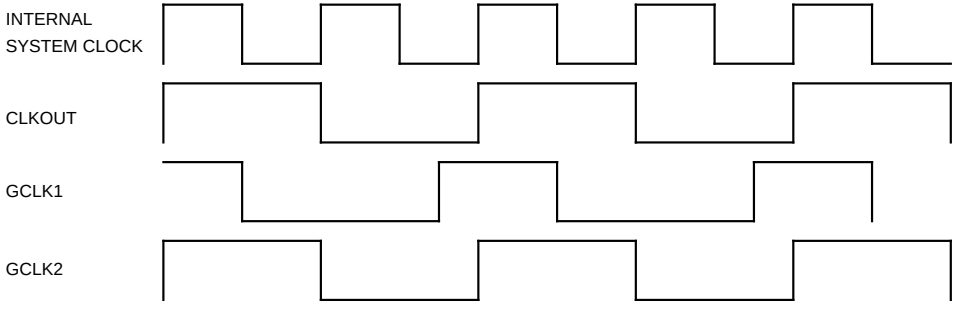


Figure 15-23. Memory Controller UPM Clock Scheme (For System_To CLKOUT Division Factor 2–EBDF = 01)

The $\overline{CS}$ signals are handled in a similar way, except that only the $\overline{CS}$ signal corresponding to the currently accessed bank is modified. The $\overline{BS}$ signal assertion and negation timing is also specified for each cycle in the RAM, but the final value of each one of these signals depends on the port size of the specified bank, the external address accessed, and the value of the TSIZ pins.

Figures 15-24 and 15-25 provide examples of how to control the timing of the $\overline{CS}x$, $\overline{GPL1}$, and $\overline{GPL2}$ pins. 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.

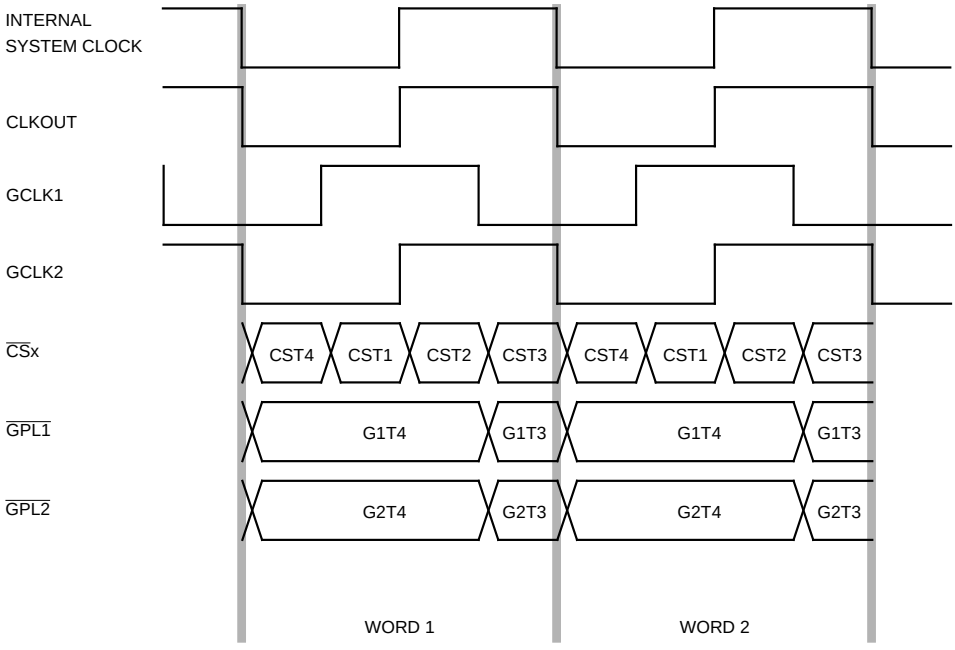


Figure 15-24. UPM Signals Timing Example (For System_To CLKOUT Division Factor 1–EBDF = 00)

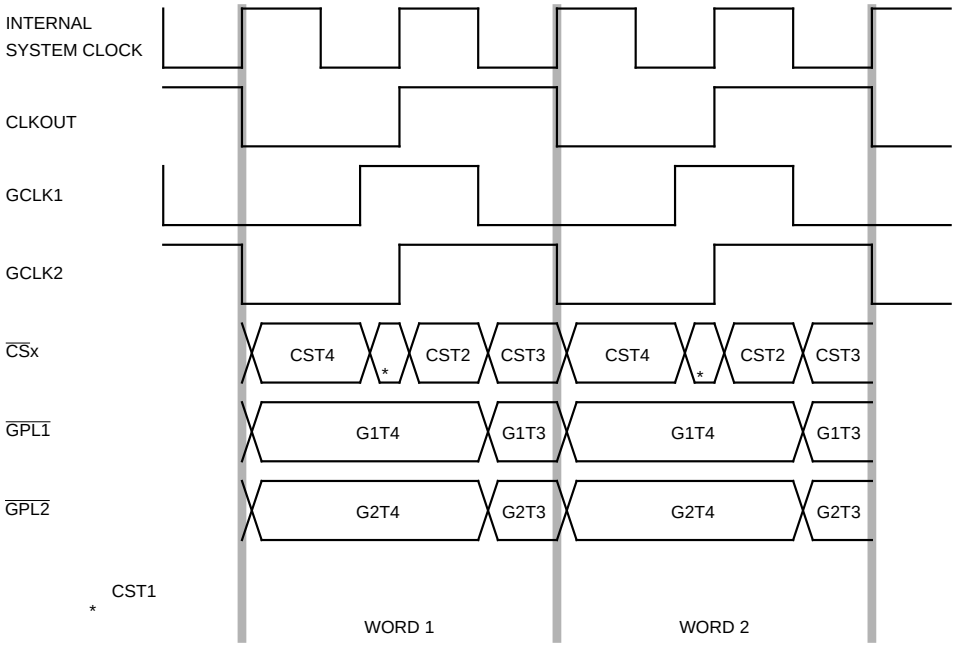


Figure 15-25. UPM Signals Timing Example
(For System_To CLKOUT Division Factor 2–EBDF = 01)

The RAM array size for each user-programmable machine is 64 locations deep and 32 bits wide as illustrated in Figure 15-26.

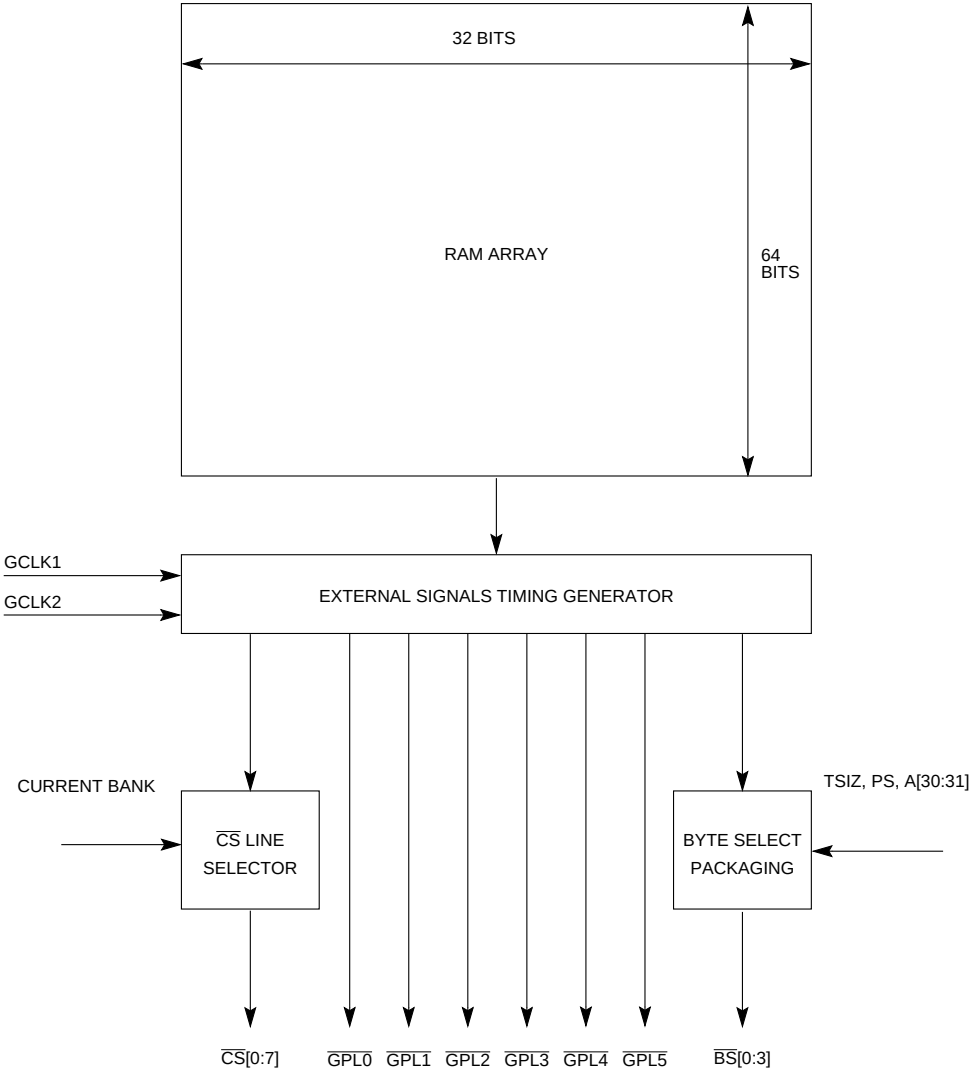


Figure 15-26. UPM External Signal Generation

15.2.3.1 RAM WORD STRUCTURE AND TIMING SPECIFICATION

The RAM word structure is illustrated in Figure 15-27 and described in Table 15-2. With typical DRAM, the $\overline{CS}$ signals correspond to $\overline{RAS}$ and the $\overline{BS}$ signals correspond to $\overline{CAS}$. Likewise, the $\overline{GPL}$ signals 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	RESERVED	RESERVED

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 means the value of the $\overline{CS}$ signal at the trailing edge of GCLK2 will be ‘0’ CST4 = 1 means the value of the $\overline{CS}$ signal at the trailing edge of GCLK2 will be ‘1’
1	CST1	CST1 = 0 means the value of the $\overline{CS}$ signal at the rising edge of GCLK1 will be ‘0’ CST1 = 1 means the value of the $\overline{CS}$ signal at the rising edge of GCLK1 will be ‘1’
2	CST2	CST2 = 0 means the value of the $\overline{CS}$ signal at the rising edge of GCLK2 will be ‘0’ CST2 = 1 means the value of the $\overline{CS}$ signal at the rising edge of GCLK2 will be ‘1’
3	CST3	CST3 = 0 means the value of the $\overline{CS}$ signal at the trailing edge of GCLK1 will be ‘0’ CST3 = 1 means the value of the $\overline{CS}$ signal at the trailing edge of GCLK1 will be ‘1’
4	BST4	BST4 = 0 means the value of the $\overline{BS}$ signals at the trailing edge of GCLK2 will be ‘0’ BST4 = 1 means the value of the $\overline{BS}$ signals at the trailing edge of GCLK2 will be ‘1’ NOTE: The final value of the $\overline{BS}$ signals depends on the value of the PS bits in the BR accessed, the value of the TSIZ signals for the access, and the value of the A[30:31] signals. For more information about the $\overline{BS}$ signals, see **Section 15.2.3.1 RAM Word Structure and Timing Specification**.

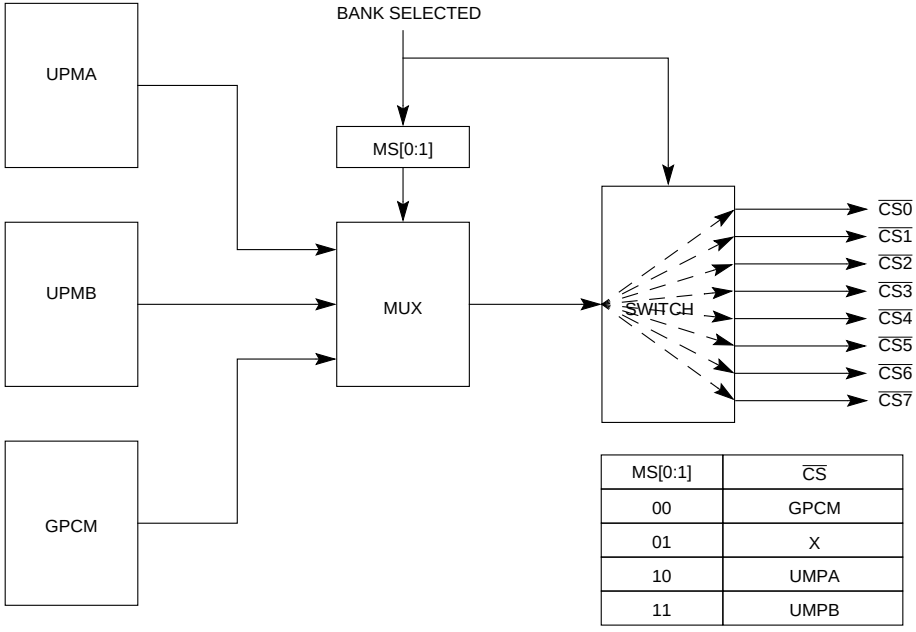
Table 15-2. UPM RAM Word (Continued)

BITS	MNEMONIC	FUNCTION
5	BST1	BST1 = 0 means the value of the $\overline{BS}$ signals at the rising edge of GCLK1 will be '0' BST1 = 1 means the value of the $\overline{BS}$ signals at the rising edge of GCLK1 will be '1' NOTE: The final value of the $\overline{BS}$ signals depends on the value of the PS bits in the BR accessed, the value of the TSIZ signals for the access and the value of the A[30:31] signals.
6	BST2	BST2 = 0 means the value of the $\overline{BS}$ signals at the rising edge of GCLK2 will be '0' BST2 = 1 means the value of the $\overline{BS}$ signals at the rising edge of GCLK2 will be '1' NOTE: The final value of the $\overline{BS}$ signals depends on the value of the PS bits in the BR accessed, the value of the TSIZ signals for the access, and the value of the A[30:31] signals.
7	BST3	BST3 = 0 means the value of the $\overline{BS}$ signals at the trailing edge of GCLK1 will be '0' BST3 = 1 means the value of the $\overline{BS}$ signals at the trailing edge of GCLK1 will be '1' NOTE: The final value of the $\overline{BS}$ signals depends on the value of the PS bits in the BR accessed, the value of the TSIZ signals for the access, and the value of the A[30:31] signals.
8-9	G0L(0:1)	G0L = 10 means the value of the $\overline{GPL0}$ signal at the trailing edge of GCLK2 will be '0' G0L = 11 means the value of the $\overline{GPL0}$ signal at the trailing edge of GCLK2 will be '1' G0L = 00 means the value of the $\overline{GPL0}$ signal at the trailing edge of GCLK2 will be as defined in the G0CL field in the MxMR'
10-11	G0H(0:1)	G0H = 10 means the value of the $\overline{GPL0}$ signal at the trailing edge of GCLK1 will be '0' G0H = 11 means the value of the $\overline{GPL0}$ signal at the trailing edge of GCLK1 will be '1' G0H = 00 means the value of the $\overline{GPL0}$ signal at the trailing edge of GCLK1 will be as defined in the G0CL field in the MxMR'
12	G1T4	G1T4 = 0 means the value of the $\overline{GPL1}$ signal at the trailing edge of GCLK2 will be '0' G1T4 = 1 means the value of the $\overline{GPL1}$ signal at the trailing edge of GCLK2 will be '1'
13	G1T3	G1T3 = 0 means the value of the $\overline{GPL1}$ signal at the trailing edge of GCLK1 will be '0' G1T3 = 1 means the value of the $\overline{GPL1}$ signal at the trailing edge of GCLK1 will be '1'
14	G2T4	G2T4 = 0 means the value of the $\overline{GPL2}$ signal at the trailing edge of GCLK2 will be '0' G2T4 = 1 means the value of the $\overline{GPL2}$ signal at the trailing edge of GCLK2 will be '1'
15	G2T3	G2T3 = 0 means the value of the $\overline{GPL2}$ signal at the trailing edge of GCLK1 will be '0' G2T3 = 1 means the value of the $\overline{GPL2}$ signal at the trailing edge of GCLK1 will be '1'
16	G3T4	G3T4 = 0 means the value of the $\overline{GPL3}$ signal at the trailing edge of GCLK2 will be '0' G3T4 = 1 means the value of the $\overline{GPL3}$ signal at the trailing edge of GCLK2 will be '1'
17	G3T3	G3T3 = 0 means the value of the $\overline{GPL3}$ signal at the trailing edge of GCLK1 will be '0' G3T3 = 1 means the value of the $\overline{GPL3}$ signal at the trailing edge of GCLK1 will be '1'
18	G4T4/DLT3	When GPL4_xDIS = 0 in the corresponding MxMR: G4T4/DLT3 = 0 means the value of the $\overline{GPL4}$ signal at the trailing edge of GCLK2 will be '0' G4T4/DLT3 = 1 means the value of the $\overline{GPL4}$ signal at the trailing edge of GCLK2 will be '1' When GPL4_xDIS = 1 in the corresponding MxMR: 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). 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: G4T3/WAEN = 0 means the value of the $\overline{\text{GPL4}}$ signal at the trailing edge of GCLK1 will be '0' G4T3/WAEN = 1 means the value of the $\overline{\text{GPL4}}$ signal at the trailing edge of GCLK1 will be '1' When GPL4_xDIS = 1 in the corresponding MxMR: 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 asserted. This condition lasts until the $\overline{\text{WAIT}}$ signal is negated.
20	G5T4	G5T4 = 0 means the value of the $\overline{\text{GPL5}}$ signal at the trailing edge of GCLK2 will be '0' G5T4 = 1 means the value of the $\overline{\text{GPL5}}$ signal at the trailing edge of GCLK2 will be '1'
21	G5T3	G5T3 = 0 means the value of the $\overline{\text{GPL5}}$ signal at the trailing edge of GCLK1 will be '0' G5T3 = 1 means the value of the $\overline{\text{GPL5}}$ signal 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 where the LOOP bit is '1' is marked as the LOOP START WORD. The next word in the same pattern where the LOOP bit is '1' is marked as the LOOP END WORD. The user-programmable machine runs the subpattern between the LOOP START WORD and LOOP END WORD many times as defined in the corresponding loop field of the MxMR.
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 means the value of the A[0:31] signals at the trailing edge of GCLK1 will be the address requested by the internal master for the external access. Ex: Column address. AMX = 10 means the value of the A[0:31] signals at the trailing edge of GCLK1 will be the address requested by the internal master for the external access multiplexed according to the one specified in the AMA/AMB bits of the MAMR/MBMR. Ex: Row address. AMX = 11 means the value of the A[0:31] signals at the trailing edge of GCLK1 will be the contents of the MAR. Ex: SDRAM mode initialization.
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. NA = 1 if the port size of the accessed bank is 16 bits, the value of the A[28:31] signals at the trailing edge of GCLK1 will be incremented by 2. NA = 1 if the port size of the accessed bank is 8 bits, the value of the A[28:31] signals at the trailing edge of GCLK1 will be incremented by 1. NA = 0 means the address increment is disabled. NOTE: The value of the NA bit is relevant only when the UPM serves a burst-read or burst-write request. Under other patterns this bit is reserved.
29	UTA	This line indicates the value of the $\overline{\text{TA}}$ signal sampled by the system interface unit in the current cycle. The $\overline{\text{TA}}$ signal is output at the rising edge of GCLK2.
30	TODT	TODT = 1 if 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 user-programmable machines) until the disable timer is expired. Ex: Precharge time.
31	LAST	LAST = 1 means the service to the UPM request is done

15.2.3.2 $\overline{\text{CS}}$ SIGNALS. If the MS bits in the BR*x* of the accessed memory bank selects the user-programmable machine on the currently requested cycle, the user-programmable machine can only affect the electrical value of the $\overline{\text{CS}}$*x* signal and the timing of the change is specified in the UPM internal RAM. Figure 15-28 illustrates how the $\overline{\text{CS}}$ signals are controlled by the user-programmable machines.



MS[0:1]	$\overline{\text{CS}}$
00	GPCM
01	X
10	UMPA
11	UMPB

Figure 15-28. $\overline{\text{CS}}$ Signal Control Model

15.2.3.3 BYTE SELECT SIGNALS. If the MS bits in the BR*x* of the memory bank being accessed selects the UPMA or UPMB to handle the current cycle, the UPMA or UPMB only determines the timing and value of the $\overline{BS}$ signals if it is allowed by the port size of the accessed bank, the transfer size of the transaction, and the address accessed. Figure 15-29 illustrates how the $\overline{BS}$ signals are controlled by the user-programmable machines.

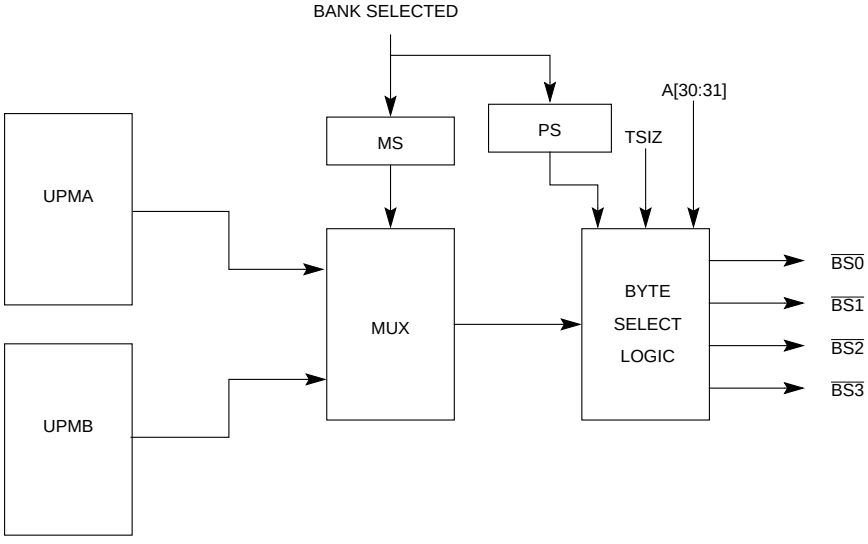


Figure 15-29. Byte Select Control Model

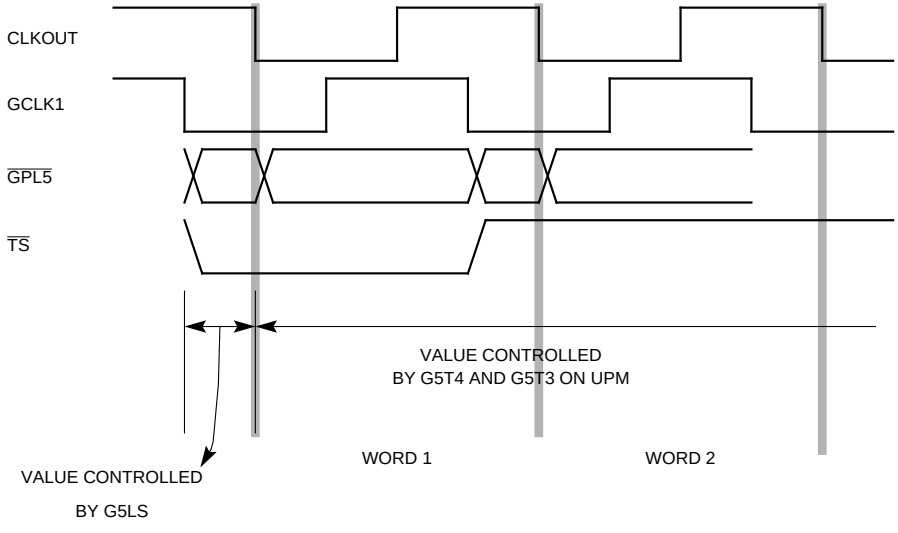
The upper-upper byte select ($\overline{BS0}$) indicates that the upper eight bits of the data bus (D[0:7]) contain valid data during a cycle and the upper-middle write enable ($\overline{BS1}$) indicates that the upper-middle eight bits of the data bus (D[8:15]) contain valid data during a cycle. The lower-middle write enable ($\overline{BS2}$) indicates that the lower-middle eight bits of the data bus (D[16:23]) contain valid data during a cycle and the lower-lower write enable ($\overline{BS3}$) indicates that the lower eight bits of the data bus contain valid data during a cycle. The manner in which the $\overline{BS}$ signals are affected in a transaction for a 32-, 16-, or 8-bit port is shown in Table 15-3. It should be noted that for a periodic timer request and a memory command request, the $\overline{BS}$ signals are only determined by the port size of the bank.

Table 15-3. Byte Select Enable Function

TRANSFER SIZE	TSIZ		ADDRESS		32-BIT PORT SIZE				16-BIT PORT SIZE				8-BIT PORT SIZE			
			A[30]	A[31]	$\overline{BS0}$	$\overline{BS1}$	$\overline{BS2}$	$\overline{BS3}$	$\overline{BS0}$	$\overline{BS1}$	$\overline{BS2}$	$\overline{BS3}$	$\overline{BS0}$	$\overline{BS1}$	$\overline{BS2}$	$\overline{BS3}$
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			

15.2.3.4 GENERAL-PURPOSE SIGNALS. The user-programmable machine controls each of the $\overline{GPL}$ signals using two bits in the UPM word. The two bits define the logical value of the signal to be changed at the falling edge of GCLK2 and/or GCLK1. $\overline{GPL5}$ and $\overline{GPL0}$ offer the following enhancements beyond the other $\overline{GPL}x$ signals:

- The logical value $\overline{GPL5}$ 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 $\overline{GPL0}$ signal can output the value of an address signal as specified in the corresponding M*x*MR that is controlled by the user-programmable machine. This is helpful when there needs to be some control of the value output to the address signals that connect to some types of memory devices.



15.2.3.5 LOOP CONTROL SIGNAL. The LOOP bit in the user-programmable machine allows you to run repetitive subpatterns included in a memory cycle pattern a specific number of times. The memory controller marks the first word that the LOOP bit is found asserted in a pattern 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 that the LOOP bit is found asserted in the same pattern is marked as the loop end word in the memory controller. 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 user-programmable machine continues with the word after loop end word. After exiting a loop, the next word read in the user-programmable machine 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 is loaded into the loop counter when the user-programmable machine services a request. 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

15.2.3.6 EXCEPTION HANDLING. When an access to a memory device is initiated by the MPC801 under UPM control on the memory controller, the external device may assert the $\overline{\text{TEA}}$ or $\overline{\text{RESET}}$ signal. The MPC801 tries to close the bus transfer immediately. The user-programmable machine in the memory controller provides a mechanism by which you can handle the memory control signals 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 bit in the user-programmable machine is set to 1, the next word read and run by the user-programmable machine is found at the fixed address “exception start address”—EXSA (RAM ADDRESS = 0x’3C). Normally, there is a pattern here that allows immediate negation of the control signals. If the EXEN bit is 0, the user-programmable machine 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 user-programmable machine. When the “branch” to the EXSA is performed, the user-programmable machine continues reading from successive locations until the LAST bit is equal to 1 in a UPM word.

15.2.3.7 ADDRESS CONTROL SIGNALS. You can control the address signals with the pattern written into the user-programmable machine. The AMA and AMB bits choose between outputting an address requested by the internal master or outputting it according to the multiplexing specified by the AMA and 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. The address for the first clock cycle of a read or write memory access is generated according to the value of the SAM bit in the corresponding option register.

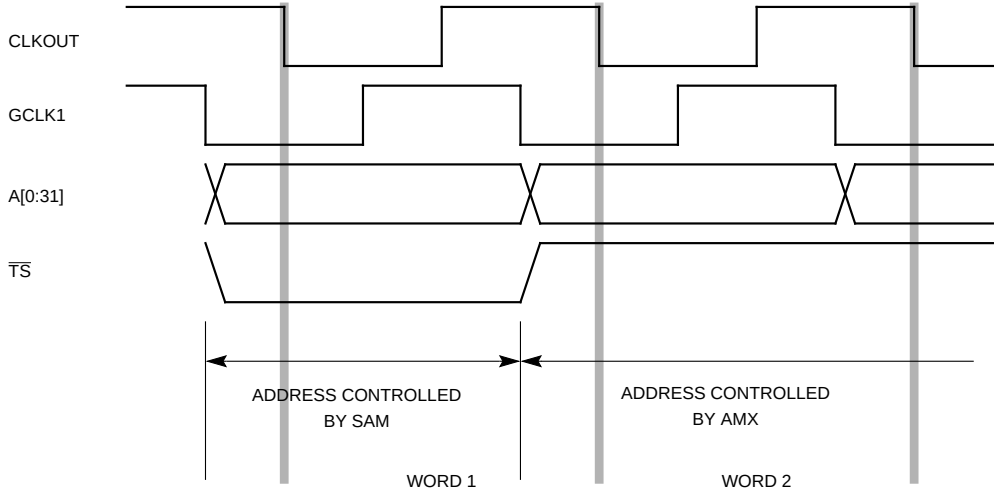


Table 15-5. Address Multiplexing

AMA/ AMB	A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31
000	RES	RES	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21	A22	A23
001	RES	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21	A22
010	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21
011	RES	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20
100	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19
101	RES	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18

Table 15-6 shows how the AMA and AMB bits can be defined to interface with a wide range of DRAM modules.

Table 15-6. AMA/AMB Definition For DRAM Interfaces

WIDTH	SIZE (K)	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 Interfaces (Continued)

WIDTH	SIZE (K)	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 Interfaces (Continued)

WIDTH	SIZE (K)	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

15.2.3.8 THE DISABLE TIMER MECHANISM. The disable timer associated with each user-programmable machine allows you 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 $\overline{RAS}$ 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 user-programmable machine, it is accommodated. To avoid conflicts between different banks accessing the same user-programmable machine, it is recommended that each pattern on the user-programmable machine be equal to or greater than the defined disable timer period.

15.2.3.9 TRANSFER ACKNOWLEDGE AND DATA SAMPLE CONTROL. During UPM memory access, the value of the $\overline{TA}$ signal driven by the memory controller and sampled by the external bus interface is indicated in the UTA bit of the user-programmable machine RAM word. The $\overline{TA}$ signal is driven at the rising edge of the GCLK2 signal. When a read access is handled by the user-programmable machine 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 MPC801. Figure 15-30 illustrates a schematic description of the hardware controlled by the user-programmable machine.

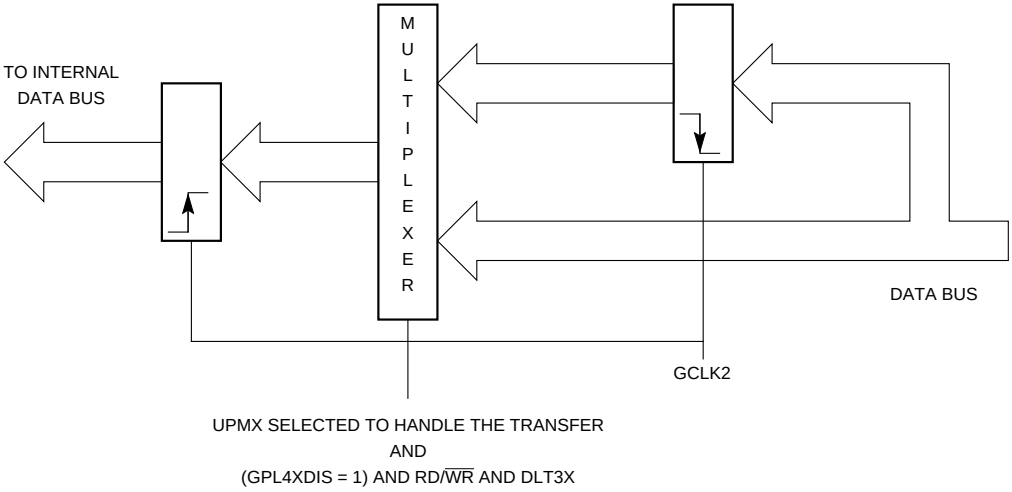


Figure 15-30. UPM Data Handling In Read Accesses

15.2.3.10 THE WAIT MECHANISM. The memory controller provides two mechanisms that it uses to interface with slave devices that are either slow or cannot guarantee a predefined access time—the Wait and the external TA. These devices can be divided into two main types:

- Variable access time devices
- Slow devices

The Wait mechanism is only used in accesses that are controlled by the user-programmable machine. The GPLA4DIS bit of the MAMR and the GPLB4DIS bit of the MBMR enable this mechanism. The external TA mechanism is only used in accesses that are controlled by the GPCM. The SETA bit in the option register specifies whether the TA is generated internally or externally.

15.2.3.10.1 Variable Access Time Solution. Assume that the core initiates a read cycle on the local bus that addresses the main storage connected to the system bus. The hierarchical bus interface accepts the local bus request and generates a read cycle on the system bus. The programmer cannot foresee when the data will be valid to be latched by the core since the system bus can be occupied by the DMA. There are two possible solutions:

- The external module signals to the memory controller that the data is not ready yet by asserting the UPWAIT signal. The memory controller synchronizes the signal since the wait signal is asynchronous. When the wait signal is asserted, the user-programmable machine enters freeze mode at the falling edge of the CLOCKOUT when the WAEN bit is set in the UPM word. The user-programmable machine remains in this state as long as the UPWAIT signal is asserted. After UPWAIT is negated, the user-programmable machine will continue executing from the next entry to the end of the pattern and the LAST bit is set.
- The bus interface module signals to the memory controller when it can sample the data by asserting the synchronous TA signal.

15.2.3.10.2 Slow Device Solution. Assume that the core initiates a read cycle from slow devices whose access time is greater than the maximum allowed by the user-programmable machine. There are two possible solutions:

- The core generates a read access from the slow device. The device will react by asserting a wait signal as long as the data is not ready. The core will sample the data only after the wait signal is negated.
- The core generates a read access from the slow device, which is responsible for generating the synchronous TZ when the data is ready.

15.2.3.10.3 Internal and External Synchronous Master. Figure 15-31 illustrates how the WAEN bit in the word is read by the user-programmable machine and the UPWAIT signal is used to hold the user-programmable machine in a particular state until the UPWAIT signal is negated.

The UPWAIT signal is sampled at the falling edge of the CLKOUT. If the signal is asserted and the WAEN bit in the current user-programmable machine is enabled word, the user-programmable machine is frozen until the UPWAIT signal is negated. The value of the external pins driven by the user-programmable machine remains as indicated in the word previously read by the user-programmable machine. When the UPWAIT signal is negated, the user-programmable machine continues with its normal functions. During the wait cycles, the $\overline{TA}$ signal is negated by the user-programmable machine.

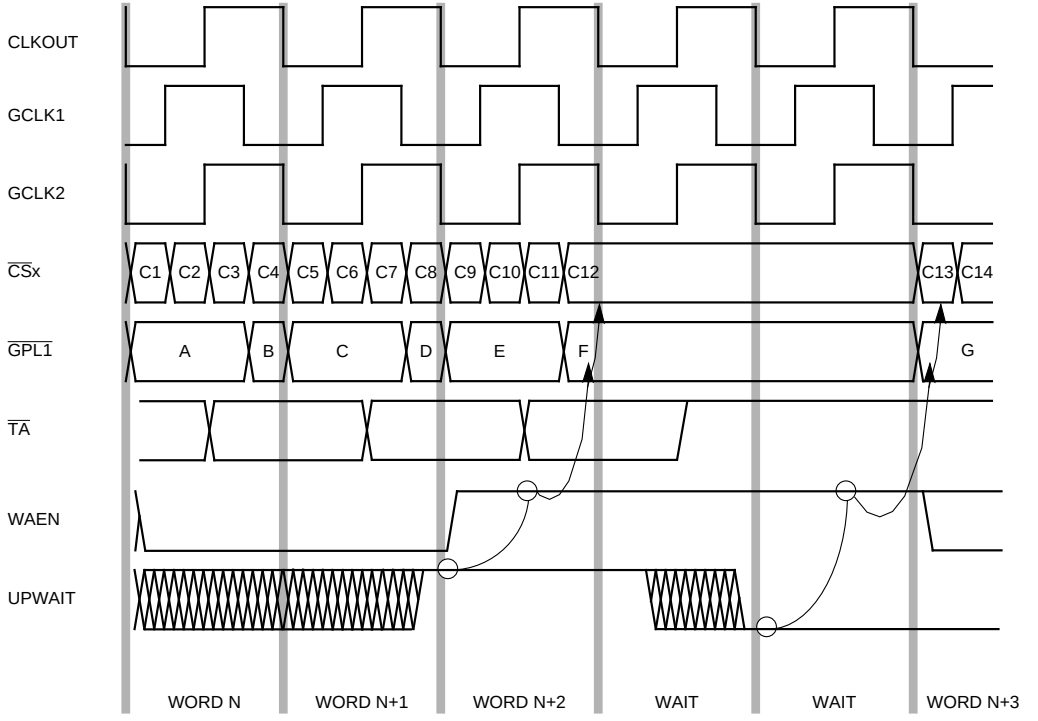


Figure 15-31. UPM Wait Mechanism Timing For Internal and External Synchronous Masters

15.2.3.10.4 External Asynchronous Master. When the user-programmable machine is activated to support an asynchronous external master, the wait mechanism operates in a way that allows the $\overline{AS}$ signal to behave as the external wait signal. The user-programmable machine enters a wait state if, after synchronizing it, the $\overline{AS}$ signal is 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 user-programmable machine remains as indicated in the previous word read by the user-programmable machine.

To exit the wait state, the $\overline{AS}$ signal should be negated, thus causing all external signals controlled that are by the user-programmable machine to be driven high a circuit delay after the negation. The external signals are driven in this state until the LAST bit is found in a UPM word. The $\overline{TA}$ signal that is driven by the user-programmable machine remains in its previous value until the $\overline{AS}$ signal is negated. The TODT bit is relevant in the words read by the user-programmable machine after $\overline{AS}$ is negated. Refer to **Section 15.3 External Master Support** for more information.

When the LAST bit is read in a word of the user-programmable machine RAM array, the highest priority pending request (if any) is serviced without a “gap cycle” in the external memory transactions dependent on the disable timer values.

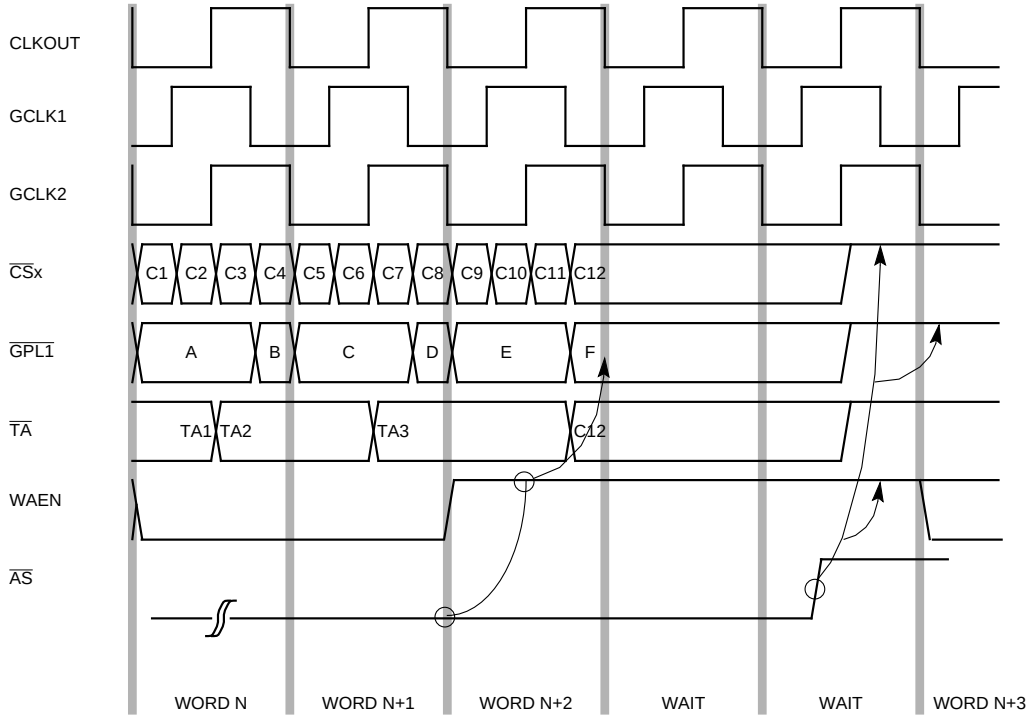


Figure 15-32. UPM Wait Mechanism Timing For An External Asynchronous Master

15.2.3.11 LOCATION OF UPM START ADDRESSES. Table 15-7 provides the starting addresses of the user-programmable machine RAM words for each transaction type.

Table 15-7. UPM Start Address Locations

TRANSACTION TYPE	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

15.2.3.12 EXAMPLE DRAM INTERFACE. Connecting the MPC801 to a DRAM device requires a detailed examination of the timing diagrams that represent the possible memory cycles the MPC801 must perform to access the device.

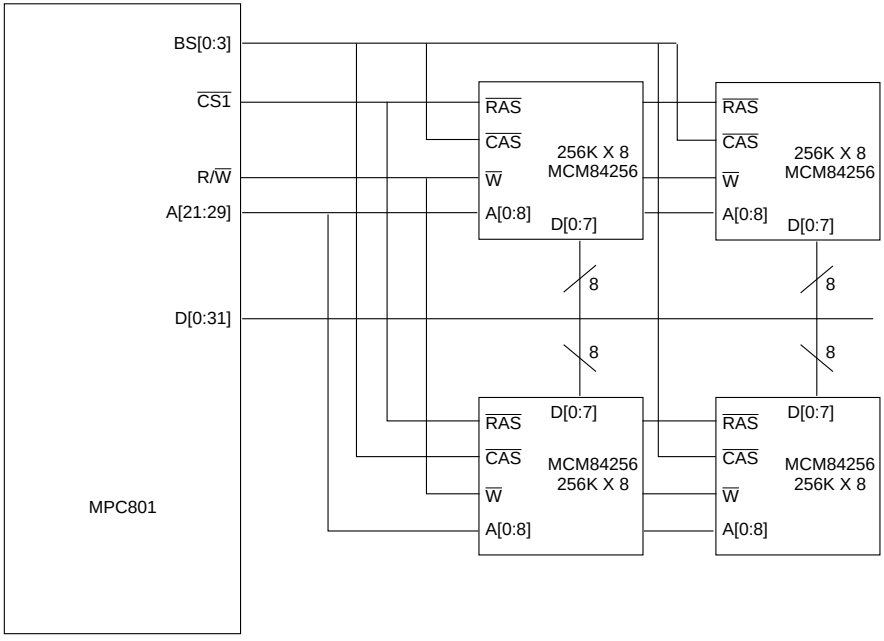


Figure 15-33. MPC801–DRAM Interface Connection

After the timing diagrams are created, the programming process translates the timings into tables that represent the RAM array contents for each possible cycle. When the tables are completed, the global parameters of the user-programmable machine must be defined for handling the disable timer (precharge) and periodic timer (refresh) relative to Figure 15-33. The following table shows the different field 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/$\overline{GPLA}$	0
SAM	1
$\overline{BI}$	0

The RAM array of the user-programmable machine can be written with the MCR. The option and base registers of the specific bank must be initialized according to the address mapping of the DRAM device being used. The MS field should indicate the user-programmable machine selected to handle the cycle. Figure 15-34 illustrates the first locations addressed by the user-programmable machine, according to the different services required by DRAM.

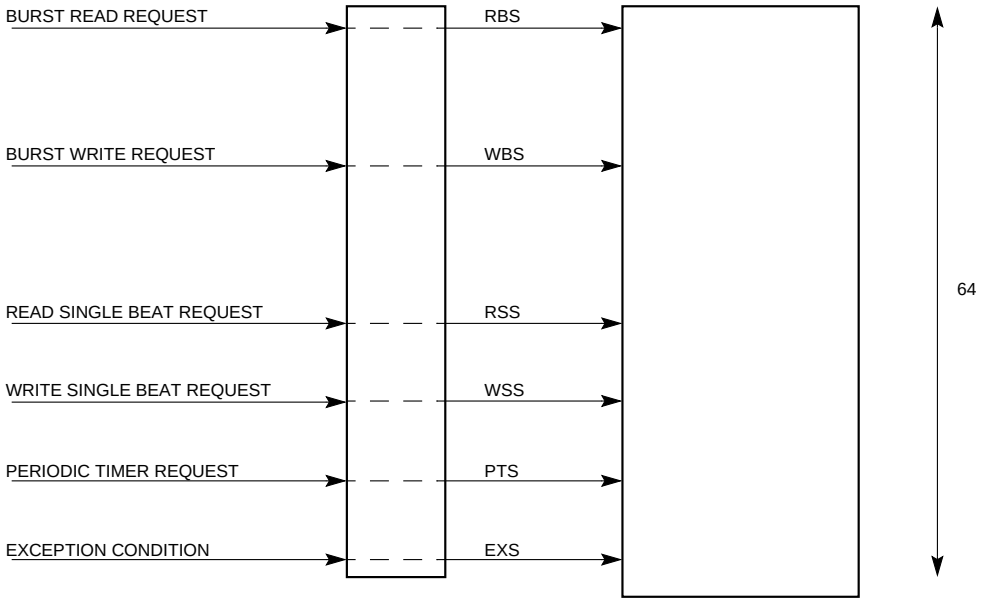
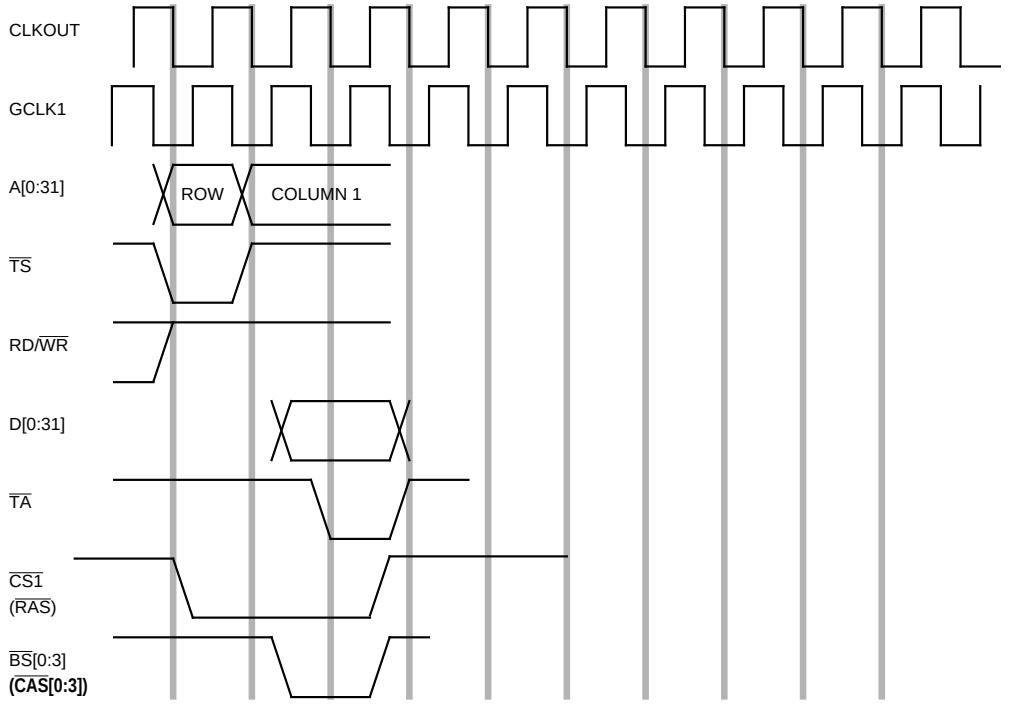


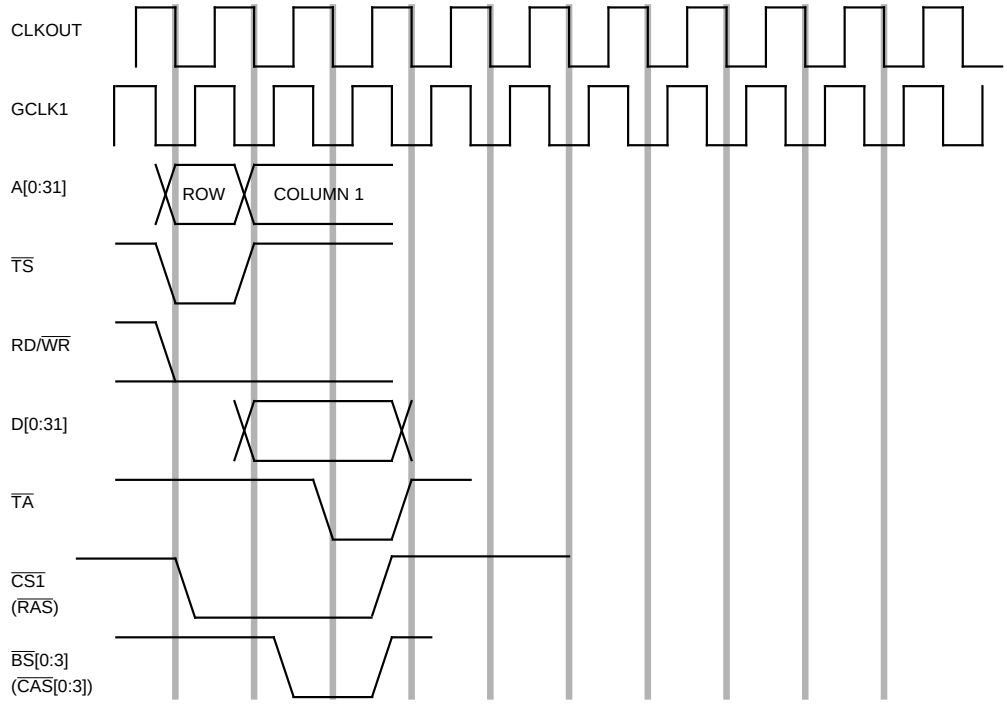
Figure 15-34. Address Start Pointers of the UPM RAM Array

In Figures 15-35 through 15-41, the table portion at the bottom of each figure represents the RAM array contents that handle each of the possible cycles. Each column represents a different word in the RAM array. The SAM bit in the option register determines address multiplexing for the first clock cycle and subsequent cycles are controlled by the user-programmable machine RAM words. Also notice that the AMX bits in the user-programmable machine RAM word control the address multiplexing for the following clock cycles rather than the current cycle.



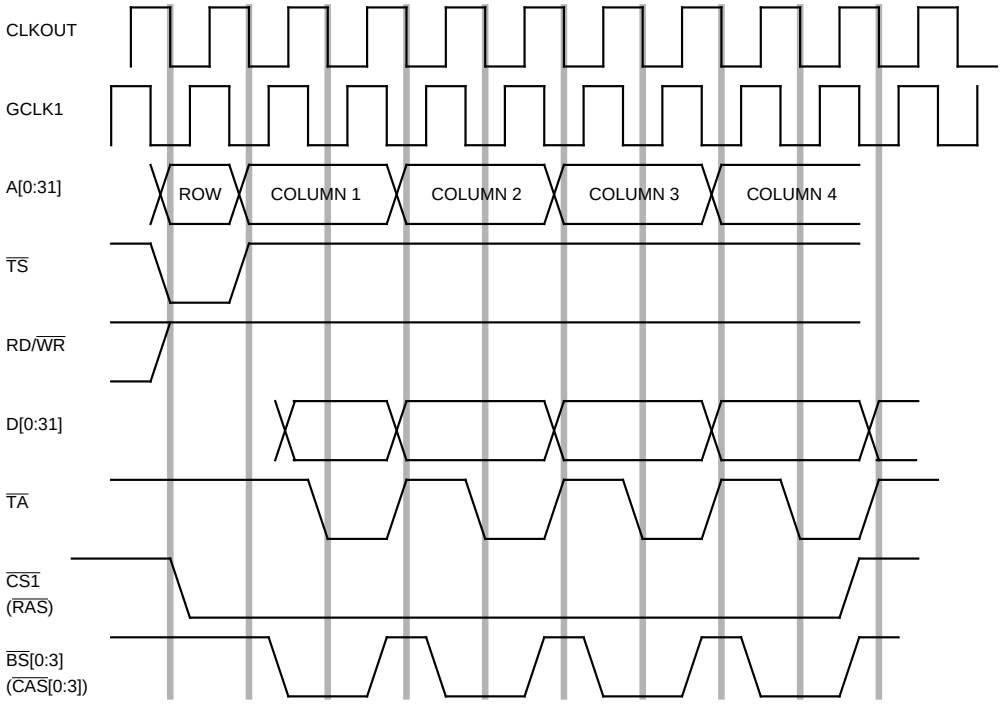
cst4	0	0	0		Bit 0
cst1	0	0	0		Bit 1
cst2	0	0	1		Bit 2
cst3	0	0	1		Bit 3
bst4	1	1	0		Bit 4
bst1	1	0	0		Bit 5
bst2	1	0	1		Bit 6
bst3	1	0	1		Bit 7
g0l0					Bit 8
g0l1					Bit 9
g0h0					Bit 10
g0h1					Bit 11
g1t4					Bit 12
g1t3					Bit 13
g2t4					Bit 14
g2t3					Bit 15
g3t4					Bit 16
g3t3					Bit 17
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



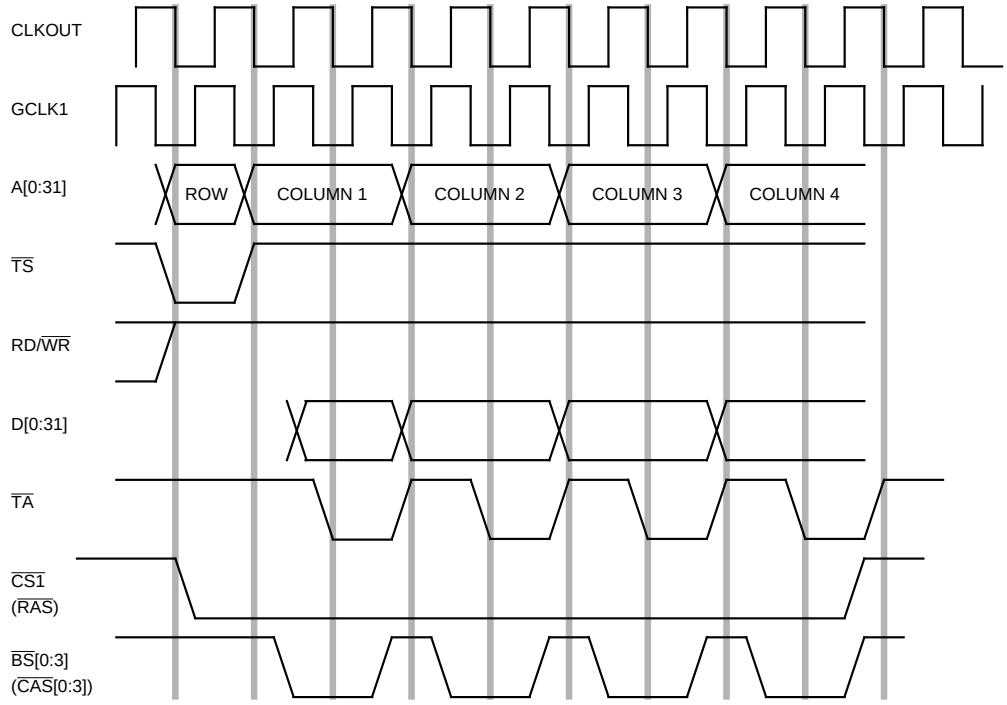
cst4	0	0	0		Bit 0
cst1	0	0	0		Bit 1
cst2	0	0	1		Bit 2
cst3	0	0	1		Bit 3
bst4	1	1	0		Bit 4
bst1	1	0	0		Bit 5
bst2	1	0	1		Bit 6
bst3	1	0	1		Bit 7
g0l0					Bit 8
g0l1					Bit 9
g0h0					Bit 10
g0h1					Bit 11
g1t4					Bit 12
g1t3					Bit 13
g2t4					Bit 14
g2t3					Bit 15
g3t4					Bit 16
g3t3					Bit 17
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



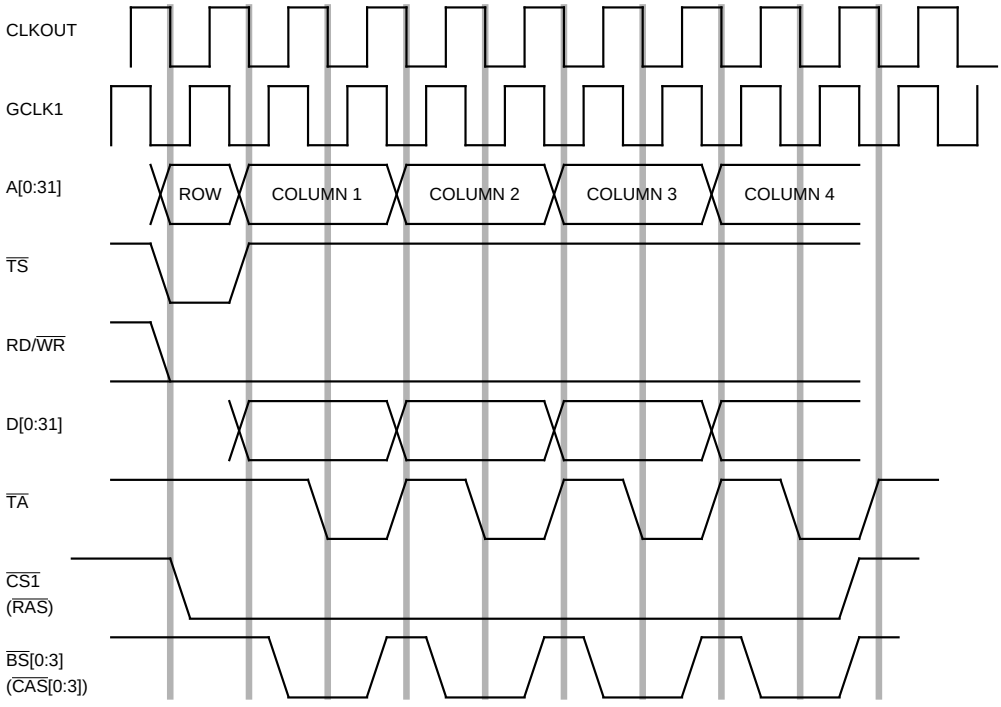
cst4	0	0	0	0	0	0	0	0	0		Bit 0
cst1	0	0	0	0	0	0	0	0	0		Bit 1
cst2	0	0	0	0	0	0	0	0	1		Bit 2
cst3	0	0	0	0	0	0	0	0	1		Bit 3
bst4	1	1	0	1	0	1	0	1	0		Bit 4
bst1	1	0	0	0	0	0	0	0	0		Bit 5
bst2	1	0	1	0	1	0	1	0	1		Bit 6
bst3	1	0	1	0	1	0	1	0	1		Bit 7
g0l0											Bit 8
g0l1											Bit 9
g0h0											Bit 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)



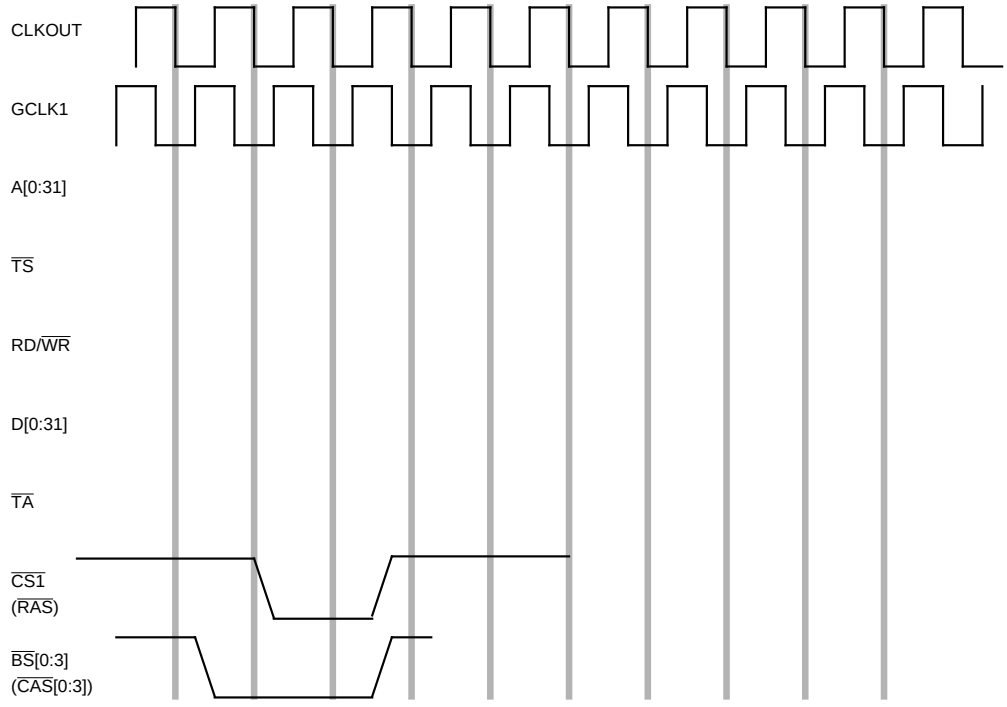
cst4	0	0	0	0	0		Bit 0
cst1	0	0	0	0	0		Bit 1
cst2	0	0	0	0	1		Bit 2
cst3	0	0	0	0	1		Bit 3
bst4	1	1	0	1	0		Bit 4
bst1	1	0	0	0	0		Bit 5
bst2	1	0	1	0	1		Bit 6
bst3	1	0	1	0	1		Bit 7
g0l0							Bit 8
g0l1							Bit 9
g0h0							Bit 10
g0h1							Bit 11
g1t4							Bit 12
g1t3							Bit 13
g2t4							Bit 14
g2t3							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)



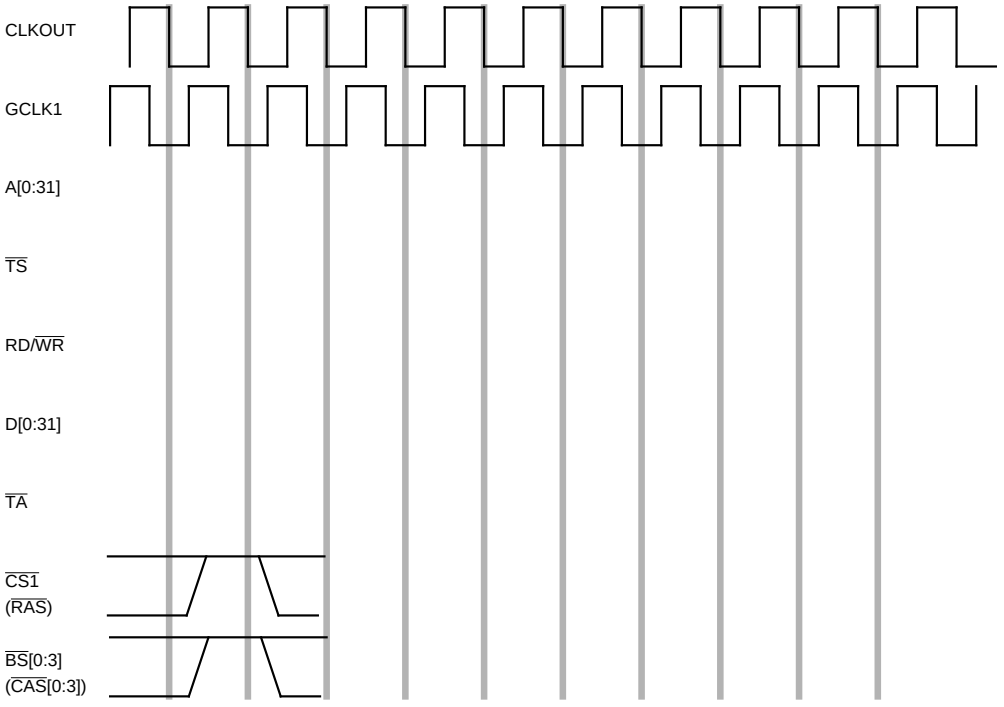
cst4	0	0	0	0	0	0	0	0	0		Bit 0
cst1	0	0	0	0	0	0	0	0	0		Bit 1
cst2	0	0	0	0	0	0	0	0	1		Bit 2
cst3	0	0	0	0	0	0	0	0	1		Bit 3
bst4	1	1	0	1	0	1	0	1	0		Bit 4
bst1	1	0	0	0	0	0	0	0	0		Bit 5
bst2	1	0	1	0	1	0	1	0	1		Bit 6
bst3	1	0	1	0	1	0	1	0	1		Bit 7
g0l0											Bit 8
g0l1											Bit 9
g0h0											Bit 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 cst1 cst2 cst3	1 1 1 1		Bit 0 Bit 1 Bit 2 Bit 3
bst4 bst1 bst2 bst3	1 1 1 1		Bit 4 Bit 5 Bit 6 Bit 7
g0l0 g0l1 g0h0 g0h1			Bit 8 Bit 9 Bit 10 Bit 11
g1t4 g1t3			Bit 12 Bit 13
g2t4 g2t3			Bit 14 Bit 15
g3t4 g3t3			Bit 16 Bit 17
g4t4 g4t3			Bit 18 Bit 19
g5t4 g5t3			Bit 20 Bit 21
- -			Bit 22 Bit 23
loop exen	0 0		Bit 24 Bit 25
amx0 amx1 na uta todt last	0 0 0 0 1 1		Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31
	EXS		

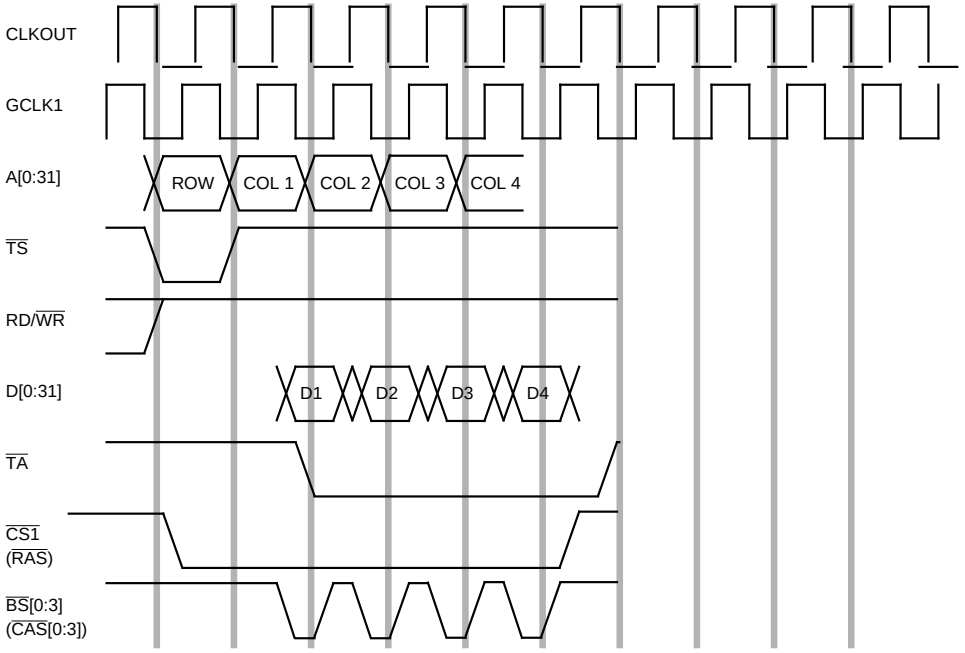
Figure 15-41. Exception Cycle

If the $\overline{\text{GPLA4}}$ signal 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”. The data bus is sampled at the falling edge of GCLK1 if instructed by the UPM word. The following table shows an example of how the burst read access to page mode DRAM (no LOOP) can be modified. 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)

15.2.3.13 EXTENDED DATA-OUT INTERFACE EXAMPLE

Figure 15-43 illustrates a memory connection to extended data-out (EDO) types of devices. For this connection, GPL1 is connected to the memory device OE pins.

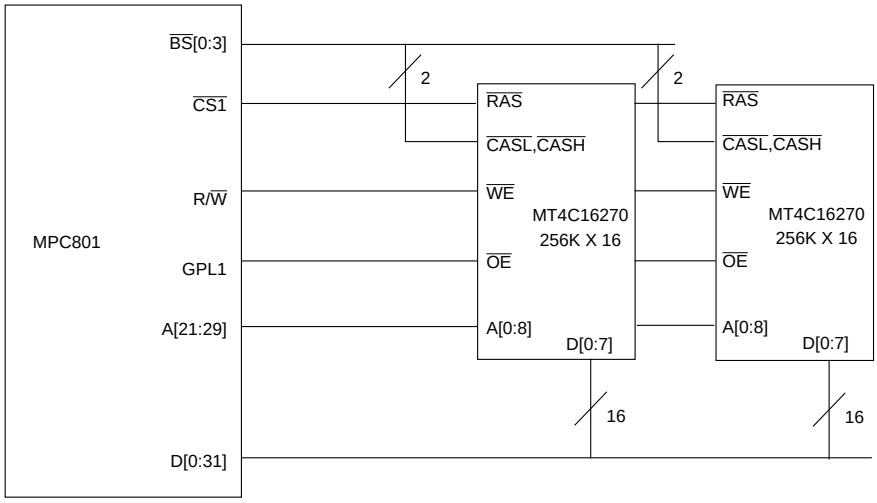
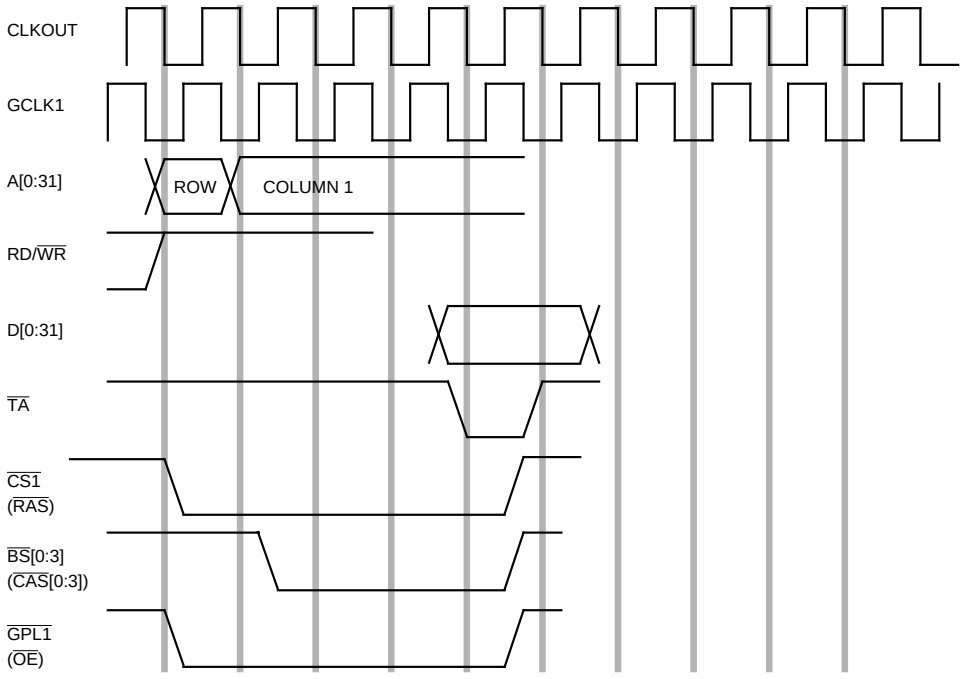


Figure 15-43. EDO Interface Connection

Table 15-10 shows the register field programming that supports the configuration illustrated in Figure 15-43. The assumption is that the BRGCLK frequency is 25MHz and that the device needs a 512-cycle refresh every 8 milliseconds. The example assumes a CLKOUT frequency of 50MHz.

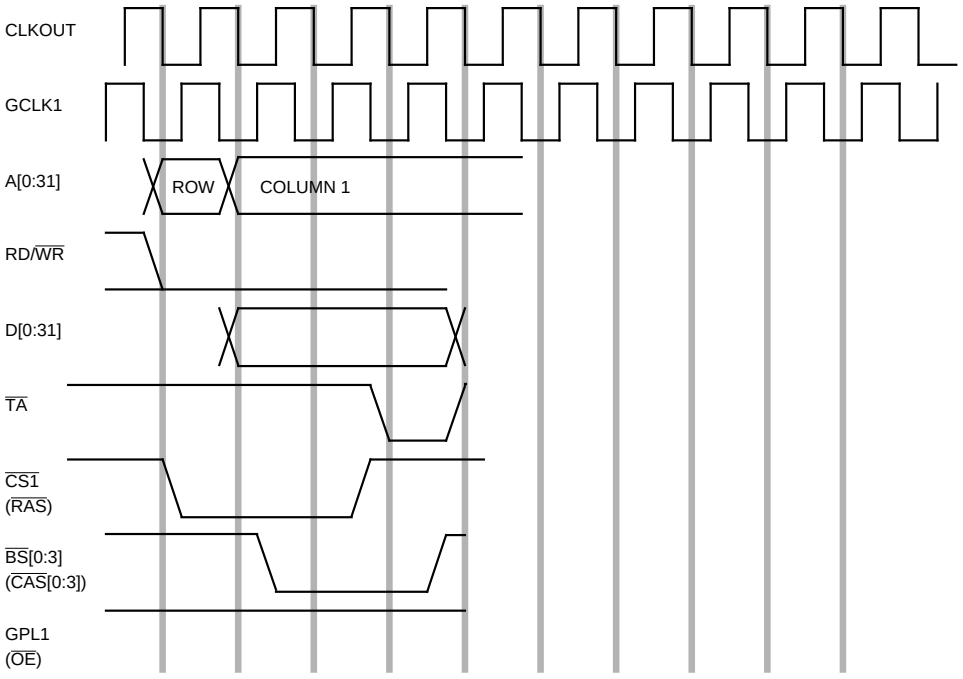
Table 15-10. EDO Connection Field Value Example

FIELD	VALUE
MS	10
PS	00
WP	0
PTP	x'02
PTA	x'0C
PTAE	1
AMA	001
DSA	10
SAM	1
BI	0



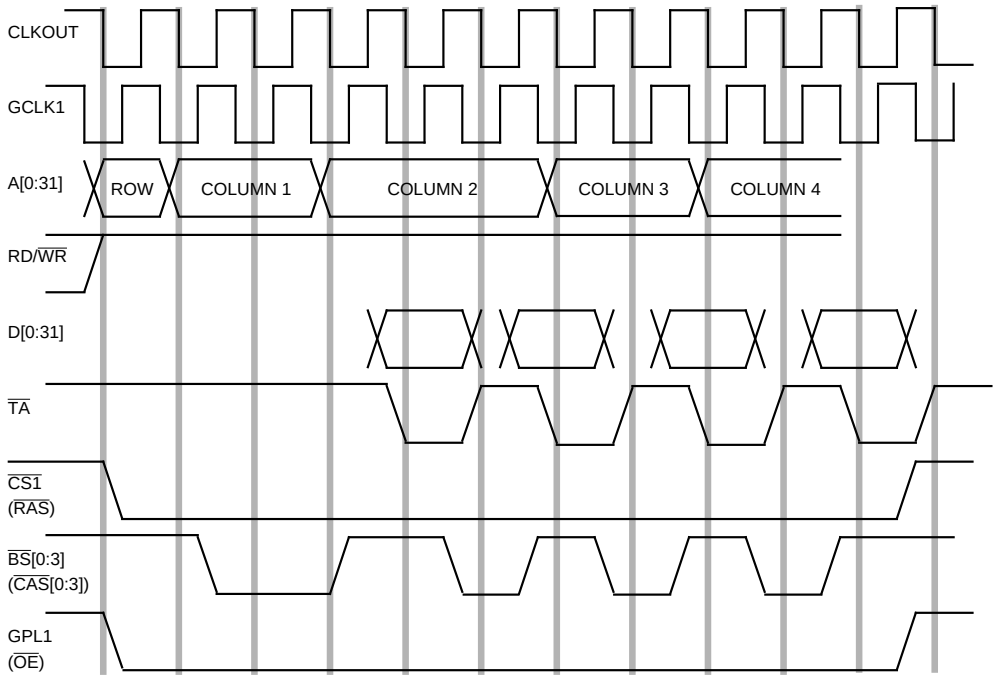
cst4	0	0	0	0	0		Bit 0
cst1	0	0	0	0	0		Bit 1
cst2	0	0	0	0	1		Bit 2
cst3	0	0	0	0	1		Bit 3
bst4	1	1	0	0	0		Bit 4
bst1	1	0	0	0	0		Bit 5
bst2	1	0	0	0	1		Bit 6
bst3	1	0	0	0	1		Bit 7
g0l0							Bit 8
g0l1							Bit 9
g0h0							Bit 10
g0h1							Bit 11
g1t4	0	0	0	0	0		Bit 12
g1t3	0	0	0	0	1		Bit 13
g2t4							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



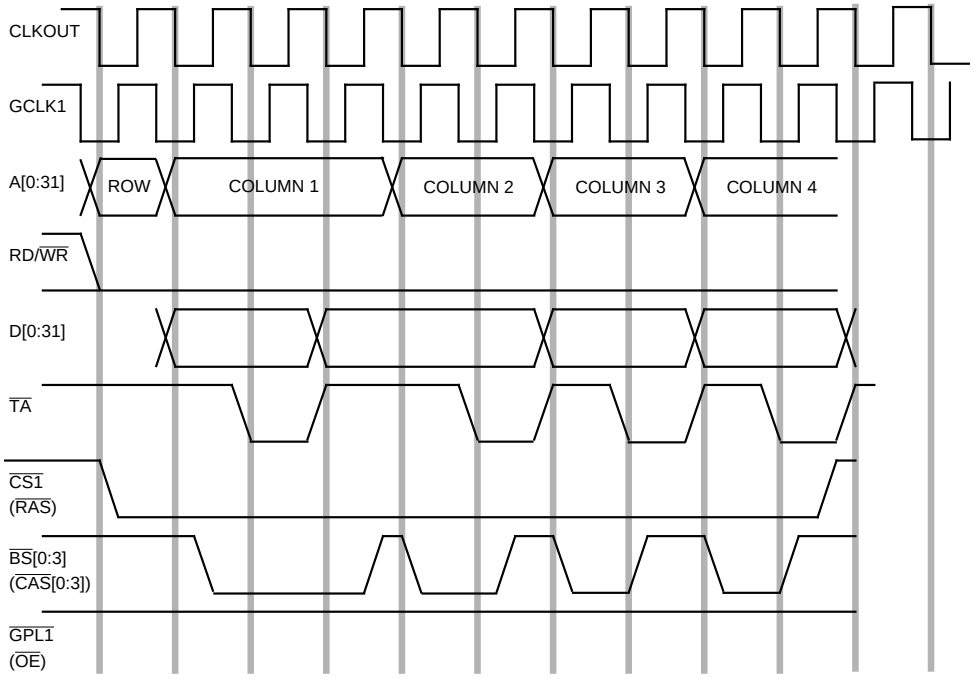
cst4	0	0	0	1		Bit 0
cst1	0	0	0	1		Bit 1
cst2	0	0	1	1		Bit 2
cst3	0	0	1	1		Bit 3
bst4	1	1	0	0		Bit 4
bst1	1	0	0	0		Bit 5
bst2	1	0	0	0		Bit 6
bst3	1	0	0	1		Bit 7
g0l0						Bit 8
g0l1						Bit 9
g0h0						Bit 10
g0h1						Bit 11
g1t4	1	1	1	1		Bit 12
g1t3	1	1	1	1		Bit 13
g2t4						Bit 14
g2t3						Bit 15
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		Bit 24
exen	0	0	0	0		Bit 25
amx0	0	0	0	1		Bit 26
amx1	0	0	0	0		Bit 27
na	0	0	0	0		Bit 28
uta	1	1	0	1		Bit 29
todt	0	0	0	1		Bit 30
last	0	0	0	1		Bit 31
	WSS	**WSS+1**	**WSS+2**	**WSS+3**		

Figure 15-45. Single Beat Write Access To Page Mode DRAM With Extended Data-Out



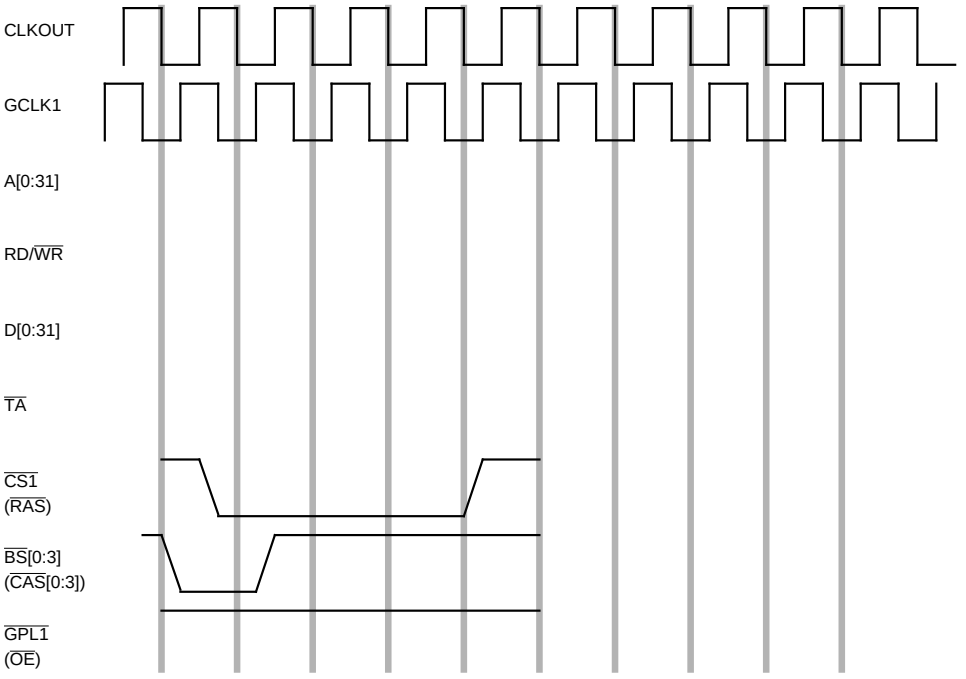
cst4	0	0	0	0	0	0	0	0	0	0	0	Bit 0
cst1	0	0	0	0	0	0	0	0	0	0	0	Bit 1
cst2	0	0	0	0	0	0	0	0	0	0	1	Bit 2
cst3	0	0	0	0	0	0	0	0	0	0	1	Bit 3
bst4	1	1	0	1	1	0	1	0	1	0	1	Bit 4
bst1	1	0	0	1	1	0	1	0	1	0	1	Bit 5
bst2	1	0	0	1	0	1	0	1	0	1	1	Bit 6
bst3	1	0	0	1	0	1	0	1	0	1	1	Bit 7
g0l0												Bit 8
g0l1												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



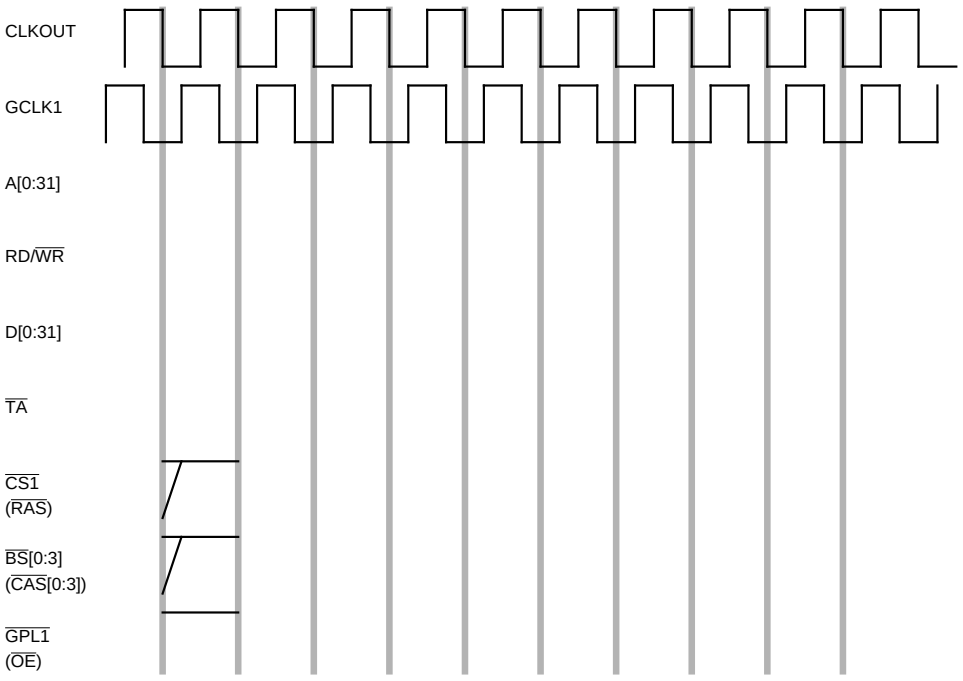
cst4	0	0	0	0	0	0	0	0	0	0		Bit 0
cst1	0	0	0	0	0	0	0	0	0	0		Bit 1
cst2	0	0	0	0	0	0	0	0	0	1		Bit 2
cst3	0	0	0	0	0	0	0	0	0	1		Bit 3
bst4	1	1	0	0	0	0	0	1	0	1		Bit 4
bst1	1	0	0	0	0	1	0	1	0	1		Bit 5
bst2	1	0	0	1	0	1	0	1	0	1		Bit 6
bst3	1	0	0	1	0	1	0	1	0	1		Bit 7
g0l0												Bit 8
g0l1												Bit 9
g0h0												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 cst1 cst2 cst3	1 1 1 1		Bit 0 Bit 1 Bit 2 Bit 3
bst4 bst1 bst2 bst3	1 1 1 1		Bit 4 Bit 5 Bit 6 Bit 7
g0l0 g0l1 g0h0 g0h1			Bit 8 Bit 9 Bit 10 Bit 11
g1t4 g1t3	1 1		Bit 12 Bit 13
g2t4 g2t3			Bit 14 Bit 15
g3t4 g3t3			Bit 16 Bit 17
g4t4 g4t3			Bit 18 Bit 19
g5t4 g5t3			Bit 20 Bit 21
- -			Bit 22 Bit 23
loop exen	0 0		Bit 24 Bit 25
amx0 amx1 na uta todt last	0 0 0 1 1 1		Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31
	EXS		

Figure 15-49. Exception Cycle For Page Mode DRAM With Extended Data-Out

15.3 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 12.12.1.1 SIU Module Configuration Register** for more information. The external bus masters are classified into two types:

- Synchronous—Bus masters that work with CLKOUT and the MPC801 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 the $\overline{TS}$ signal. The address bus A[0:31] must be stable throughout the transaction, starting at the rising edge of CLKOUT in which $\overline{TS}$ is asserted until the last $\overline{TA}$ acknowledges the transfer. Since the external master works synchronously with the MPC801, only setup and hold times near the rising edge of CLKOUT are important. Assuming the SEME bit in the SIU module configuration register is set and the $\overline{TS}$ signal is asserted, the memory controller compares the address with each one of its defined valid banks. If a match is found, control signals to the memory devices are generated and the $\overline{TA}$ signal 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 $\overline{AS}$ signal. The A[0:31] signals, together with the RD/$\overline{WR}$ and TSIZE[0:1] signals, must be stable at setup time before the $\overline{AS}$ pin is asserted. If the AEME bit in the SIU module configuration register is set, the memory controller in the MPC801 synchronizes the $\overline{AS}$ assertion to its internal clock and generates the control line to the external memory devices. A $\overline{TA}$ signal is given to the external master to acknowledge the transaction. All the control signals to the memory device and the $\overline{TA}$ signal are negated with the $\overline{AS}$ pin.

NOTE

When external masters access slaves on the bus, the internal AT[0:2] signals reaching the memory controller will be forced to `100'.

The BADDR[28:30] pins should be used to generate addresses to memory devices during burst accesses. They duplicate the value of the A[28:30] signals 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] signals reflect the value of the A[28:30] signals on the first memory access clock cycle. Afterwards, they behave according to the memory controller.

To connect to external memory devices that require address multiplexing, the MPC801 can use the $\overline{\text{GPL5A}}$ and $\overline{\text{GPL5B}}$ signals to control external multiplexing logic. The $\overline{\text{GPL5}}x$ signal logic value is changed if you specify when any of the user-programmable machines in the memory controller control the slave access. The $\overline{\text{GPL5}}x$ signal reflects the value of the GL5S bit in the corresponding option register in the first clock cycle of the memory device access. In the following cycles, the value is determined by the G5T4 and G5T3 bits in the user-programmable machine's RAM. If UPMB controls the slave access, the GL5A bit of the option register indicates that the value of GL5, G5T4 and G5T3 in the UPMB controls the logical value of the $\overline{\text{GPL5A}}$ signal. GL5S is only considered for read or write accesses to memory and not for patterns initiated by the user-programmable machine as a result of an internal periodic timer request or software request.

Table 15-11. $\overline{\text{GPL5}}$ Signal Behavior

MACHINE CONTROLLING MEMORY ACCESS	MEMORY ACCESS CLOCK CYCLE	G5LA	G5LS	G5T4	G5T3	$\overline{\text{GPL5}}x$
GPCM	x	x	x	x	x	$\overline{\text{GPL5A}}$ and $\overline{\text{GPL5B}}$ do not change their value.
UPMA	First	x	0	x	x	$\overline{\text{GPL5A}}$ is driven low at the falling edge of GCLK1.
			1			$\overline{\text{GPL5A}}$ is driven high at the falling edge of GCLK1.
	Second, Third...	x	x	0	x	$\overline{\text{GPL5A}}$ is driven low at the falling edge of GCLK2 in the current UPM cycle.
				1	x	$\overline{\text{GPL5A}}$ is driven high at the falling edge of GCLK2 in the current UPM cycle.
				x	0	$\overline{\text{GPL5A}}$ is driven low at the falling edge of GCLK1 in the current UPM cycle.
				x	1	$\overline{\text{GPL5A}}$ is driven high at the falling edge of GCLK1 in the current UPM cycle.
UPMB	First	0	0	x	x	$\overline{\text{GPL5B}}$ is driven low at the falling edge of GCLK1.
			1			$\overline{\text{GPL5B}}$ is driven high at the falling edge of GCLK1.
		1	0	x	x	$\overline{\text{GPL5A}}$ is driven low at the falling edge of GCLK1.
			1			$\overline{\text{GPL5A}}$ is driven high at the falling edge of GCLK1.
	Second, Third...	0	0	0	x	$\overline{\text{GPL5B}}$ is driven low at the falling edge of GCLK2 in the current UPM cycle.
				1	x	$\overline{\text{GPL5B}}$ is driven high at the falling edge of GCLK2 in the current UPM cycle.
				x	0	$\overline{\text{GPL5B}}$ is driven low at the falling edge of GCLK1 in the current UPM cycle.
				x	1	$\overline{\text{GPL5B}}$ is driven high at the falling edge of GCLK1 in the current UPM cycle.

Table 15-11. $\overline{\text{GPL5}}$ Signal Behavior

MACHINE CONTROLLING MEMORY ACCESS	MEMORY ACCESS CLOCK CYCLE	G5LA	G5LS	G5T4	G5T3	$\overline{\text{GPL5}}x$
UPMB	Second, Third...	1	x	0	x	$\overline{\text{GPL5A}}$ is driven low at the falling edge of GCLK2 in the current UPM cycle.
				1	x	$\overline{\text{GPL5A}}$ is driven high at the falling edge of GCLK2 in the current UPM cycle.
				x	0	$\overline{\text{GPL5A}}$ is driven low at the falling edge of GCLK1 in the current UPM cycle.
				x	1	$\overline{\text{GPL5A}}$ is driven high at the falling edge of GCLK1 in the current UPM cycle.

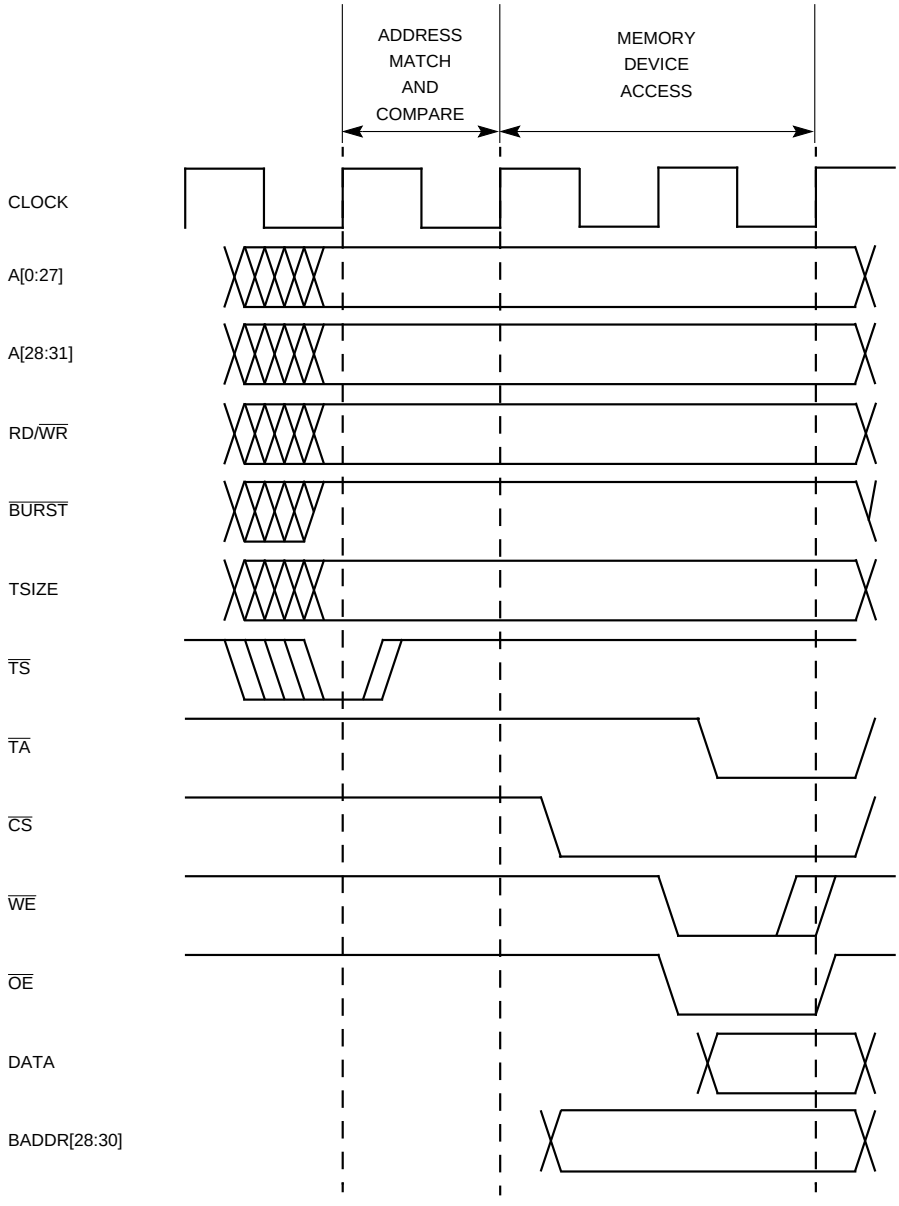


Figure 15-50. Synchronous External Master Basic Access (GPCM Controlled)

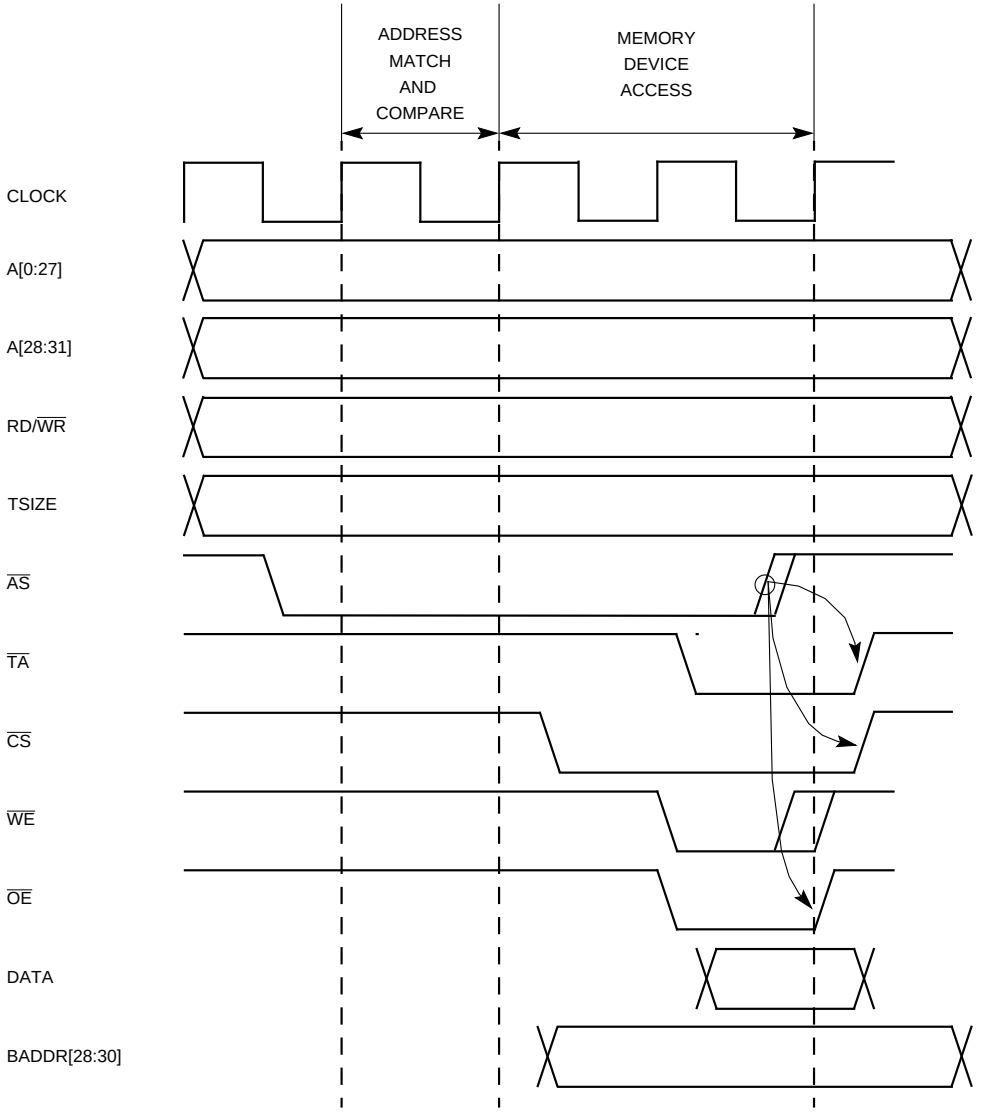


Figure 15-51. Asynchronous External Master Basic Access (GPCM Controlled)

Figure 15-52 illustrates a typical system configuration in which the MPC801 and an external master accesses a DRAM device. Figure 15-53 illustrates the timing behavior of the GPL5, BADDR, and the other control signals for a burst read access initiated by an external master to a DRAM device. The value of the 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.

15

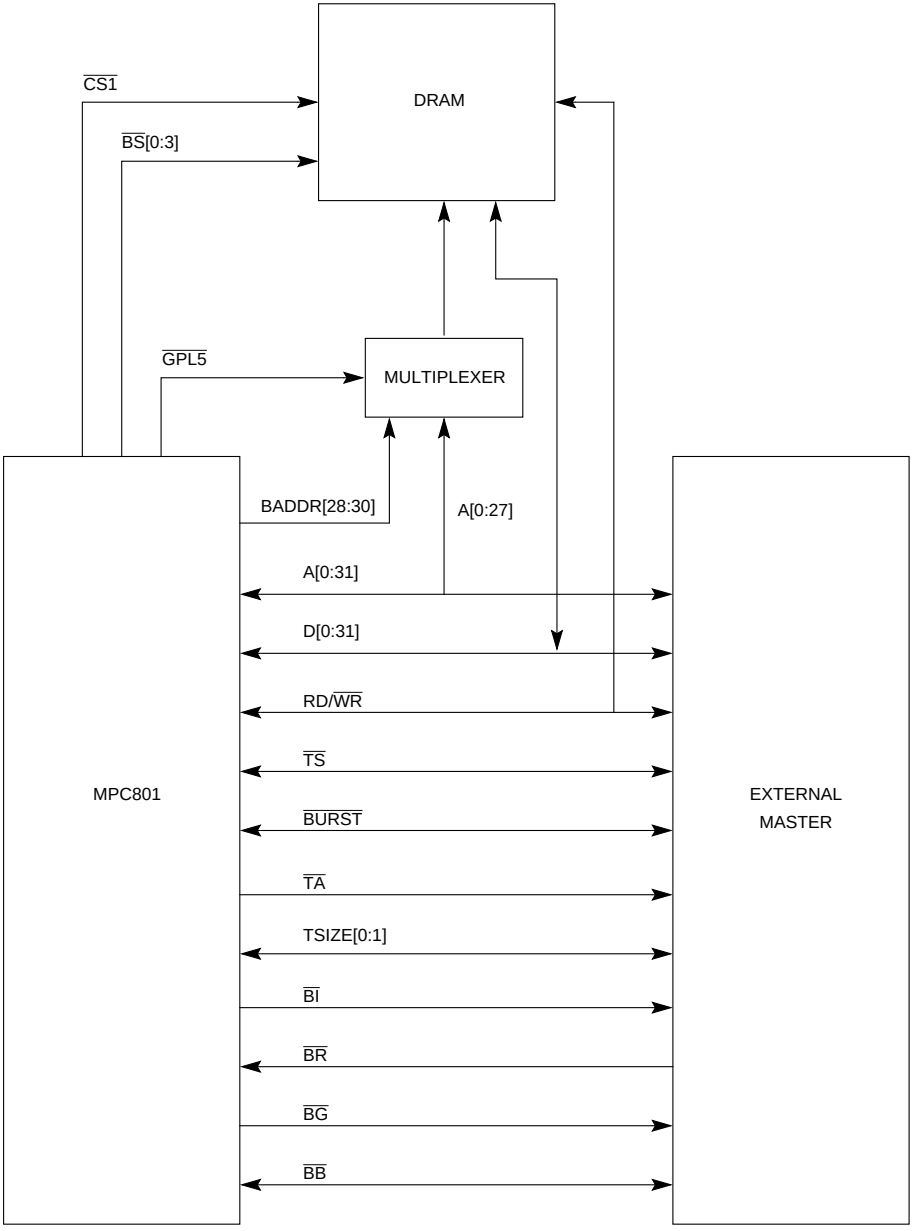
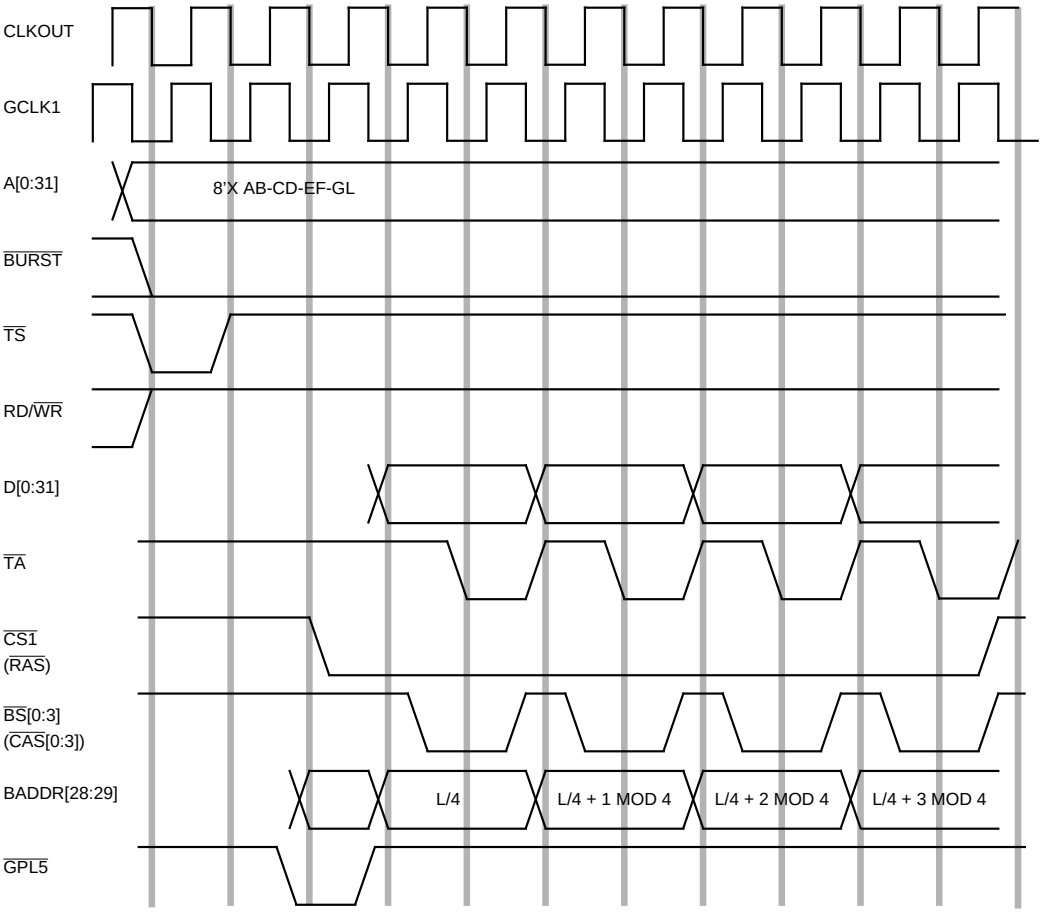


Figure 15-52. Synchronous External Master–MPC801–DRAM Device Typical Configuration



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

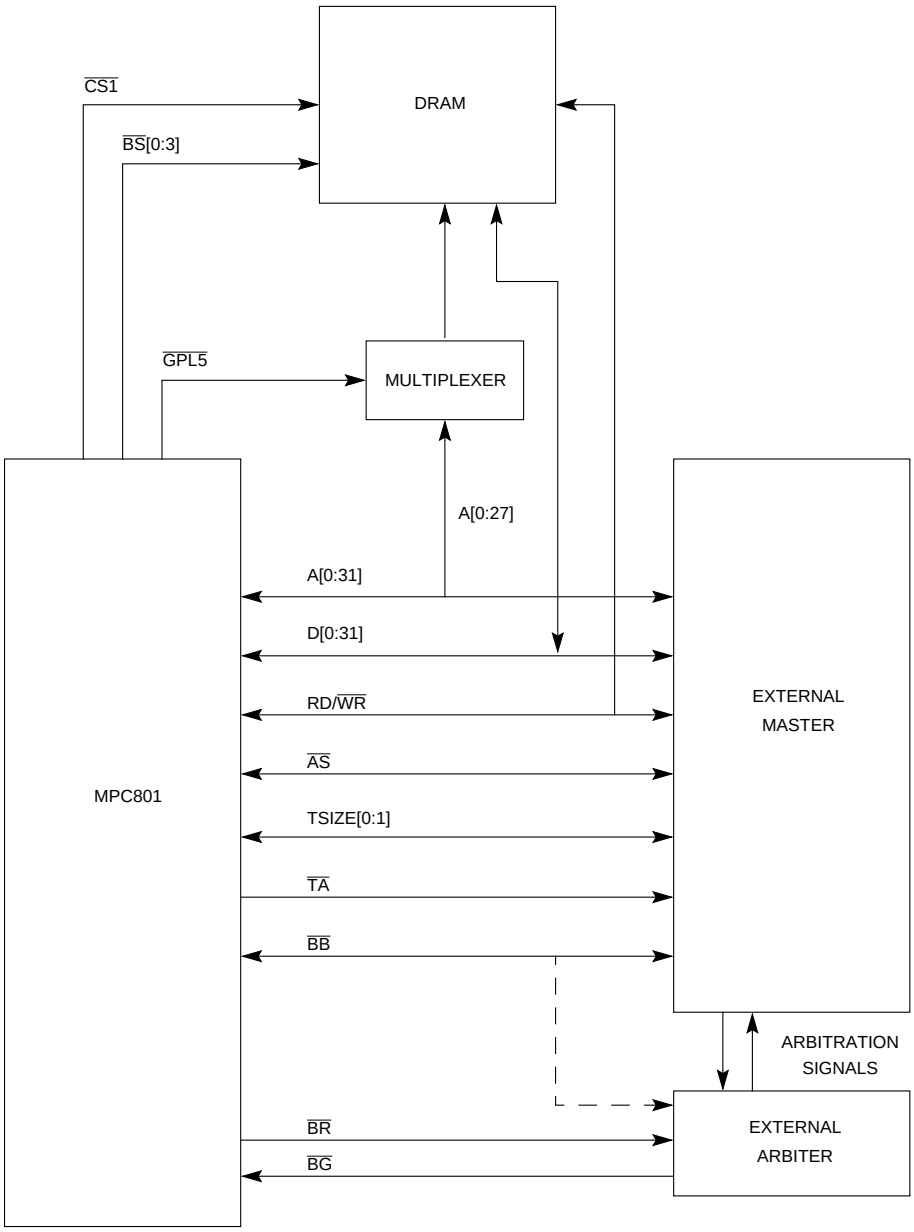
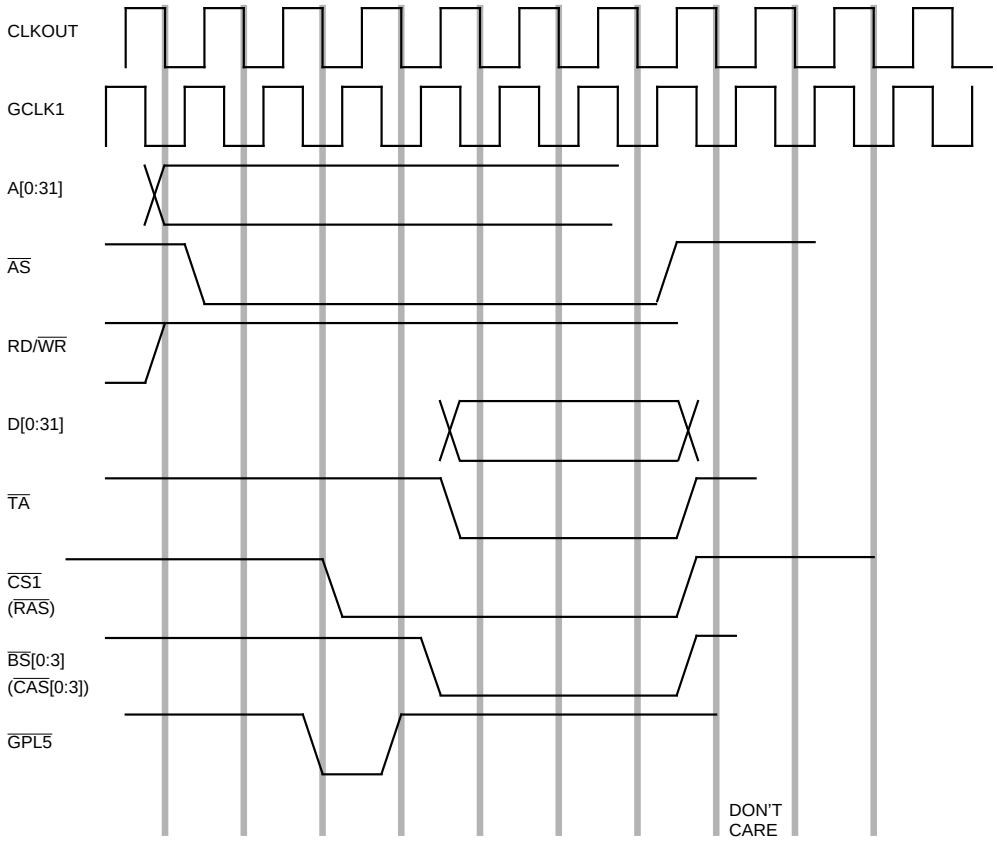


Figure 15-54. Asynchronous External Master–MPC801–DRAM Device Typical Configuration

15



cst4			0	0	0	0	0			Bit 0
cst1			0	0						Bit 1
cst2			0	0						Bit 2
cst3			0	0						Bit 3
bst4			1	1	0	0	0			Bit 4
bst1			1	0						Bit 5
bst2			1	0						Bit 6
bst3			1	0						Bit 7
≈										
g4t4										Bit 18
g4t3										Bit 19
g5t4					1	1	1			Bit 20
g5t3			0	1						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	0			Bit 26
amx1			0	0	0	0	0			Bit 27
na			0	0	0	0	0			Bit 28
uta			1	0	0	0	0			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

15.4 PROGRAMMING THE MEMORY CONTROLLER

You can interface with the memory controller by using eight identical sets of two registers—the option register and the base register. There are also two identical registers (MAMR and MBMR) that control the user-programmable A and B machines. There are also some general registers.

15.4.1 Memory Status Register

The memory status register (MSR) is used to report parity or write-protect errors that are found when accessing the external bus.

MSR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PER0	PER1	PER2	PER3	PER4	PER5	PER6	PER7	WPER	RESERVED						

PER0—Parity Error Bank 0

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.

PER1—Parity Error Bank 1

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.

PER2—Parity Error Bank 2

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.

PER3—Parity Error Bank 3

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.

PER4—Parity Error Bank 4

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.

PER5—Parity Error Bank 5

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.

PER6—Parity Error Bank 6

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.

PER7—Parity Error Bank 7

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.

WPER—Write-Protection Error

This bit is asserted when a write-protect error occurs. A bus monitor, if enabled, will prompt you to read this register if no $\overline{TA}$ signal 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.

Bits 9–15—Reserved

These bits are reserved and should be set to 0.

15.4.2 Memory Periodic Timer Prescaler Register

The memory periodic timer prescaler register (MPTPR) determines the BRG prescaler value output of the prescaler used as a memory periodic timer input clock.

MPTPR

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

PTP—Periodic Timers Prescaler

This attribute determines the period of the memory periodic timers input clock. The BRGCLK clock is divided according to the encoding of these bits.

001x xxxx = Divide by 2.
0001 xxxx = Divide by 4.
0000 1xxx = Divide by 8.
0000 01xx = Divide by 16.
0000 001x = Divide by 32.
0000 0001 = Divide by 64.
1xxx xxxx = Reserved.
01xx xxxx = Reserved.

Bits 8–15—Reserved

These bits are reserved and should be set to 0.

15.4.3 Base Register

The base register (BR) contains the base address and address types that should be compared to the address bus. It also includes a memory attribute and selects the machine for memory operation handling.

BR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	BA															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	BA	AT			PS		PARE	WP	MS		RESERVED					V

BA—Base Address

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 of the option register.

AT—Address Type

This field can be used to limit accesses to the memory bank to a certain address space type. These bits are used in conjunction with the ATM[0:2] bits of the option register.

PS—Port Size

This field specifies the port size of the memory region.

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

PARE—Parity Enable

This bit is used to enable parity checking on this bank.

0 = Parity checking disable.
1 = Parity checking enable.

WP—Write-Protect

This bit can restrict write accesses within the address range of a base register. If you try to write to the range of addresses specified in a base address register that has this bit set, the $\overline{\text{TEA}}$ signal can be asserted by the bus monitor logic and cause this cycle to be terminated.

0 = Both read and write accesses are allowed.
1 = Only read accesses are allowed. The $\overline{\text{CS}}x$ and $\overline{\text{TA}}$ signals are not asserted by the memory controller on write cycles to this memory bank. WPER is set in the MSTAT register if you try to write to this memory bank.

MS—Machine Select

This field specifies the machine that is selected for memory operations handling.

00 = GPCM.
0X = Reserved.
10 = UPMA.
11 = UPMB.

Bits 26–30—Reserved

These bits are reserved and should be set to 0.

V—Valid Bit

This bit indicates that the contents of the base and option registers are valid. The $\overline{\text{CS}}$ signal does not assert until V is set. An access to a region that does not have V set can causes a bus monitor timeout. Following a system reset, this bit is set in BR0.

0 = This bank is invalid.
1 = This bank is valid.

15.4.4 Option Register

The option register (OR) contains the address and address type mask bit for address bus comparison. It also includes the CS general bits and all the GPCM parameters.

OR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	AM															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	AM	ATM			CSNT /SAM	ACS/ G5LA,G5LS		BI	SCY				SETA	TRLX	EHTR	RES

AM—Address Mask

These read/write bits 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 and 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, thus allowing a resource to reside in more than one area of the address map. Be aware that following a system reset, these bits are reset in OR0.

ATM—Address Type Mask

These bits can be used to mask certain address type bits, thus allowing more than one address space type to be assigned to a chip-select. Any set bit causes the corresponding address type code bits to be used as part of the address comparison and any cleared bit masks the corresponding address type code bit. The ATM bits should be cleared so that address type codes are ignored as part of the address comparison. Be aware that following a system reset, these bits are reset in OR0.

CSNT—Chip-Select Negation Time

This bit is used to determine when the $\overline{CS}$/$\overline{WE}$ signals 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. Be aware that following a system reset, this bit is set in OR0.

0 = $\overline{CS}$/$\overline{WE}$ are negated normally.
1 = $\overline{CS}$/$\overline{WE}$ are negated a quarter of a clock earlier.

SAM—Start Address Multiplex

This bit 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 the UPMA or UPMB.

0 = Address pins are the address requested by the internal master.
1 = Address pins are the address requested by the internal master multiplexed according to the AMA field (if UPMA is selected to control the memory access) or the AMB field (if UPMB is selected).

ACS—Address to Chip-Select Setup

This bit can be used when the external memory access is handled by the general-purpose chip-select machine (GPCM). It allows $\overline{CS}$ assertion to be delayed when there is an address change. Be aware that following a system reset, these bits are set in OR0.

00 = $\overline{CS}$ is output at the same time as the address lines.
01 = Reserved.
10 = $\overline{CS}$ is output a quarter of a clock later than the address lines.
11 = $\overline{CS}$ is output half a clock later than the address lines.

G5LA/G5LS—General-Purpose Line 5 A and Line 5 Start

These bits determine how the $\overline{GPL5}$ pin is output when the memory access is handled by the UPMA or UPMB.

G5LA (valid only if MS = 11 in the $\overline{BR}$):
0 = Output $\overline{GPL5}$ on the $\overline{GPL5\_B}$ signal.
1 = Output $\overline{GPL5}$ on the $\overline{GPL5\_A}$ signal.

G5LS:
0 = The $\overline{GPL5}$ signal is driven low on the falling edge of GCLK1 during the first clock cycle of a read or write memory access.
1 = The $\overline{GPL5}$ signal is driven high on the falling edge of GCLK1 during the first clock cycle of a read or write memory access.

BI—Burst Inhibit

This bit determines whether or not this memory bank supports burst accesses. When a burst does not occurs, the memory controller drives the $\overline{BI}$ signal active when accessing this memory region. If the machine selected to handle this access is the GPCM, this bit must be set to 1. Be aware that following a system reset, this bit is set in OR0.

0 = Drive $\overline{BI}$ negated. The bank supports burst accesses.
1 = Drive $\overline{BI}$ asserted. The bank does not support burst accesses.

SCY—Cycle Length in Clocks

These bits determine the number of wait states inserted in the cycle when the general-purpose chip-select machine handles the external memory access and it is the main parameter for determining the cycle's length. The total cycle length may vary, depending on the settings of other timing attributes. The total memory access length is (2 + SCY) × clocks If an external $\overline{TA}$ response has been selected for this memory bank (by setting the SETA bit), then SCY[0:3] bits are not used. Be aware that following a system reset, these bits are set to 1111 in OR0.

0000 = 0 clock cycle wait state.
0001 = 1 clock cycle wait state.
0010 = 2 clock cycle wait states.
0011 = 3 clock cycle wait states.
0100 = 4 clock cycle wait states.
0101 = 5 clock cycle wait states.
0110 = 6 clock cycle wait states.
0111 = 7 clock cycle wait states.
1000 = 8 clock cycle wait states.
1001 = 9 clock cycle wait states.
1010 = 10 clock cycle wait states.
1011 = 11 clock cycle wait states.
1100 = 12 clock cycle wait states.
1101 = 13 clock cycle wait states.
1110 = 14 clock cycle wait states.
1111 = 15 clock cycle wait states.

SETA—External Transfer Acknowledge

This bit specifies when the $\overline{TA}$ signal is externally generated once the GPCM is selected to handle the memory access that was initiated to this memory region. Be aware that following a system reset, this bit is reset in OR0.

0 = $\overline{TA}$ is internally generated by the memory controller, unless asserted earlier.
1 = $\overline{TA}$ is generated by external logic.

TRLX—Timing Relaxed

When this bit is asserted, it modifies the timing of the signals controlling the memory devices once the GPCM is selected to handle the memory access that was initiated to this memory region. Be aware that following a system reset, this bit is set in OR0.

0 = Normal timing is generated by the GPCM.
1 = Relaxed timing is generated by the GPCM.

EHTR—Extended Hold Time on Read Accesses

When this bit is asserted, it inserts an idle clock cycle after a read access from the current bank and any MPC801 write or read access to a different bank occurs. Be aware that following a system reset, this bit is reset in OR0.

0 = Normal timing is generated by the memory controller.
1 = Extended hold time is generated by the memory controller.

Bit 31—Reserved

This bit is reserved and should be set to 0.

15.4.5 Machine A Mode Register

The machine A mode register (MAMR) contains the control bits for the user-programmable machine A.

MAMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PTA								PTAE	AMA			RES	DSA		RES
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	G0CLA			GPLA 4DIS	RLFA				WLFA				TLFA			

PTA—Periodic Timer A Period

These bits affect the periodic timer A and determine the timer period according to the following equation:

$$\text{TimerPeriod} = \left(\frac{\text{PTA}}{F_{\text{MPTC}}}\right)$$

For example, for a 25MHz BRGCLK with a required service rate of 15.6 microseconds, given PTP = 32, the PTA value should be 12 decimal. 12/ (25MHz / 32) = 15.36 microseconds, which is less than the required service period of 15.6 microseconds.

PTAE—Periodic Timer A Enable

This bit allows the periodic timer A to request service. Be aware that following a system reset, this bit is reset.

0 = Periodic timer A is disabled.
1 = Periodic timer A is enabled.

AMA—Address Multiplex Size A

These bits determine how the address of the current memory cycle is output on the address pins. The content of the RAM array in the UPMA controls the address output on the pins. These bits are used to connect the MPC801 to DRAM devices that require row and column addresses to be multiplexed on the same pins.

Bits 12 and 15—Reserved

These bits are reserved and should be set to 0.

DSA—Disable Timer Period

This bit guarantees a minimum time between accesses to the same memory bank if it is controlled by the UPMA. The TODT turns on the disable timer in the RAM array and, when expired, the UPMA allows the machine access to issue a memory pattern to the same memory region. Accesses to different memory regions can be handled by the same UPMA. To avoid conflicts with successive accesses to different memory regions, the minimum pattern in the RAM array for a serviced request should be equal to or greater than the period established by this bit.

00 = 1-cycle disable period.
01 = 2-cycle disable period.
10 = 3-cycle disable period.
11 = 4-cycle disable period.

G0CLA—General Line 0 Control A

These bits determine the address signal that is output to the $\overline{\text{GPL0}}$ pin when the UPMA is selected to control memory access.

000 = A12.
001 = A11.
010 = A10.
011 = A9.
100 = A8.
101 = A7.
110 = A6.
111 = A5.

GPLA4DIS—GPLA4 Output Line Disable

This bit determines whether or not the UPWAITA/$\overline{\text{GPLA4}}$ pin will behave as an output line controlled by the corresponding bits in the UPMA array ($\overline{\text{GPL4A}}$). Be aware that following a system reset, this bit is set.

0 = UPWAITA/$\overline{\text{GPLA4}}$ behaves as $\overline{\text{GPLA4}}$ when the G4T4/DLT3 bit in the UPMA is interpreted as G4T4 and when the G4T3/WAEN bit in the UPMA is interpreted as G4T3.
1 = UPWAITA/$\overline{\text{GPLA4}}$ behaves as UPWAITA when the G4T4/DLT3 bit in the UPMA is interpreted as DLT3 and when the G4T3/WAEN bit in the UPMA is interpreted as WAEN.

RLFA—Read Loop Field A

These bits determine the number of times a loop defined in the UPMA is executed for a burst read or single beat read pattern.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

WLFA—Write Loop Field A

This field 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.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

TLFA—Timer Loop Field A

These bits determine 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.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

15.4.6 Machine B Mode Register

The machine B mode register (MBMR) contains the control bits for the user-programmable B machine.

MBMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PTB								PTBE	AMB			RES	DSB		RES
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	G0CLB			GPLB 4DIS	RLFB				WLFB				TLFA			

PTB—Periodic Timer B

This bit affects the periodic timer B and determine the timer period according to the following equation:

$$\text{TimerPeriod} = \left(\frac{\text{PT}B}{\text{F}_{\text{MPTC}}}\right)$$

For example, a 25MHz BRGCLK with a required service rate of 15.6 microseconds, given PTP = 32, the PTB value should be 12 decimal. 12/ (25MHz / 32) = 15.36 microseconds, which is less than the required service period of 15.6 microseconds.

PTBE—Periodic Timer B Enable

This bit allows the periodic timer B to request service. Be aware that following a system reset, this bit is reset.

0 = Periodic timer B is disabled.
1 = Periodic timer B is enabled.

AMB—Address Multiplex Size B

These bits determine how the address of the current memory cycle is output on the address pins. The content of the RAM array in the UPMB controls the address output on the pins. These bits are used to connect the MPC801 to DRAM devices that require row and column addresses to be multiplexed on the same pins.

Bits 12 and 15—Reserved

These bits are reserved and should be set to 0.

DSA—Disable Timer Period

This bit guarantees a minimum time between accesses to the same memory bank if it is controlled by the UPMB. The TODT turns on the disable timer in the RAM array and, when expired, the UPMB allows the machine access to issue a memory pattern to the same memory region. Accesses to different memory regions can be handled by the same UPMB. To avoid conflicts with successive accesses to different memory regions, the minimum pattern in the RAM array for a serviced request should be equal to or greater than the period established by this bit.

00 = 1-cycle disable period.
01 = 2-cycle disable period.
10 = 3-cycle disable period.
11 = 4-cycle disable period.

G0CLB—General Line 0 Control B

These bits determine the address signal that is output to the $\overline{\text{GPL0}}$ pin when the UPMB is selected to control memory access.

000 = A12.
001 = A11.
010 = A10.
011 = A9.
100 = A8.
101 = A7.
110 = A6.
111 = A5.

GPLB4DIS—GPLB4 Output Line Disable

This bit determines whether or not the UPWAITB/$\overline{\text{GPLB4}}$ pin will behave as an output signal controlled by the corresponding bits in the UPMB array ($\overline{\text{GPL4B}}$). Be aware that following a system reset, this bit is set.

0 = UPWAITB/$\overline{\text{GPLB4}}$ behaves as $\overline{\text{GPLB4}}$ when the G4T4/DLT3 bit in the UPMB is interpreted as G4T4 and when the G4T3/WAEN bit in the UPMB is interpreted as G4T3.
1 = UPWAITB/$\overline{\text{GPLB4}}$ behaves as UPWAITB when the G4T4/DLT3 bit in the UPMB is interpreted as DLT3 and when the G4T3/WAEN bit in the UPMB is interpreted as WAEN.

RLFB—Read Loop Field B

These bits determine the number of times a loop defined in the UPMB is executed for a burst read or single beat read pattern.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

WLFB—Write Loop Field B

This field determines the number of times a loop defined in the UPMB is executed for a burst write or a single beat write patterns.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

TLFB—Timer Loop Field B

These bits determine 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.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

15.4.7 Memory Command Register

The memory command register (MCR) controls user-programmable read-write operations. It also stimulates UPM routine execution.

MCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	OP		RESERVED						UM	RESERVED						
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	MB			RES	MCLF				RESERVED		MAD					

OP—Command Opcode

This field determines the command to be executed by the machine that is specified in the UM field.

00 = Write the contents of the MDR bit into the RAM location pointed by the MAD bit in the UPM that is specified in the UM bit.
01 = Read the contents of the RAM location pointed by the MAD bit in the UPM that is specified in the UM bit into the MDR.
10 = Run the pattern written in the RAM array of the UPM specified in the UM bit servicing the memory bank that is specified in the MB bit. The pattern run starts at the location pointed to by the MAD bit and continues until the LAST bit in the RAM is set.
11 = Reserved.

Bits 2–7—Reserved

These bits are reserved and should be set to 0.

UM—User Machine

These bits indicate the user-programmable machine to which the command is executed.

0 = UPMA.
1 = UPMB.

Bits 9–15—Reserved

These bits are reserved and should be set to 0.

MB—Memory Bank

These bits indicate the enabling of the $\overline{CS}$ signal when a RUN_PATTERN command is executed.

000 = $\overline{CS0}$ enabled.
001 = $\overline{CS1}$ enabled.
010 = $\overline{CS2}$ enabled.
011 = $\overline{CS3}$ enabled.
100 = $\overline{CS4}$ enabled.
101 = $\overline{CS5}$ enabled.
110 = $\overline{CS6}$ enabled.
111 = $\overline{CS7}$ enabled.

Bit 19, 24, 25—Reserved

These bits are reserved and should be set to 0.

MCLF—Memory Command Loop Field

These bits determine the number of times a loop is executed for a MEMORY COMMAND service pattern.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

MAD—Machine Address

These bits make up the executed command's RAM address pointer.

15.4.8 Memory Data Register

The memory data register (MDR) is used as a data source in UPM write operations and a destination in UPM read operations.

MDR

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

MD—Memory Data

These bits represent the data to be written into the RAM array when a write command is issued to the memory command register. It is also the data read from the array when a read command is issued.

15.4.9 Memory Address Register

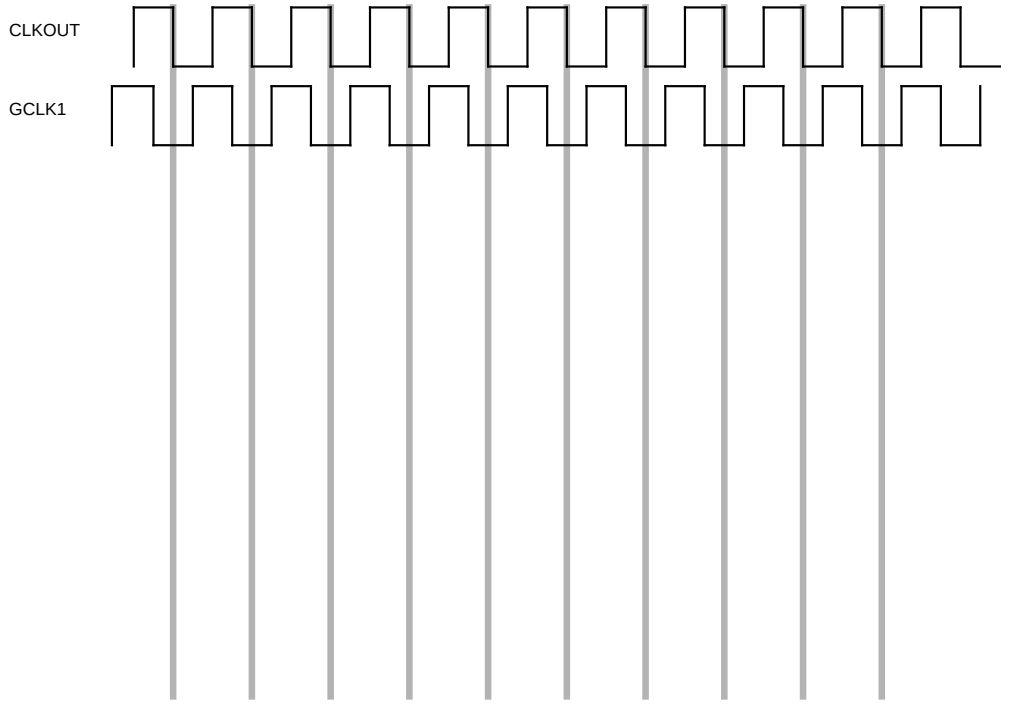
The memory address register (MAR) contains an address to be output on the address pin that is controlled by the AM*x* bit in the user-programmable machine.

MAR

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

A—Memory Address

These bits are output to the address signals that are controlled by the AM*x* bits in the user-programmable machine.



cst4 cst1 cst2 cst3					Bit 0 Bit 1 Bit 2 Bit 3
bst4 bst1 bst2 bst3					Bit 4 Bit 5 Bit 6 Bit 7
g0l0 g0l1 g0h0 g0h1					Bit 8 Bit 9 Bit 10 Bit 11
g1t4 g1t3					Bit 12 Bit 13
g2t4 g2t3					Bit 14 Bit 15
g3t4 g3t3					Bit 16 Bit 17
g4t4 g4t3					Bit 18 Bit 19
g5t4 g5t3					Bit 20 Bit 21
- -					Bit 22 Bit 23
loop exen					Bit 24 Bit 25
amx0 amx1 na uta todt last					Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31
	RSS	**RSS+1**	**RSS+2**		

Figure 15-56. Blank Worksheet for a UPM

SECTION 16
SERIAL COMMUNICATION MODULES

This section discusses the five serial communication modules of the MPC801:

- Two universal asynchronous receiver transmitter controllers
- Serial peripheral interface
- I^2C controller
- Parallel I/O port

16.1 THE UART CONTROLLERS

The universal asynchronous receiver transmitter (UART) controllers provide serial communication with external devices (modems and other computers) using the standard “start-stop” format. They perform all normal operations associated with “start-stop” asynchronous communication. The serial data is transmitted and received at standard bit rates using the internal baud rate generator. For those applications that need other bit rates, a 1× clock mode is provided. However, you must provide general-purpose chip-select machine the data bit clock. Figure 4-1 illustrates the block diagram of this module.

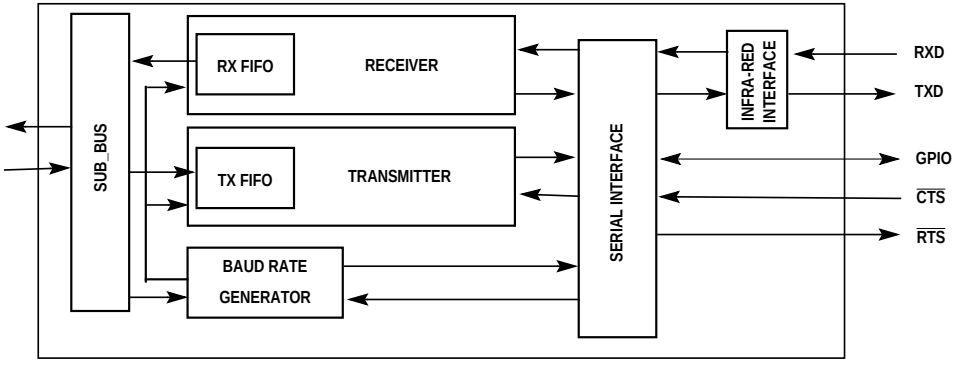


Figure 16-1. UART Block Diagram

16.1.1 Features

The following list summarizes the main features of the UART controllers:

- Full Duplex Operation
- Flexible 5-Wire Serial Interface
- Direct Support of IrDA Physical Layer Protocol
- Robust Receiver Data Sampling with Noise Filtering
- 8-Byte FIFOs for Transmit and Receive
- 7- and 8-Bit Operation with Optional Parity
- Generation and Detection of Break
- Baud Rate Generator
- Flexible Clocking Options
- Standard Baud Rates of 300bps to 115.2kbps with a 16× Sample Clock
- External 1× clock for High-Speed Synchronous Communication
- Eight Maskable Interrupts
- Optional Low-Power Idle Mode

16.2 SERIAL INTERFACE SIGNALS

You can access the following signals, which become default signals after reset. If you do not need any UART signals, program the appropriate port as an I/O.

- Transmit Data (TXD)—This pin is the transmitter serial output. While in normal mode, Non Return to Zero (NRZ) data is output. While in IrDA mode, a 3- or 16-bit period pulse is output for each “zero” bit transmitted. For RS-232 applications, this pin must be connected to an RS-232 transmitter. For infra-red applications this pin can directly drive an infra-red LED.
- Clear to Send ($\overline{\text{CTS}}$)—This active low input is used to control the transmitter. Normally, the transmitter waits until this signal is active (low) before a character is transmitted. If the ICTS bit of the transmitter register is set, the transmitter sends a character whenever one is ready to transmit. This pin can then be used as a general-purpose input whose status is read in the CTSS bit of the transmitter register. It can post an interrupt on any transition of this pin if the interrupt is enabled.
- Receive Data (RXD)—This pin is the receiver serial input. While in normal operation, NRZ data is expected. However, while in infra-red mode, a narrow pulse is expected for each “zero” bit received. External circuitry must be used to convert the infra-red signal to an electrical signal. RS-232 applications need an external RS-232 receiver to convert from RS-232 voltage levels.

- Ready to Send ($\overline{\text{RTS}}$)—This output pin serves two basic purposes. Normally, the receiver indicates that it is ready to receive data by asserting this pin (low), which would be connected to the end transmitter's $\overline{\text{CTS}}$ pin. When the receiver detects a pending overrun, it negates this pin. However, for other applications, the $\overline{\text{RTS}}$ pin can be a general-purpose output controlled by the $\overline{\text{RTS}}$ pin in the global register.
- General-Purpose Input/Output (GPIO)—This bidirectional pin serves several purposes. It can be a general-purpose input that posts interrupts on any transition or it can be a general-purpose output controlled by the GPIO bit in the baud register. It can also serve as the clock source to the baud rate generator and output the bit clock at the selected baud rate.

16.2.1 Sub-Block Description

This UART module is easy to use from both a hardware and software perspective because five working registers perform all possible status and control functions. In addition, all register bits can be read and most of them are read/write. The modem control signals are flexible.The $\overline{\text{CTS}}$ pin is an input that can either provide hardware flow-control to the transmitter or be used as a general-purpose input.

A maskable interrupt is posted on each transition of this signal. The $\overline{\text{RTS}}$ pin is an output from the receiver that indicates when the receiver FIFO is not full. This bit can be configured as a general-purpose output. A GPIO pin is provided that can be used to bring an external bit-clock into the module or it can be used as a general-purpose input with a maskable interrupt posted on each transition. It can even be configured as an output that provides a bit-clock or signal under software control. The UART consists of four submodules—the transmitter, receiver, baud rate generator, and global controller interface.

16.2.1.1 THE TRANSMITTER. The transmitter accepts a parallel character from the PowerPC™ core and transmits it serially. While the FIFO is empty, the transmitter outputs a continuous IDLE bit (1 in NRZ mode and 0 in IrDA mode). When a character is available for transmission, the START, STOP, and PARITY (if enabled) bits are added to the character and it is serially shifted at the selected bit rate. The transmitter posts a maskable interrupt when it is ready for parallel data. Three interrupt sources are available. If you want to take full advantage of the 8-byte FIFO, you should enable the FEM interrupt. In the interrupt service routine, the FIFO should be interrogated after each byte is loaded.

If there is space available and the TXAVEN bit of the control register is set, more data is loaded into the FIFO. Another interrupt from the transmitter will not be generated until the FIFO has completely emptied. If your software has a large interrupt service latency, the FHAL of the transmitter register interrupt can be used. In this case, an interrupt is generated when the occupancy in the FIFO is less than half of its size. If you do not need the FIFO, you should use the TXAVEN interrupt of the control register. An interrupt is generated when at least one space is available in the FIFO.

$\overline{CTS}$ controls the flow of the serial data as needed. If $\overline{CTS}$ is negated, the transmitter finishes sending the character in progress, then stops and waits for $\overline{CTS}$ to become asserted. A BRK character can be generated by setting the SBRK bit of the transmitter register. However, your software must be aware of the baud rate. The SBRK bit must be asserted for a sufficient amount of time to generate a valid BRK character. For debugging purposes, parity errors can be generated. The transmitter operates from the 1× clock provided by the baud rate generator. While the infra-red interface is enabled, the transmitter produces a pulse that is $^3/_{16}$ of a bit time for each "zero" bit sent. The TXD port can drive an infra-red LED or interface to popular IrDA transceivers.

16.2.1.1.1 Transmitter Register. This register controls the operation of the transmitter and resets to $0000.

TRANSMITTER

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	FEM	FHAL	TXAV	SBRK	ICTS	RES	CTSS	CTSD	TX DATA							
RESET	$E000															

FEM—FIFO Empty

This read-only bit indicates that the transmit FIFO is empty.

0 = Transmit FIFO is not empty.
1 = Transmit FIFO is empty.

FHAL—FIFO Half

This read-only bit indicates whether or not the transmit FIFO is half full.

0 = Transmit FIFO is less than half full.
1 = Transmit FIFO is at least half full.

TXAV—TX Available

This read-only bit indicates that the transmit FIFO has at least one slot available for data.

0 = Transmit is full.
1 = Transmit is not full.

SBRK—Send Break

This bit forces the transmitter to send a BRK character. The transmitter finishes sending the character in progress and then sends the BRK bit until this bit is reset. You are responsible for ensuring that this bit is high for a sufficient amount of time to generate a valid BRK bit. Then you continue filling the FIFO and any remaining characters are transmitted when the BRK bit is terminated.

0 = Do not send the BRK bit.
1 = Send the BRK bit.

ICTS—Ignore $\overline{\text{CTS}}$

When it is high, this bit forces the $\overline{\text{CTS}}$ pin to be asserted, thus effectively ignoring the external pin. While in this mode, the $\overline{\text{CTS}}$ pin can be used as a general-purpose input.

0 = Transmit only while the $\overline{\text{CTS}}$ pin is asserted.
1 = Ignore the $\overline{\text{CTS}}$ pin.

CTSS—$\overline{\text{CTS}}$ Status

This bit indicates the current status of the $\overline{\text{CTS}}$ pin. A snapshot of the pin is taken before this bit is presented to the data bus. When the ICTS pin is high, this bit can be used as a general-purpose input.

0 = $\overline{\text{CTS}}$ pin is low.
1 = $\overline{\text{CTS}}$ pin is high.

CTSD—$\overline{\text{CTS}}$ Delta

When it is high, this bit indicates that the $\overline{\text{CTS}}$ pin has been changed. It generates a maskable interrupt. The current status of the $\overline{\text{CTS}}$ pin can be found on the CTSS bit. The $\overline{\text{CTS}}$ interrupt is cleared by writing a zero to this bit.

0 = $\overline{\text{CTS}}$ pin has not been changed since it was last cleared.
1 = $\overline{\text{CTS}}$ pin has been changed.

TX DATA—Transmit Data

These bits are the parallel transmit data inputs. When in 7-bit mode, Bit 8 is ignored. When in 8-bit mode, all bits are used. Data is transmitted beginning with the least-significant bit. A new character is transmitted when these bits are written.

16.2.1.2 THE RECEIVER. The receiver accepts a serial datastream and converts it into a parallel character. It operates in two modes—16× and 1×. In 16× mode, it searches for a START bit, qualifies it, then samples the succeeding DATA bits at the bit's center. Jitter tolerance and noise immunity are provided by sampling at a 16× rate and using a voting technique to clean up the samples. In 1× mode, the RXD bit is sampled on each rising edge of the bit clock. Once the START bit has been found, the DATA, PARITY, and STOP bits are shifted in.

If parity is enabled, it is checked and its status is reported in the receiver register. Similarly, frame errors and breaks are checked and reported. When a new character is ready to be read by the core, the $\overline{\text{RTS}}$ pin is asserted and an interrupt is posted (if enabled). When the receiver register is read as a 16-bit word, the complete FIFO status, the four status bits, and the received character byte are read by the core. The $\overline{\text{RTS}}$ pin can be configured as an output that indicates the receiver is ready for data or it can be directly controlled by the software.

As with the transmitter, the receiver FIFO is flexible. If your software has a short interrupt latency, the FFU interrupt can be enabled. One space is available in the FIFO when this interrupt is generated. By reading the receiver register as a word, the status of the FIFO is presented to the core along with the data. If the FIFO status indicates that data remains in the FIFO, it can then be emptied byte-by-byte. If the software has a longer latency, the FHAL interrupt can be used. This interrupt is generated when at least four bytes in the FIFO are ready to be read. If the FIFO is not needed, the DRDY interrupt can be used. This interrupt is generated whenever one or more characters are in the FIFO. When the infra-red interface is enabled, the receiver expects narrow pulses for each “zero” bit received. Otherwise, normal NRZ is expected. An IrDA transceiver, external to the MPC801, transforms the infra-red signal to an electrical signal.

NOTE

Any program that deals with the UART by polling must read the FIFO status bit from the receiver register. Since a normal read from that register has a side effect, it updates the FIFO. Therefore, you should use a special receiver register address that returns receiver data, but does not affect the FIFO.

16.2.1.2.1 Receiver Register. This read-only register controls the receiver and the status of each received character is read from this register. The highest nibble of this register resets to $0. The other bits contain random data until the first character is received. There are two addresses for this register. Generally, reading this register updates the receiver FIFO. Reading via the special address does not affect the FIFO, which is used for polling the channel.

RECEIVER

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	FFU	FHAL	DRDY	RES	OVR	FERR	BRK	PERR	RX DATA							
RESET	$0XXX															

FFU—FIFO Full

This bit indicates that the receive FIFO is full and may generate an overrun.

0 = Receive FIFO contains more than one available space.
1 = Receive FIFO contains a maximum of one available space.

FHAL—FIFO Half

This bit indicates whether or not the receive FIFO is half full.

0 = Receive FIFO is less than half full.
1 = Receive FIFO is at least half full.

DRDY—Data Ready

This bit indicates that at least one byte is in the receive FIFO.

0 = Receive FIFO is empty.
1 = Receive FIFO is not empty.

OVR—FIFO Overrun

When it is high, this bit indicates that the receiver overwrote data in the FIFO. The character with this bit set is valid, but at least one previous character was lost. Under normal circumstances, this bit should never be set. It indicates that your software is not keeping up with the incoming data rate. This bit is up-to-date and valid for each received character.

0 = No FIFO overrun.
1 = A FIFO overrun is detected.

FERR—Frame Error

When it is high, this bit indicates that the current character had a framing error (missing the STOP bit). The data is possibly corrupted. This bit is updated for each character read from the FIFO.

0 = Current character has no framing error.
1 = Current character has a framing error.

BRK—Break

When it is high, this bit indicates that the current character is a break. The data bits are all zero and the stop bit is also zero. The FERR bit will always be set when this bit is set. If odd parity is selected, the PERR bit will be set when this bit is set. This bit is updated for each character read from the FIFO.

0 = Current character is not a break character.
1 = Current character is a break character.

PERR—Parity Error

When it is high, this bit indicates that the current character has a parity error and the data could be corrupted. This bit is updated for each character read from the FIFO. While parity is disabled, this bit always reads zero.

RX DATA—Receive Data

These bits are the next characters received into the FIFO. They have no meaning if the DRDY bit is zero. When in 7-bit mode, the most-significant bit is forced to zero. When in 8-bit mode, all bits are active.

16.2.1.3 THE BAUD RATE GENERATOR. The baud rate generator provides bit clocks to the transmitter and receiver blocks. It consists of a prescaler that divides the clock source by any integer between 2 and 64. The output of the prescaler is then further divided by a 2^n divider. Eight taps are available at 1, 2, 4, 8, 16, 32, 64, and 128. The selected tap is the 16× clock for the receiver. This clock is even further divided by 16 to provide the 50% duty-cycle 1× clock to the transmitter.

The baud rate generator has enough flexibility to provide almost any standard baud rate from a variety of clock frequencies. The baud rate generator is flexible in that its master clock source can be the baud rate generator clock or it can be provided by the GPIO pin. By configuring the proper bit in port B as an input and setting the baud source bit to 1, the baud rate generator can be driven by an external signal. For synchronous applications, the GPIO pin can also be configured as an input or output for the 1× bit clock.

16.2.1.4 THE GLOBAL CONTROLLER INTERFACE. The global controller interface contains the baud control register and the control register, as well as all global logic. The interrupt line is the logical-OR of the eight interrupt sources. The interrupt level that is sent to the PowerPC core is user programmable. When the UARTEN bit is low, the master clock is disabled. In this mode, power consumption is reduced to a minimum.

16.2.1.4.1 Control Register. This register controls the overall operation of the UART controller and it resets to $0000.

CONTROL

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UART EN	RX EN	TX EN	RXCLK CONT	PEN	ODE	SL	CL	GPIO DEEN	CTS DEEN	RX FUEN	RX HAEN	RX RDYEN	TX EMEN	TX HAEN	TX AVEN
RESET VALUE: $0000															

UARTEN—UART Enable

This bit enables the UART controller. When it is low, the UART controller is disabled and in low-power mode. When it is high, the UART is active.

0 = The UART controller is disabled.
1 = The UART controller is enabled.

NOTE

The status bits of the baud control, receiver, and transmitter registers are not valid unless the UARTEN bit is set.

RXEN—RX Enable

This bit enables the receiver block. When this bit is low, the receiver is disabled and the receive FIFO is flushed.

0 = Receiver is disabled and receive FIFO is flushed.
1 = Receiver is enabled.

TXEN—TX Enable

This bit enables the transmitter block. When this bit is low, the transmitter is disabled and the transmit FIFO is flushed.

0 = Transmitter is disabled and transmit FIFO is flushed.
1 = Transmitter is enabled.

RXCLKCONT—Receiver Clock Control

This bit controls the operating mode of the receiver. When this bit is low, the receiver is in 16× mode where it synchronizes to the incoming datastream and samples at the perceived center of each bit period. When it is high, the receiver is in 1× mode where it samples the datastream on each rising edge of the bit clock.

0 = 16× clock mode.
1 = 1× clock mode.

PEN—Parity Enable

This bit controls the parity generator in the transmitter and the parity checker in the receiver. When this bit is high, they are enabled. When it is low, they are disabled.

0 = Parity is disabled.
1 = Parity is enabled.

ODE—Odd Even

This bit controls the sense of the parity generator and checker. When it is high, odd parity is generated and expected. When it is low, even parity is generated and expected. This bit does not function if the PEN bit is low.

0 = Even parity.
1 = Odd parity.

SL—Stop Bits

This bit controls the number of stop bits transmitted after a character. When it is high, two stop bits are sent. When it is low, one stop bit is sent. However, this bit has no effect on the receiver that expects one or more stop bits.

0 = One stop bit is transmitted.
1 = Two stop bits are transmitted.

CL—Character Length

This bit controls the character length. When it is high, the transmitter and receiver are in 8-bit mode. When it is low, they are in 7-bit mode. The transmitter ignores D7 and the receiver sets it to zero.

0 = 7-bit transmit and receive character length.
1 = 8-bit transmit and receive character length.

GPIODEEN—GPIO Delta Enable

This bit enables an interrupt when the GPIO pin (configured as an input) changes state. The current state of the GPIO pin is read in the baud control register.

0 = GPIO interrupt is disabled.
1 = GPIO interrupt is enabled.

CTSDEEN—$\overline{\text{CTS}}$ Delta Enable

When it is high, this bit enables an interrupt as the $\overline{\text{CTS}}$ pin changes state. When it is low, this interrupt is disabled. The current status of the $\overline{\text{CTS}}$ pin is read in the transmitter register.

0 = $\overline{\text{CTS}}$ interrupt is disabled.
1 = $\overline{\text{CTS}}$ interrupt is enabled.

RXFUEN—RX Full Enable

When it is high, this bit enables an interrupt as the receiver FIFO becomes full.

0 = RX FULL interrupt is disabled.
1 = RX FULL interrupt is enabled.

RXHAEN—RX Half Enable

When it is high, this bit enables an interrupt as the receiver FIFO gets more than half full.

0 = RX HALF interrupt is disabled.
1 = RX HALF interrupt is enabled.

RXRDYEN—RX Ready Enable

When it is high, this bit enables an interrupt as the receiver accumulates at least one data byte in the FIFO. When it is low, this interrupt is disabled.

0 = RX interrupt is disabled.
1 = RX interrupt is enabled.

TXEMEN—TX Empty Enable

When it is high, this bit enables an interrupt as the transmitter FIFO becomes empty and needs data. When it is low, this interrupt is disabled.

0 = TX EMPTY interrupt is disabled.
1 = TX EMPTY interrupt is enabled.

TXHAEN—TX Half Enable

When it is high, this bit enables an interrupt as the transmit FIFO gets more than half empty. When it is low, the TX HALF interrupt is disabled.

0 = TXHAEN interrupt is disabled.
1 = TXHAEN interrupt is enabled.

TXAVEN—TX Available Enable

When it is high, this bit enables an interrupt as the transmitter makes a slot available in the FIFO. When it is low, the TX AVAIL interrupt is disabled.

0 = TXAVEN interrupt is disabled.
1 = TXAVEN interrupt is enabled.

16.2.1.4.2 Baud Control Register. This register controls the operation of the baud rate generator and the GPIO pin.

BAUD CONTROL

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	GDEL	GPIO	G DIR	GSRC	BSRC	DIV			RESERVED		PRE					
RESET	$003F															

GDEL—General-Purpose Input Delta

This bit indicates that a change occurred on the GPIO pin while the GPIO pin is used as an input pin. If the GPIO interrupt is enabled, this bit posts an interrupt. This bit is cleared by writing zero to clear the GPIO interrupt.

0 = GPIO pin has not been changed since it was last cleared.
1 = GPIO pin has been changed.

GPIO—General-Purpose Input/Output

This bit contains the current status of the GPIO pin. If GPIO is configured as an input, the user can only read this bit. If it is configured as an output, this bit controls the pin.

0 = GPIO pin is low.
1 = GPIO pin is high.

GDIR—General-Purpose Input/Output Direction

This bit controls the direction of the GPIO pin. When it is high, the pin is an output. When it is low, GPIO is an input.

0 = GPIO pin is an input.
1 = GPIO pin is an output.

GSRC—General-Purpose Input/Output Source

This bit controls the source of the GPIO pin. When it is high, the source is the 1× clock from the baud rate generator. When it is low, the source is the GPIO bit. However, when the GPIO is an input pin, this bit does not function.

0 = GPIO is driven by the GPIO bit.
1 = GPIO is driven by the bit clock from the baud generator.

BSRC—Baud Source

This bit controls the clock source to the baud rate generator.

0 = Baud generator source is baud rate generator clock.
1 = Baud generator source is the GPIO pin, which must be an input pin.

DIV—Divider

These bits control the clock frequency generated by the baud generator and are encoded as follows:

000 = Divide by 1.
001 = Divide by 2.
010 = Divide by 4.
011 = Divide by 8.
100 = Divide by 16.
101 = Divide by 32.
110 = Divide by 64.
111 = Divide by 128.

Bits 8–9—Reserved

These bits are reserved and should be set to 0.

PRE—Prescaler

These bits control the divide value of the baud generator prescaler. The value of the PRE bit must be greater than or equal to 1 or an erroneous operation can occur. The divide value is determined by the following formula:

Prescaler divide value = 65 (decimal) – PRE

16.2.1.4.3 Typical Asynchronous Communication Baud Rate Examples. The required values of the DIV and PRE fields for typical bit rates of asynchronous communication is shown below in Table 4-1. Notice that for this mode, the internal clock rate is assumed 16× the baud rate.

Table 16-1. Typical Baud Rates of Asynchronous Communication

BAUD RATES	MPC801 SYSTEM FREQUENCY (MHZ)								
	20			24.5760			33		
	DIVIDER	PRE-SCALER	ACTUAL FREQUENCY	DIVIDER	PRE-SCALER	ACTUAL FREQUENCY	DIVIDER	PRE-SCALER	ACTUAL FREQUENCY
300	7	32	295.93	7	25	300	7	11	298.39
600	6	32	591.85	6	25	600	6	11	596.79
1,200	5	32	1183.7	5	25	1200	5	11	1193.6
2,400	4	32	2367.4	4	25	2400	4	11	2387.2
4,800	3	32	4735	3	25	4800	3	11	4774
9,600	2	32	9470	2	25	9600	2	11	9549
19,200	1	32	18939	1	25	19200	1	11	19097
38,400	0	32	37879	0	25	38400	0	11	38194
57,600	0	43	56818	0	38	56889	0	29	57292
115,200	0	54	113636	0	52	118154	0	47	114583

NOTE: All Values Are Decimal.

16.2.1.4.4 Global Register. This register contains global control and testing bits of the UART block and resets to $0000.

GLOBAL

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	CLSRC	FPERR	LOOP	RESERVED		RXPOL	TXPOL	RTSC	RTS	IRDEN	IRDAL	RES	UIPL [0:2]		
RESET	: $E000															

Bits 0, 4–5, and 12—Reserved

These bits are reserved and should be set to zero.

CLSRC—Clock Source

This bit selects the source of the 1× bit clock for transmit and receive. When it is high, the bit clock is derived from the GPIO pin and must be configured as an input. When it is low (normal), the bit clock is supplied by the baud rate generator. This bit allows high-speed synchronous applications in which a clock is provided by the external system.

0 = Bit clock is generated by baud rate generator.
1 = Bit clock is supplied by GPIO.

FPERR—Force Parity Errors

When this bit is high, it forces the transmitter to generate parity errors if parity is enabled. This bit is provided for system debugging.

0 = Generate normal parity.
1 = Generate a parity error.

LOOP—Loopback

This bit controls a loopback from the transmitter to the receiver. When this bit is high, the receiver input is internally connected to the transmitter and ignores the RXD pin. The transmitter is unaffected by this bit. The receiver can be in 16× or 1× clock mode for proper operation. This bit is provided for system testing.

0 = Normal receiver operation.
1 = Internally connect transmitter output to receiver input.

RXPOL—Received Data Polarity

This bit selects the function of the RXD pin.

0 = Normal.
1 = Inverted RXD.

TXPOL—Transmit Data Polarity

This bit selects the function of the TXD pin.

0 = Normal.
1 = Inverted TXD.

RTSC—$\overline{\text{RTS}}$ Control

This bit selects the function of the $\overline{\text{RTS}}$ pin.

0 = $\overline{\text{RTS}}$ pin is controlled by the $\overline{\text{RTS}}$ bit.
1 = $\overline{\text{RTS}}$ pin is controlled by the receiver FIFO. When the FIFO is full (one slot remaining), $\overline{\text{RTS}}$ is negated.

RTS—Ready To Send

This bit controls the $\overline{\text{RTS}}$ pin while the $\overline{\text{RTSC}}$ bit is zero.

0 = $\overline{\text{RTS}}$ pin is 1.
1 = $\overline{\text{RTS}}$ pin is 0.

IRDEN—IrDA Enable

This bit enables the IrDA interface.

0 = Normal NRZ operation.
1 = IrDA operation.

IRDAL—IrDA Loopback

This bit controls a loopback from the transmitter to the receiver in the IrDA interface. It is provided for system testing.

0 = No infra-red loop.
1 = Connect the infra-red transmit to the infra-red receiver.

NOTE

The LOOP bit must be zero for IrDA interface loopback testing.

UIPL—UART Interrupt Priority Level

This bit contains the priority request level of the UART interrupt that is sent to interrupt level 0-7.

16.3 THE SERIAL CONTROLLER

The serial controller module serves both the serial peripheral interface and the inter-integrated circuit (I^2C) modules. Its primary responsibility is to act as an interface between the U-bus and peripheral bus. It can also generate a reset signal to both the serial peripheral interface and I^2C modules, according to the PowerPC core commands.

16.3.1 Programming the Serial Controller

16.3.1.1 SERIAL CONTROLLER COMMAND REGISTER. The read/write serial controller command (SECCOM) register controls the serial peripheral interface and I^2C operation modes.

SEECOM

BIT	0	1	2	3	4	5	6	7
FIELD	RST	RESERVED						

RST—Software Reset Command

This bit is set by the core and cleared by the serial controller. It is useful when the core wants to reset the registers and state machines of the I^2C and serial peripheral interface channels. This command does not affect the parallel I/O registers.

Bits 1–7—Reserved

These bits are reserved and should be set to zero.

16.3.2 The Serial Peripheral Interface

The serial peripheral interface (SPI) allows the MPC801 to exchange data with other chips in the MPC860 Family, the MC68360, MC68302, M68HC11, and M68HC05 microcontroller families, as well as a number of peripheral devices. The serial peripheral interface is a full-duplex, synchronous, character-oriented channel that supports a four-wire interface—receive, transmit, clock and slave select.

The SPI block consists of transmitter and receiver sections, an independent baud rate generator, and a control unit. The transmitter and receiver sections use the same clock that 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. Figure 4-2 below illustrates the serial peripheral interface block diagram.

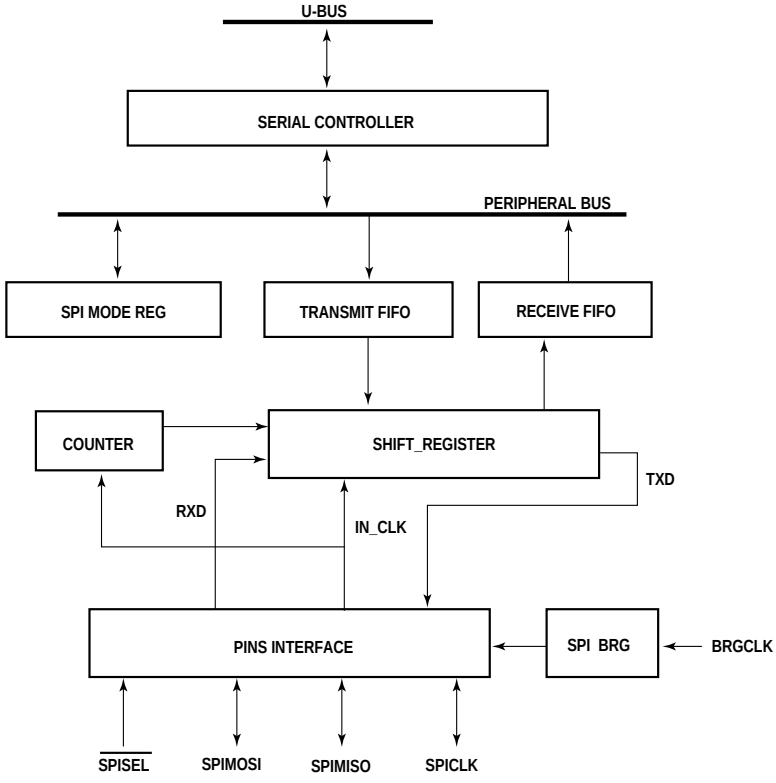


Figure 16-2. SPI Block Diagram

The depth of the SPI receiver and transmitter FIFOs is two characters. Combined with the shift register, this corresponds to an effective FIFO depth of three characters. The MPC801 serial peripheral interface most-significant bit is shifted out first and when the serial peripheral interface is not enabled in the SPMODE register, it consumes the least amount of power.

16.3.2.1 FEATURES . The following is a list of the serial peripheral interface's main features:

- Four-Wire Interface
- Full-Duplex Operation
- 8- and 16-Bit Data Character Operation
- Back-to-Back Character Transmission and Reception Support
- Master or Slave SPI Mode Support
- Multimaster Environment Support
- Clock Rate Support at a Maximum of 6.25MHz in Master Mode and 12.5MHz in Slave Mode, Assuming a 25MHz System Clock

- Independently Programmable Baud Rate Generator
- Programmable Clock Phase and Polarity
- Open-Drain Output Pins Support Multimaster Configuration
- Local Loopback Capability for Testing

16.3.2.2 CLOCKING AND PIN FUNCTIONS. The serial peripheral interface can be configured as a master for the serial channel and generate both the enable and clock signals. It can also be configured as a slave in which both the enable and clock signals are inputs. The serial peripheral interface supports operation in a multimaster environment. When the serial peripheral interface is a master, the SPI baud rate generator is used to generate the SPI transmit and receive clocks. The SPI baud rate generator takes its input from the baud rate generator clock, which is generated in the clock synthesizer of the MPC801.

The serial peripheral interface's SPIMISO pin is an input in master mode and an output in slave mode. The serial peripheral interface's SPIMOSI pin is an output in master mode and an input in slave mode. The reason the pin names change functionality between master and slave mode is to support a multimaster configuration that allows communication from one serial peripheral interface to another with the same hardware configuration. When the serial peripheral interface is a master, SPICLK is the clock output pin that shifts in the received data from the SPIMISO pin and shifts out the transmitted data to the SPIMOSI pin. Additionally, an SPI master device must provide a slave select signal output to enable the SPI slave devices. You can implement this using one of the general-purpose I/O pins. However, the $\overline{\text{SPISEL}}$ pin should not be asserted while the SPI is working as a master or an error will occur.

When the serial peripheral interface is a slave, SPICLK is the clock input pin that shifts in the received data from the SPIMOSI pin and shifts out the transmitted data to the SPIMISO pin. The $\overline{\text{SPISEL}}$ pin provided by the MPC801 is the enable input to the SPI slave. When the serial peripheral interface is in a multimaster environment, the $\overline{\text{SPISEL}}$ pin is still an input and is used to find an error condition when more then one master is operating.

SPICLK is a gated clock (the clock toggles only while data is being transferred) and four phase/polarity combinations of SPICLK are available. You can select any of them using two bits in the SPI mode register. The serial peripheral interface pins can also be configured as open-drain pins to support a multimaster configuration where the same pin can either be driven by the MPC801 or an external SPI device.

16.3.2.3 SPI TRANSMISSION AND RECEPTION PROCESS. Since the serial peripheral interface is a character-oriented communication unit, the PowerPC core is responsible for packing and unpacking the receive/transmit frames.The core supplies and collects words to or from the serial peripheral interface as requested. The core receives data by reading the SPI receive data hold (SPIRD) register. It resets the F bit in the SPI event register (SPIER) to free up the SPIRD register for the next operation. The core transmits data by writing it into the SPI transmit data hold (SPITD) register when the FIRST or LAST word is indicated.

The SPI core handshake protocol can be implemented by using a polling or interrupt mechanism. When using a polling mechanism, the core reads the SPIER in a predefined frequency and acts according to the value of the SPIER bits.The polling frequency depends on the SPI serial channel frequency. When using an interrupt mechanism, setting either the E or F bits of the SPIER causes an interrupt to the core. The interrupt level is determined by the SPIRL bits of the SPIMR. The core then reads the SPIER and acts appropriately. There are three basic modes of operation for transmitting and receiving—master, slave, and multimaster.

NOTE

Once the core realizes that the F and E bits are set,
it should treat the receive request first.

16.3.2.3.1 SPI Master Mode. When the serial peripheral interface is in master mode, it transmits a message to the peripheral or SPI slave, which replies simultaneously. When the MPC801 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 core writes the data to be transmitted into the SPITD register. It then sets the STR bit in the SPCOM register to start transmitting 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 serial peripheral interface shifts the received data in from the SPIMISO pin. During the transmission process, the core is responsible for supplying the data whenever the serial peripheral interface requests it to ensure smooth operation. The STR bit should only be set for the first character of any transmission. The serial peripheral interface continues transmitting and receiving characters until the L bit in the SPITD register is set or an error occurs.

The serial peripheral interface sets the E bit of the SPIER and issues a maskable interrupt to the system interface unit interrupt controller whenever its transmit buffer is not full. It also sets the E bit after sending the last word. In response, the core should read the exception flags that relate to the last word. The serial peripheral interface sets the F bit of the SPIER and issues a maskable interrupt to the system interface unit interrupt controller whenever its receive buffer has been filled with data. If the serial peripheral interface is the only master in a system, the $\overline{\text{SPISEL}}$ pin can be used as a general-purpose I/O and the internal $\overline{\text{SPISEL}}$ pin to the serial peripheral interface will always be forced inactive internally, thus eliminating the possibility of a multimaster error.

16.3.2.3.2 SPI Slave Mode. When the serial peripheral interface is in slave mode, it receives messages from an SPI master and replies simultaneously. The $\overline{\text{SPISEL}}$ pin must be asserted before receive clocks will be recognized and once it is asserted, the SPICLK pin becomes an input from the master to the slave. The SPICLK pin can vary frequency from DC to the BRGCLK/2 (12.5MHz for a 25MHz system). Before the data exchange, the core writes the data to be transmitted into the SPITD register. The core should then set the STR bit in the SPI command (SPCOM) register to enable the serial peripheral interface to prepare the data for transmission and wait for the $\overline{\text{SPISEL}}$ pin to be asserted. Data is shifted out from the slave on the SPIMISO pin and shifted in through the SPIMOSI pin.

The serial peripheral interface sets the E bit of the SPIER and issues a maskable interrupt to the system interface unit interrupt controller whenever its transmit buffer is not full. The serial peripheral interface sets the F bit of the SPIER and issues a maskable interrupt to the system interface unit interrupt controller whenever its receive buffer has been filled with data. The serial peripheral interface then continues receiving data until the $\overline{\text{SPISEL}}$ pin is negated. Transmission continues until no more data is available or the $\overline{\text{SPISEL}}$ pin is negated. After completing the transmission of characters, the serial peripheral interface transmits ones until the $\overline{\text{SPISEL}}$ pin is negated.

16.3.2.3.3 Multimaster Operation. The serial peripheral interface 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 serial peripheral interfaces are connected together and the $\overline{\text{SPISEL}}$ input pins are connected separately. In this environment, only one SPI device can work as a master at a time and all the others must be slaves. When the serial peripheral interface is configured as a master and its $\overline{\text{SPISEL}}$ input pin goes active, a multimaster error occurs because more than one SPI device is acting as bus master.

The serial peripheral interface sets both ME bits in the SPIER and SPIRD registers. The serial peripheral interface operation and output drivers are also disabled. The core should clear the EN bit of the SPI mode (SPMODE) register before using the serial peripheral interface again. After all the issues are addressed, the ME bit should be cleared and the serial peripheral interface enabled following the same procedure that is used after a reset.

16.3.2.4 PROGRAMMING THE SERIAL PERIPHERAL INTERFACE.

16.3.2.4.1 SPI Mode Register. The read/write SPI mode (SPMODE) register controls both the SPI operation mode and clock source. This register is cleared by reset.

SPMODE

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	LOOP	CI	CP	DIV16	RES	M/S	EN	CL	RESERVED			PM			

Bits 0 and 5—Reserved

These bits are reserved and should be set to zero.

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 serial peripheral interface is in loopback mode.

CI—Clock Invert

This bit inverts the SPI clock polarity (refer to Figures 4-3 and 4-4 for details).

0 = The inactive state of SPICLK is low.
1 = The inactive state of SPICLK is high.

CP—Clock Phase

This bit selects one of two fundamentally different transfer formats.

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

This bit selects the clock source for the SPI baud rate generator when configured as an SPI master. In slave mode, the clock source is the SPICLK pin.

0 = Use the baud rate generator clock as the input to the SPI baud rate generator.
1 = Use the BRGCLK/16 as the input to the SPI baud rate generator.

M/S—Master/Slave

This bit configures the serial peripheral interface to operate as a master or slave.

0 = The serial peripheral interface is a slave.
1 = The serial peripheral interface is a master.

EN—Enable Serial Peripheral Interface

This bit enables serial peripheral interface operation. The SPIMOSI, SPIMISO, SPICLK, and $\overline{\text{SPISEL}}$ pins should be configured to connect to the serial peripheral interface. When the EN bit is cleared, the serial peripheral interface is in a reset state and consumes minimal power, thus the SPI baud rate generator is not functioning and the input clock is disabled.

0 = The serial peripheral interface is disabled.
1 = The serial peripheral interface is enabled.

NOTE

Other bits of the SPMODE register should not be modified while the EN bit is set.

CL—Character Length

This bit specifies character length.

0 = 8 bits long.
1 = 16 bits long.

Bits 9–11—Reserved

These bits are reserved and should be set to one.

PM—Prescale Modulus Select

These four bits specify the divide ratio of the prescale divider in the SPI clock generator. The baud rate generator clock is divided by 4 * ([PM0–PM3] + 1), thus giving a clock divide ratio of 4 to 64 (64 to 1,024). The clock has a 50% duty cycle.

NOTE

The SPMODE register must be written according to its size. No byte-enable mechanism is available.

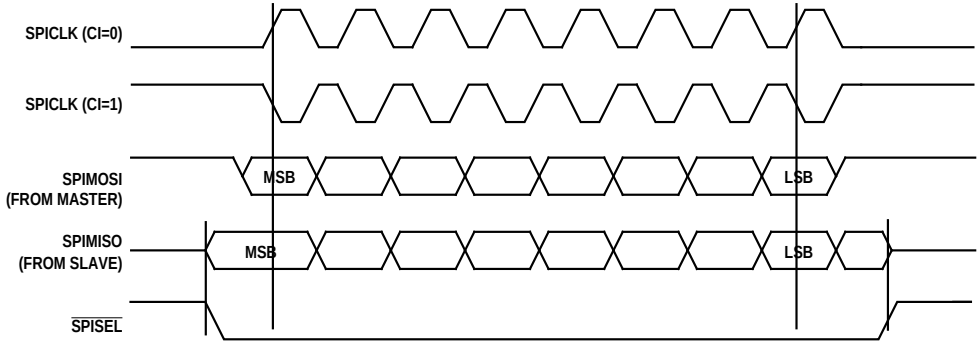


Figure 16-3. SPI Transfer Format with CP = 0

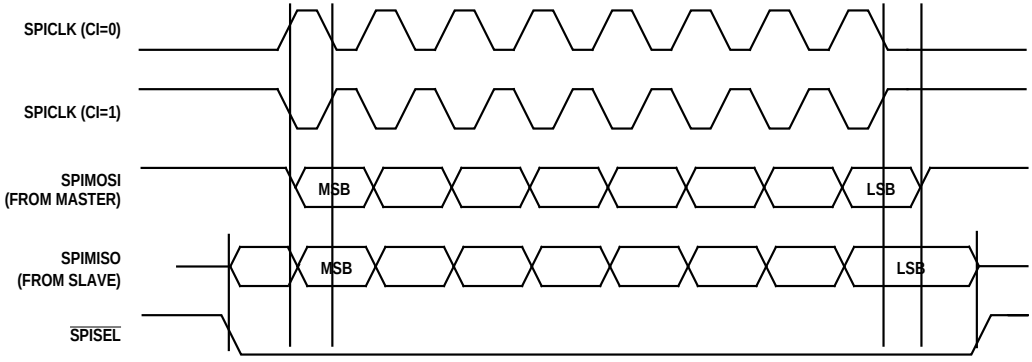


Figure 16-4. SPI Transfer Format with CP = 1

16.3.2.4.2 SPI Command Register. The 8-bit read/write SPI command (SPCOM) register is used to start a serial peripheral interface operation.

SPCOM

BIT	0	1	2	3	4	5	6	7
FIELD	STR	RESERVED						

STR—Start Transmit

When the serial peripheral interface is configured as a master, setting this bit causes the SPI controller to start transmitting or receiving data to or from the serial peripheral interface. When the serial peripheral interface is configured as a slave, setting the STR bit when the SPI is idle or between transfers causes it to load the shift register from the transmit data register and start transmitting as soon as the next SPI input clocks and select signal are received. The STR bit is automatically cleared after one system clock cycle.

Bits 1–7—Reserved

These bits are reserved and should be set to zero.

16.3.2.4.3 SPI Transmit Data Hold Register. The 32-bit read/write SPI transmit data hold (SPITD) register holds the character to be transmitted and F or L indications to the SPI channel. The core should set the F bit and prepare the first character of data before activating the SPI channel. Each time the E bit in the SPIER is set, the core can write another character of data to the SPITD register if there is no error indication in the SPIER. At the end of the frame the core should set the L bit and prepare the last character of data.

SPITD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED														L	F
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	DATA															

Bits 0–13—Reserved

These bits are reserved and should be set to 0.

L—Last

This bit represents the last character.

0 = This character is not the last character of the frame.
1 = This character is the last character of the frame.

F—First

This bit represents the first character.

0 = This character is not the first character of the frame.
1 = This character is the first character of the frame.

DATA—Transmit Data

This bit represents a character for transmission.

NOTE

Writing to the SPITD register clears the E bit from the SPIER.

16.3.2.4.4 SPI Receive Data Hold Register. The 32-bit read-only SPI receive data hold (SPIRD) register is used to receive a character of data or L and OV indications from the SPI channel. Each time the F bit in the SPIER is set, the core can read the SPIRD register.

SPIRD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	V	RESERVED												L	OV	ME
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	DATA															

V—Valid

This bit represents valid data.

0 = The data is invalid. Only the last indication is valid.
1 = The data is valid.

Bits 1–12—Reserved

These bits are reserved and should be set to 0.

L—Last

This bit represents the last character.

0 = This is not the last character in the frame.
1 = This is the last character in the frame.

OV—Overrun

This bit indicates that a receiver overrun has occurred during reception.

ME—Multi-Master Error

This bit indicates that the $\overline{\text{SPISEL}}$ pin was asserted while the serial peripheral interface was operating as a master. It indicates that there is a synchronization problem between multiple masters on the SPI bus.

DATA—Receive Data

This bit represents a character of received data.

16.3.2.4.5 SPI Event Register. The 8-bit read/write SPI event register (SPIER) is used to generate interrupts and report events recognized by the serial peripheral interface. When an event is recognized, the serial peripheral interface sets the corresponding bit in the SPIER. Interrupts are only generated by this register for E or F events and can be masked in the SPIMR. A bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. This register resets to $01.

SPIER

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED				UN	ME	F	E

Bits 0–3—Reserved

These bits are reserved and should be set to zero.

UN—Underrun

This bit indicates that the serial peripheral interface has encountered a transmitter underrun condition while transmitting the associated data buffer. This error condition is only valid when the serial peripheral interface is configured as a slave.

ME—Multimaster Error

This bit indicates that the $\overline{\text{SPISEL}}$ pin was asserted while the serial peripheral interface was operating as a master. This indicates that there is a synchronization problem between multiple masters on the SPI bus.

F—Full

Since the SPIRD register is the bottom of the receive FIFO, an empty SPIRD indicates the receive FIFO is empty.

0 = The SPIRD register is empty.
1 = The SPIRD register has been filled with received data and indications about V, L, and OV. The core is free to read the content of the receive data hold register. The F bit should be cleared to enable the reception of another character.

E—Empty

Since the SPITD register is the top of the transmit FIFO, a full SPITD indicates that the transmit FIFO is full.

0 = The SPITD register is full.
1 = The SPITD register is empty. The core is free to write to the transmit data hold register. The E bit should be cleared to enable the transmission of another character (writing to the SPITD register clears the E bit of the SPIER).

16.3.2.4.6 SPI Mask & Interrupt Level Register. The 8-bit read/write SPI mask & interrupt level register (SPIMR) contains the mask for the F and E bits in the SPIER and the priority request level of the interrupt. If the mask bit in the SPIMR is 1, the corresponding interrupt in the SPIER is enabled. If the proper mask bit is zero, the corresponding interrupt in the SPIER is masked. This register is cleared at reset.

SPIMR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			SPIRL			F	E

Bits 0–3—Reserved

These bits are reserved and should be set to zero.

SPIRL—SPI Interrupt Request Level

This field contains the priority request level of the interrupt request level 0-7.

F—Full Mask

If this bit is set to 1, the corresponding interrupt in the SPIER register is enabled. If it is zero, the corresponding interrupt is masked.

E—Empty Mask

If this bit is set to 1, the corresponding interrupt in the SPIER register is enabled. If it is zero, the corresponding interrupt is masked.

16.3.3 The I²C Controller

The inter-integrated circuit (I^2C) controller allows the MPC801 to exchange data with other I^2C devices such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The I^2C controller is a synchronous, multimaster bus that is used to connect several integrated circuits on a board. It uses two wires—serial data and serial clock—to carry information between the integrated circuits connected to the bus.

The I^2C 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^2C controller baud rate generator in master mode and generated externally in slave mode. The MPC801 I^2C most-significant bit is shifted out first. When the I^2C is not enabled in the I2MOD register, it consumes minimal power. Figure 4-5 illustrates the I^2C controller block diagram.

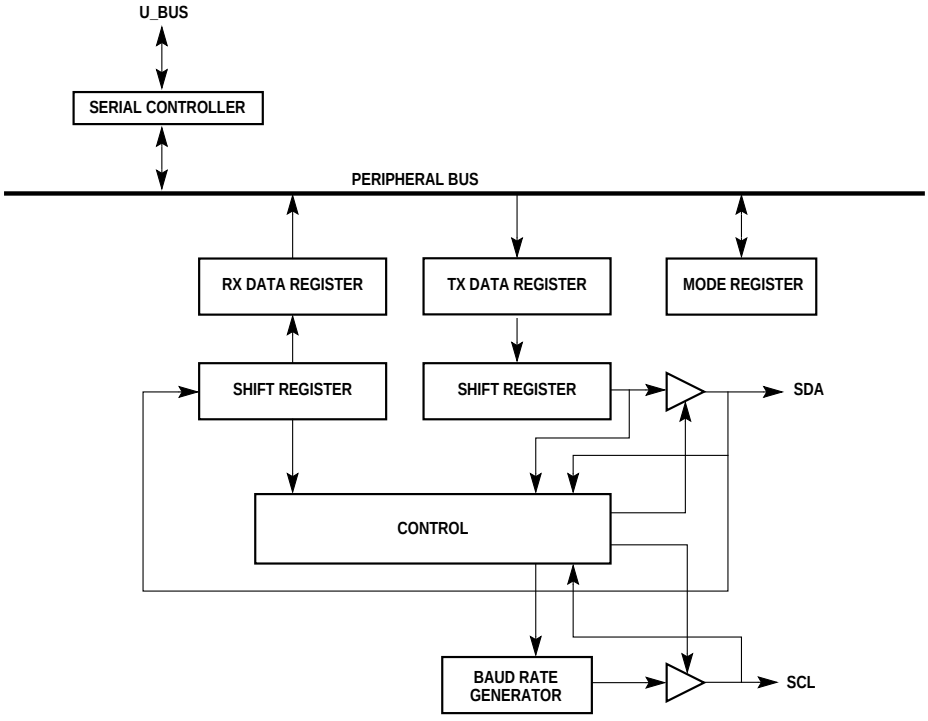


Figure 16-5. I²C Controller Block Diagram

NOTE

The I²C receiver and transmitter FIFOs are two characters deep. Combined with the shift register, this creates an effective FIFO depth of three characters.

16.3.3.1 FEATURES. The following is a list of the I²C controller's main features:

- Two-Wire Interface
- Full-Duplex Operation
- Master or Slave I²C Mode Support
- Multimaster Environment Support
- Clock Rate Support at a Maximum of 520KHz, Assuming a 25MHz System Clock
- Independent Programmable Baud Rate Generator
- Open-Drain Output Pins Support Multimaster Configuration
- Local Loopback Capability for Testing

16.3.3.2 CLOCKING AND PIN FUNCTIONS . The I^2C controller can be configured as a master for the serial channel to generate the clock signal and initiate and terminate the transfer. As a slave, it can be configured so that the clock signal is an input to the I^2C controller. When the I^2C controller is a master, the I^2C baud rate generator is used to generate the transmit and receive clocks. The I^2C baud rate generator takes its input from the baud rate generator clock, which is generated in the clock synthesizer of the MPC801.

The bidirectional serial data (SDA) and serial clock (SCL) pins are connected to a positive supply voltage via an external pull-up resistor. When the bus is free, both lines are high. When the I^2C controller is a master, the SCL pin is the clock output signal that shifts in the received data and shifts out the transmitted data from or to the SDA pin. Additionally, the transmitter arbitrates for the bus during the transmission and aborts it if it loses arbitration. When the I^2C controller is a slave, the SCL pin is the clock input signal that shifts in the received data and shifts out the transmitted data from or to the SDA pin.

16.3.3.3 I^2C CONTROLLER TRANSMISSION AND RECEPTION PROCESS. Since the I^2C is a character-oriented communication unit, the core is responsible for packing and unpacking the receive/transmit frames. The core supplies and collects words to or from the I^2C as requested. The core receives data by reading the I2CRD register. It resets the F bit in the I2CER to free the I2CRD register for the next operation. The core transmits data by writing it into the I2CTD register when a FIRST or LAST word indication occurs.

The I^2C core handshake protocol can be implemented using a polling or interrupt mechanism. With a polling mechanism, the core reads the I2CER in a predefined frequency and acts according to the value of the I2CER bits. The polling frequency depends on the SPI serial channel frequency. When an interrupt mechanism is used, setting either the E or F bits of the I2CER causes an interrupt to the core. The interrupt level is determined by the I2CRL bits of the I2CMR.The core then reads the I2CER and acts appropriately. There are two basic modes of I^2C operation—master and slave mode.

16.3.3.3.1 I^2C Master Mode. When the I^2C controller functions in master mode, the I^2C master initiates a transaction by transmitting a message to the peripheral or I^2C slave. This message specifies a read or write operation. If it is a read operation, the direction of the transfer is changed at the moment of the first acknowledgment and the slave receiver becomes a slave transmitter. To begin the data exchange, the core writes the data to be transmitted into the I2CTD register and sets the STR and M/S bits in the I2COM register to start transmitting. The I^2C master begins transmitting once the shift register is loaded with data and the I^2C bus is not busy.

The I^2C controller generates a start condition on the SDA and SCL pins and, at the same time, a programmable clock pulses on the SCL pin for each data bit shifted out on the SDA pin. For each bit shifted out, the transmitter monitors the level of the SDA pin to detect a possible collision with other I^2C master transmitters. 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 slave mode. A maskable interrupt is generated to the core to enable the software to try retransmitting later. After each byte, the master transmitter monitors the acknowledgment indication. If the receiver fails to acknowledge a byte, the transmission is aborted and the stop condition is generated by the master.

The I²C sets the E bit of the I2CER and issues a maskable interrupt to the system interface unit interrupt controller whenever its transmit buffer is not full. The I²C also sets the E bit after sending the last word. In response, the core should read the exception flags that relate to the last word. When a master I²C read operation is to be performed, the core should supply the I²C with a first byte that contains the slave's address with the R/$\overline{W}$ bit set. The master I²C determines the length of the incoming massage. The core continues to supply data to the I²C master's transmitter, but the following data bytes are ignored by the I²C controller and the transmission procedure is performed without actual transmission occurring. That is, until it encounters a byte containing the L bit. When this byte is encountered, the transmission procedure is performed and the reception process ends.

The receiver starts reception after it finds the start condition on the SDA and SCL pins.The I²C notifies the core by setting the F bit in the I2CER.The core is responsible for emptying the buffer for the next data. This process continues until a new start or stop condition is detected. If a mismatch is found, the receiver stops receiving and searches again for a new start condition. The receiver acknowledges each data byte as long as an overrun condition does not occur.

16.3.3.3.2 I²C Slave Mode. When the I²C controller is in slave mode, it receives messages from an I²C master and replies to them. Once a slave mode operation is selected in the I2MOD register, the SCL pin becomes an input from the master to the slave. The SCL pin may vary from DC to the BRGCLK/48 (520KHz for a 25MHz system). Before data is exchanged, the core writes the data to be transmitted into the I2CTD register. The core then clears the M/S bits and sets the STR bit in the I2COM register to enable the I²C controller to prepare the data for transmission and wait for a read request from the master.

When a match is detected between the first byte received after a start condition and the slave address, the R/$\overline{W}$ bit is checked. If a write operation was requested by the master (R/$\overline{W}$ =0), the data received is acknowledged and written into a receive FIFO. A maskable interrupt is issued when a character is fully received. This process continues until the last character has been sent or an error has occurred. The PowerPC master is responsible for emptying the buffer to ensure smooth operation.

If a read operation was requested by the I²C master (R/$\overline{W}$=1), the recently received address byte is only acknowledged if the transmitter FIFO has been loaded by the core. If the transmitter is ready, transmission starts on the next clock pulse after the acknowledgment. Otherwise, the transaction is aborted and the UN bit in the I2CER is set to notify the software to prepare data for transmission on the next attempt. If an underrun condition occurred, the slave transmits ones until a stop condition is detected.

After each byte, the transmitter checks the acknowledge bit. If the master receiver fails to acknowledge a byte transmission is aborted and the NAK bit in the I2CER is set. Data is shifted out from the slave on the SDA pin.

16.3.3.4 PROGRAMMING THE I²C CONTROLLER

16.3.3.4.1 I²C Mode Register. The read/write I²C mode (I2MOD) register controls I²C operation mode and the I²C clock source. It is cleared by reset.

I2MOD

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			GCD	FLT	PDIV		EN

Bit 0–2—Reserved

These bits are reserved and should be set to zero.

GCD—General Call Disable

This bit determines if the receiver acknowledges a general call address.

0 = General call address is enabled.
1 = General call address is disabled.

FLT—Clock Filter

This bit determines if the I²C input clock is filtered to prevent spikes in a noisy environment.

0 = I2CLK is not filtered.
1 = I2CLK is filtered by a digital filter.

PDIV—Pre Divider

This bit determines the division factor of the clock before it is fed into the baud rate generator. The clock source for the I²C is the baud rate generator clock that is generated by the MPC801 clock synthesizer.

00 = Use the BRGCLK/32 as the input to the I²C baud rate generator.
01 = Use the BRGCLK/16 as the input to the I²C baud rate generator.
10 = Use the BRGCLK/8 as the input to the I²C baud rate generator.
11 = Use the BRGCLK/4 as the input to the I²C baud rate generator.

EN—Enable I²C

This bit enables I²C operation. When it is cleared, the I²C is in a reset state and consumes minimal power. The I²C baud rate generator is not functioning and the input clock is disabled. Other bits of the I2MOD register should not be modified while the EN bit is set.

0 = I²C is disabled.
1 = I²C is enabled.

16.3.3.4.2 I²C Address Register. The 8-bit, memory-mapped, read/write I²C address (I2ADD) register holds the address for this I²C port.

I2ADD

BIT	0	1	2	3	4	5	6	7
FIELD	SAD							RESERVED

SAD—Slave Address

This field holds the slave address for the I²C port.

Bit 7—Reserved

This bit is reserved and should be set to 0.

16.3.3.4.3 I²C BRG Register. The 8-bit, memory-mapped, read/write I²C BRG (I2BRG) register sets the divide ratio of the baud rate generator. This register is set to all ones at hard reset.

I2BRG

BIT	0	1	2	3	4	5	6	7
FIELD	DIV							

DIV—Division Ratio

These bits specify the divide ratio of the baud rate generator divider in the I²C clock generator. The output of the prescaler is divided by 2 * ([DIV0–DIV7] + 3 + 2*FLT). The clock has a 50% duty cycle. The minimum value for each DIV bit is 3 if the digital filter is disabled and 6 if it is enabled.

16.3.3.4.4 I²C Command Register. The 8-bit read/write I²C command (I2COM) register is used to start an I²C operation.

I2COM

BIT	0	1	2	3	4	5	6	7
FIELD	STR	RESERVED						M/S

STR—Start Transmit

When the I²C is configured as a master, setting this bit to 1 causes the I²C controller to start the transmission of data from the I²C transmit buffers if you have configured them as ready. When the I²C is configured as a slave, setting the STR bit to 1 when the I²C is idle or between transfers causes the I²C to load the shift register from the I²C transmit buffer and start transmitting when an address byte is received that matches the slave address, with the R/$\overline{W}$ bit is set. The STR bit is always read as a zero.

Bits 1–6—Reserved

These bits are reserved and should be set to zero.

M/S—Master/Slave

This bit configures the I^2C to operate as a master or slave.

0 = I^2C is a slave.
1 = I^2C is a master.

16.3.3.4.5 I^2C Transmit Data Hold Register. The 16-bit read/write I^2C transmit data hold (I2CTD) register holds the character to be transmitted and the F or L indications to the I^2C channel. The core should set the F bit and prepare the first character of data before activating the I^2C channel. Each time the E bit in the I2CER is set the core can write another character of data to the I2CTD register, as long as there is no error indication in the I2CER. At the end of the frame, the core should set the L bit.

I2CTD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED						L	F	DATA							

Bits 0–5—Reserved

These bits are reserved and should be set to 0.

L—Last

This bit represents the last character.

0 = This character is not the last character of the frame.
1 = This character is the last character of the frame.

F—First

This bit represents the first character.

0 = This character is not the first character of the frame.
1 = This character is the first character of the frame.

DATA—Transmit Data

This bit represents a character for transmission.

NOTE

Writing to the I2CTD register clears the E bit in the I2CER.

16.3.3.4.6 I²C Receive Data Hold Register. The 16-bit read-only I²C receive data hold (I2CRD) register is used to receive a character of data and indications of the L and OV bits from the I²C channel. Each time the F bit in the I2CER is set, the core can read this register.

I2CRD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED						L	OV	DATA							

Bits 0–5—Reserved

These bits are reserved and should be set to 0.

V—Valid

This bit represents valid data.

0 = The data is invalid. Only the last indication is valid.
1 = The data is valid.

L—Last

This bit represents the last character in the frame.

0 = This is not the last character in the frame.
1 = This is the last character in the frame.

OV—Overrun

This bit indicates that a receiver overrun has occurred during reception.

DATA—Receive Data

This bit represents a character of received data.

16.3.3.4.7 I²C Event Register. The 8-bit read-only I²C event register (I2CER) is used to generate interrupts and report events recognized by the I²C channel. When an event is recognized, the I²C controller sets its corresponding bit in the I2CER. An interrupt can be generated by the F or E bits and can be masked by the I2CMR. A bit is cleared by writing a 1 (writing a zero has no effect). More than one bit can be cleared at a time. This register resets to $01.

I2CER

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			NAK	UN	CL	F	E

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

NAK—No Acknowledge

This bit indicates that a transmission was aborted because the last byte transmitted was not acknowledged.

UN—Underrun

This bit indicates that the I^2C has encountered a transmitter underrun condition while transmitting the associated data buffer.

CL—Collision

This bit indicates that a transmission was aborted because the transmitter lost arbitration for the bus.

F—Full

Since the I2CRD register is the bottom of the receive FIFO, an empty I2CRD indicates that the receive FIFO is empty.

0 = The I2CRD register is empty.
1 = The I2CRD register has been filled with received data and indications about the V, L, and OV bits. The core is free to read the contents of the I2CRD register. However, the F bit should be cleared to receive another character.

E—Empty

Since the I2CTD register is the top of the transmit FIFO, a full I2CTD indicates that the transmit FIFO is full.

0 = The I2CTD register is full.
1 = The I2CTD register is empty. The core is free to write to the I2CTD register. However, the E bit should be cleared to receive another character. The I2CTD register write clears the E bit of the I2CER.

16.3.3.4.8 I^2C Mask & Interrupt Level Register. The 8-bit read/write I^2C mask & interrupt level register (I2CMR) contains the mask for the F and E bits in the I2CER and the priority request level of the interrupt. If the proper mask bit in the I2CMR is 1, the corresponding interrupt in the I2CER is enabled, but if it is zero, the interrupt is masked. This register is cleared at reset.

I2CMR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			I2CRL			F	E

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

I2CRL—I^2C Interrupt Request Level

This bit contains the priority request level of the interrupt sent from the I^2C controller to the interrupt request level 0-7.

F—Full Mask

If this bit is set to 1, the corresponding interrupt in the I2CER is enabled. If it is zero, the interrupt is masked.

E—Empty Mask

If this bit is set to 1, the corresponding interrupt in the I2CER is enabled. If it is zero, the interrupt is masked.

16.4 THE PARALLEL I/O PORT

The MPC801 supports a general-purpose I/O port. Each pin in the I/O port can be configured as a general-purpose I/O pin or a dedicated peripheral interface pin. Port B is shared with the serial peripheral interface, I^2C, and the UART1 and UART2 pins. Each pin can be configured as an input or output and has a latch for data output. They can be read or written to at any time. Port B pins can be configured as open-drain (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.

16.4.1 Features

The following is a list of the parallel I/O port's main features:

- 16 Bits Wide
- Open-Drain Capability
- All Pins Are Bidirectional, Have Alternate On-Chip Peripheral Functions, and Three-Stated at System Reset.
- Pin Values are Readable While the Pin Is Connected to an On-Chip Peripheral

16.4.2 Port B Pin Functions

Port B pins are independently configured as general-purpose I/O pins if the corresponding bit in the port B pin assignment register (PBPAR) is cleared. They are configured as dedicated on-chip peripheral pins if the corresponding PBPAR and PBDIR bits are set. When acting as a general-purpose I/O pin, the signal direction for that pin is determined by the corresponding control bit in the port B data direction (PBDIR) register. The port I/O pin is configured as an input if the corresponding PBDIR bit is cleared or as an output if the corresponding PBDIR bit is set. All PBPAR and PBDIR bits are cleared at the time of total system reset and all port B pins are configured 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 port B data (PBDAT) register. 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 the PBDAT register is read, the port pin itself is read. If a port B pin is configured as an input, data written to the PBDAT register is still stored in the output latch, but is prevented from reaching the port pin. In this case, when the PBDAT register is read, the state of the port pin is read. All of the port B pins have more than one option, including on-chip peripheral functions related to the serial peripheral interface, I^2C, UART1, and UART2. Table 4-2 provides a description of the available port B pin options.

Table 16-2. Port B Pin Assignment

PIN FUNCTION			
SIGNAL	**PBPAR = 0**	**PBPAR=1 & PBDIR=1**	**INPUT TO ON-CHIP PERIPHERAL**
PB31	PORT B31	$\overline{\text{SPISEL}}$	VDD
PB30	PORT B30	SPICLK	GND
PB29	PORT B29	SPIMOSI	VDD
PB28	PORT B28	SPIMISO	SPIMISO = SPIMOSI
PB27	PORT B27	I2CSDA	VDD
PB26	PORT B26	I2CSCL	GND
PB25	PORT B25	UGPIO1	VDD
PB24	PORT B24	UGPIO2	VDD
PB23	PORT B23	$\overline{\text{UCTS1}}$	GND
PB22	PORT B22	$\overline{\text{UCTS2}}$	GND
PB21	PORT B21	URXD1	VDD
PB20	PORT B20	URXD2	VDD
PB19	PORT B19	$\overline{\text{URTS1}}$	—
PB18	PORT B18	$\overline{\text{URTS2}}$	—
PB17	PORT B17	UTXD1	—
PB16	PORT B16	UTXD2	—

NOTE

Setting the PBPAR bit without setting the corresponding PBDIR bit causes a programming error.

16.4.3 The Port B Registers

Port B has four 16-bit memory-mapped read/write control registers that must all be written according to their size. No byte-enable mechanism is available.

16.4.3.1 PORT B OPEN-DRAIN REGISTER. The port B open-drain (PBODR) register indicates a normal or wired-OR configuration of the port pins. This register is cleared at system reset.

PBODR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	OD16	OD17	OD18	OD19	OD20	OD21	OD22	OD23	OD24	OD25	OD26	OD27	OD28	OD29	OD30	OD31
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

OD16–OD31—Driver Type

Each pin in port B has a dedicated control bit that selects the driver type. It must decide between an active or open-drain driver.

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.

16.4.3.2 PORT B DATA REGISTER. A read of the port B data (PBDAT) register returns the data to the pin, regardless of whether the pin is defined as an input or output. This allows the pin to detect output conflicts by comparing the written data with the data on the pin. A write to the PBDAT register is latched and if that bit in the PBDIR register is configured as an output, the value latched for that bit is driven to its respective pin. This register can be read or written to at any time. It is not initialized, but is undefined at reset.

PBDAT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

D16–D31—Data

Each bit in the register reflects the pin value, regardless of whether the pin is an input or output.

16.4.3.3 PORT B DATA DIRECTION REGISTER. The port B data direction (PBDIR) register is cleared at system reset.

PBDIR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	DR16	DR17	DR18	DR19	DR20	DR21	DR22	DR23	DR24	DR25	DR26	DR27	DR28	DR29	DR30	DR31
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

DR16–DR31—Data Direction

Each pin can be defined as an input or output pin.

0 = The corresponding pin is an input.
1 = The corresponding pin is an output.

16.4.3.4 PORT B PIN ASSIGNMENT REGISTER. The port B pin assignment register (PBPAR) is cleared at system reset.

PBPAR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	DD16	DD17	DD18	DD19	DD20	DD21	DD22	DD23	DD24	DD25	DD26	DD27	DD28	DD29	DD30	DD31
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

DD16–DD31—

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 and the on-chip peripheral function to which it is dedicated can be determined by other bits.

SECTION 17
DATA ALIGNMENT

For referencing purposes, this section serves as a placeholder to keep the sequence of other sections the same as they appear in other manuals.

SECTION 18
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. The following features allow you to efficiently debug systems based on the MPC801.

- Program flow tracking
- Internal watchpoint and breakpoint generation
- Emulation systems that control core (debug mode) activity

18.1 PROGRAM FLOW TRACKING

The MPC801 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 MPC801 is programmed to operate in serialized mode and show all fetch cycles on the external bus. When working in this mode, although tracking of the program instruction flow is simpler, the performance of the MPC801 is much lower than when working in regular mode. For more details about programming the core to operate in this mode, see Table 18-18.

The MPC801 implements a prefetch queue combined with parallel, out of order, and pipelined execution. These features, plus the fact that most fetch cycles are performed internally from the instruction cache, increases performance but makes it very difficult to provide 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 executing with no exception. Therefore, only retired instructions can be considered *a*rchitecturally executed.

To reconstruct program trace, the program code combined with additional MPC801 information is required. Reporting program trace during retirement significantly complicates visibility and increases the die size for two reasons—more than one instruction can retire in a clock cycle and it is harder to report on indirect branches during retirement. Because of this, program trace is reported during fetch and helps to reconstruct the instructions that actually retire after fetch canceled instructions are reported. Instructions are fetched sequentially until branches (direct or indirect), exceptions, or interrupts appear in the program flow or until a stall in execution forces the machine to avoid fetching the next address. These instructions may be architecturally executed or they may be canceled in any 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.

18.1.1 Basic Operation

18.1.1.1 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, you must ensure that these cycles are visible on the external bus.

The VSYNC signal, when asserted, forces all fetch cycles marked with the program trace cycle attribute to be visible on the external bus, even if their data is found in one of the internal devices. To enable the external hardware to properly synchronize with the internal activity of the core, 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 18.1.1.5 The External Hardware**.

NOTE

To keep the pin count of the chip as low as possible, the VSYNC signal is not implemented as one of the chip's external pins. Instead, it is asserted and negated using the serial interface implemented in the development port. For more information on this interface, refer to **Section 18.3.3 The Development Port**. Forcing the core to show all fetch cycles marked with the program trace cycle attribute can be accomplished by either asserting the VSYNC signal (as mentioned above) or by programming the fetch show cycle bits in the instruction support control register (ICTRL). For more information refer to **Section 18.1.2 Controlling the Instruction Fetch Show Cycle**.

When the VSYNC signal is asserted, all fetch cycles marked with the program trace cycle attribute become visible on the external bus. These cycles generate regular bus cycles when the instructions reside in one of the external devices or generate address-only cycles when the instructions are in one of the internal devices. When 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, perform these additional external bus cycles, and maintain the same behavior when VSYNC is asserted and negated. For more information see Table 18-18.

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 18-1 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 18.3.2 Debug Mode**.

Table 18-1. VF Pin 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 18.1.1.4 Sequential Instructions Marked As Indirect Branch**.
2. Unless the next clock VF = 111, refer to **Section 18.1.1.2 Special Case of Queue Flush Information**.

18.1.1.2 SPECIAL CASE OF QUEUE FLUSH INFORMATION

There is one special case where the queue flush information is expected on the VF pins. This is easily monitored since the only case where this can happen is when VF =111 and the maximum number of possible queue flushes is five.

18.1.1.3 PROGRAM TRACE 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 externally visible. When the core is in debug mode, the VF pins equal '000' and the VFLS pins equal '11'. For more information on the MPC801 debug mode, refer to **Section 18.3 Development System Interface**.

If the VSYNC signal is asserted or negated while the CPU is in debug mode, this information is announced 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 being issued. In both cases, the first instruction fetch after debug mode is marked with the program trace cycle attribute and is externally visible.

18.1.1.4 SEQUENTIAL INSTRUCTIONS MARKED AS INDIRECT BRANCH

There are instances in which nonbranch (sequential) instructions affect the machine like indirect branch instructions affect it. These instructions include the **rfi**, **mtmsr**, **isync**, and **mtspr** instructions to the CMPA-F, ICTRL, ICR, and DER registers.

The core marks these instructions as indirect branch instructions (VF = '101') and the following instruction address is marked with the program trace cycle attribute, as if it was an indirect branch target. Therefore, when one of these special instructions is detected in the core, the address of the following instruction is externally visible. The reconstructing software is now able to correctly evaluate the effect of these instructions.

18.1.1.5 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 the most common methods.

18.1.1.5.1 Back Trace. This is useful when a record of the program trace is needed before an event like system failure occurs. If back trace is required, the external hardware should start sampling the VF and VFLS pins and the addresses of all cycles marked with the program trace cycle attribute immediately after reset is negated. Since the instruction show cycles programming defaults to "show all" out of reset, all cycles marked with the program trace cycle attribute are visible on the external bus. VSYNC should be asserted sometime after reset and negated when the actual event occurs. If "show all" is not preferable for the instruction show cycles prior to the occurrence of the actual event, 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 cyclical buffers. After the VSYNC signal is negated, the trace buffer contains the program flow trace of the program that was executed before the event in question occurred.

18.1.1.5.2 Window Trace. This is useful when a record of the program trace between two events is required. The VSYNC signal should be asserted between these two events. After VSYNC is negated, the trace buffer will contain information describing the program trace of the program that was executed between the two events.

18.1.1.5.3 Synchronizing the Trace Window to the Internal Core Events. The assertion and negation of the VSYNC signal is accomplished using the serial interface implemented in the development port. To synchronize the assertion and negation of VSYNC to an internal core event, the internal breakpoints hardware should be used with the debug mode. This method is available only when debug mode is enabled. For more information on debug mode, refer to **Section 18.3 Development System Interface**.

To synchronize the trace window to the internal core events, follow these steps:

1. Enter debug mode either immediately out of reset of using the debug mode request.
2. Using the control registers defined in **Section 18.2 Watchpoint And Breakpoint Generation**, program the hardware to break on the event that marks the start of the trace window.
3. Enable debug mode entry for the programmed breakpoint in the debug enable register (see Table 18-18).
4. Return to the regular code run.
5. The hardware generates a breakpoint when the event in question is detected and the machine enters debug mode.
6. Program the hardware to break on the event that marks the end of the trace window.
7. Assert the VSYNC signal.
8. Return to the regular code run. The first report on the VF pins is VSYNC, where VF =’011’.
9. The external hardware starts sampling the program trace information after the VF pins indicate VSYNC.
10. The hardware generates a breakpoint when the event in question is detected and the machine enters debug mode.
11. Negate VSYNC.
12. Return to the regular code run (issue an **rfi**). The first encoding on the VF pins is VSYNC, where VF =’011’.
13. The external hardware stops sampling the program trace information after recognizing VSYNC on the VF pins.

18.1.1.5.4 Detecting the Trace Window Start Address. When using back trace, latching VF, VFLS, and the address of the cycles marked program trace cycle should all start immediately after reset is negated. The start address is the first address in the program trace cycle buffer. When using window trace, latching VF, VFLS, and the address of the cycles marked as program trace cycle should all start immediately after the first VSYNC signal 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. The following table should be used to calculate the trace window start address.

Table 18-2. Detecting the Trace Buffer Starting Point

VF1	VF2	STARTING POINT	DESCRIPTION
011 VSYNC	001 Sequential	T1	VSYNC asserted. Followed by a sequential instruction. The start address is T1.
011 VSYNC	110 Branch Direct Taken	T1 - 4 + Offset(T1 - 4)	VSYNC asserted. Followed by a taken direct branch. The Start address is the target of the direct branch.
011 VSYNC	101 Branch Indirect Taken	T2	VSYNC asserted. Followed by a taken indirect branch. The start address is the target of the indirect branch.

18.1.1.5.5 Detecting VSYNC Assertion and Negation. 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’.

18.1.1.5.6 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 arbitrates over the external bus. Therefore, there is a delay between the time a cycle marked as program trace cycle is performed and the actual time that the cycle is detected on the external bus.

When you negate VSYNC using the serial interface of the development port, the core delays reporting that VSYNC occurred on the VF pins until all addresses marked with the program trace cycle attribute are externally visible. Therefore, the external hardware should stop sampling VF, VFLS, and the address of the cycles marked as program trace cycle immediately after VF = VSYNC. The last two instructions reported on the VF pins are not always valid. Therefore, at the last stage of the reconstruction software, the last two instructions should be ignored.

18.1.1.6 BENEFITS OF COMPRESSION

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.

18.1.2 Controlling the Instruction Fetch Show Cycle

Instruction fetch show cycles are controlled by the bits in the ICTRL register and the state of the VSYNC signal. The following table defines the level of fetch show cycles generated by the core. For information on the fetch show cycle control bits, see Table 18-3.

Table 18-3. Fetch Show Cycle Control

VSYNC	ISCTL (29:31) INSTRUCTION FETCH SHOW CYCLE CONTROL BITS	SHOW CYCLES GENERATED
X	000	All Fetch Cycles
X	X01	All Change of Flow (Direct and Indirect)
X	X10	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.

18.2 WATCHPOINT AND BREAKPOINT GENERATION

When they are detected, watchpoints are reported to the external world (on dedicated pins), but do not change the timing and flow of the machine. When breakpoints are detected, they force the machine to branch to the appropriate exception handler. The core supports watchpoints that are generated inside the core as well as breakpoints that are generated inside and outside the core.

Internal watchpoints are generated when a user-programmable set of conditions are met. Internal breakpoints can be programmed to be generated either when one of the internal watchpoints is asserted or after an internal watchpoint is asserted for user-programmable times. Programming a certain internal watchpoint to generate an internal breakpoint can be done either in the software, by setting the corresponding software trap enable bit, or on-the-fly using the serial interface of the development port to set the corresponding trap enable bit. External breakpoints can be generated by any of the system peripherals, including those found on or outside the MPC801 or those found by an external development system. Peripherals on the external bus use the serial interface of the development port to assert an external breakpoint.

In the core, as in other microprocessors, saving and restoring the machine state on the stack during exception handling is done in the software. When the software is in the middle of saving or restoring, the MSR_{RI} bit is cleared. Exceptions that occur are handled by the core when the MSR_{RI} bit is clear and they result in a nonrestartable machine state. For more information refer to **Section 6.2.4.1 Restartability After An Interrupt**.

In general, breakpoints are recognized in the core only when the MSR_{RI} bit is set, which guarantees machine restartability after a breakpoint. In this working mode, breakpoints are masked. There are times when it is preferable to enable breakpoints even when the MSR_{RI} bit is clear, even though there is the risk of causing a nonrestartable machine state. In programmable nonmasked mode, an external development system can choose to assert a nonmaskable external breakpoint. Watchpoints are not masked and are always reported on the external pins, regardless of the value of the MSR_{RI} bit. The counters, although they are counting watchpoints, are part of the internal breakpoints logic and are not decremented when the core is in masked mode and the MSR_{RI} bit is clear. Figure 18-1 illustrates the watchpoint and breakpoint support of the core.

18.2.1 Internal Watchpoints and Breakpoints

This section describes the internal breakpoints and watchpoints support of the core. For more information on external breakpoints support, refer to **Section 18.3 Development System Interface**. Internal breakpoint and watchpoint support is based on:

- Eight comparators that compare information on instruction and load/store cycles
- Two counters
- Two AND-OR logic structures

The comparators perform a comparison on the instruction address (I-address), load/store address (L-address), and load/store data (L-data). The comparators can detect the following conditions:

- Equal to
- Not equal to
- Greater than
- Less than

Greater-than-or-equal-to and less-than-or-equal-to are easily obtained from these four conditions. Refer to **Section 18.1.1.6 Benefits of Compression** for more information. Using the AND-OR logic structures, “in range” and “out of range” detections (on the address and data comparators) are supported. Using the counters, a breakpoint can be programmed to be recognized after an event is detected after a predefined number of times.

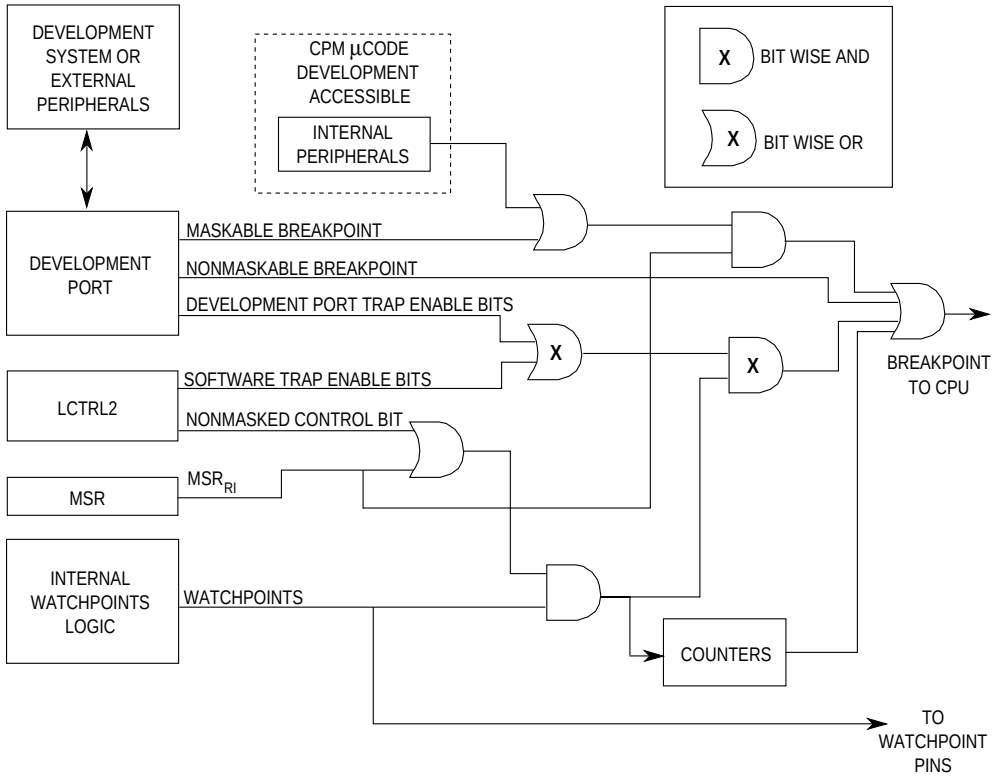


Figure 18-1. Watchpoints and Breakpoint Support

The L-data comparators operate on load or store fixed-point data. When operating on fixed-point data, the L-data comparators perform a comparison on bytes, half-words and words. They treat numbers as either signed or unsigned values. The comparators generate match events and then instruction match events enter the instruction AND-OR logic where the instruction watchpoints and breakpoint are generated. The asserted instruction watchpoints can generate the instruction breakpoint. Two different events can decrement one of the counters. When a counter on one of the instruction watchpoints expires, the instruction breakpoint is asserted.

The instruction watchpoints and load/store match events on the address/data comparators enter the load/store AND-OR logic where the load/store watchpoints and breakpoint are generated. 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 when they retire. Internal breakpoints progress in the machine until they reach the top of the history buffer when the machine branches to the breakpoint exception routine. So the breakpoint features can be used without restricting the software, the address of the load/store cycle that generated the load/store breakpoint is not stored in the data address register. In a load/store breakpoint, the address of the load/store cycle that generated the breakpoint is stored in the breakpoint address register.

18.2.1.1 FEATURES. The following list summarizes the main features of the internal watchpoints and breakpoints:

- Four I-Address Comparators Supporting Equal, Not Equal, Greater Than, and Less Than.
- Two L-Address Comparators Supporting Equal, Not Equal, Greater Than, and Less Than.
- Two L-Data Comparators Supporting Equal, Not Equal, Greater Than, and Less Than.
- No Internal Breakpoint/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 Detect In and Out of Range Conditions, Including Either Signed or Unsigned Values on the L-Data.
- A Programmable AND-OR Logic Structure Between the Four Instruction Comparators Results in Five Outputs, Four Instruction Watchpoints, and One Instruction Breakpoint.
- A Programmable AND-OR Logic Structure Between the Four Instruction Watchpoints and the Four Load/Store Comparators Results in Three Outputs, Two Load/Store Watchpoints, and One Load/Store Breakpoint.
- Five Watchpoint Pins—Three for the Instruction and Two for the Load/Store.
- Two Dedicated 16-bit Down Counters. Each Can be Programmed to Count Either an Instruction Watchpoint or a Load/Store Watchpoint. Only Architecturally Executed Events are Counted.
- On-the-Fly Trap Enable Programming of the Different Internal Breakpoints Using the Serial Interface of the Development Port. Software Control is Also Available.
- Watchpoints Do Not Change the Timing of the Machine.
- Internal Breakpoints and Watchpoints are Detected on the Instruction During Instruction Fetch.
- Internal Breakpoints and Watchpoints are Detected on the Load/Store During Load/Store Bus Cycles.
- Instruction and Load/Store Breakpoints and Watchpoints are Handled on Retirement and Then Reported.
- Breakpoints and Watchpoints on Recovered Instructions (As a Result of Exceptions, Interrupts, or Miss Prediction) Are Not Reported and Do Not Change the Timing of the Machine.

- Instructions with Instruction Breakpoints Are Not Executed. The Machine Branches To the Breakpoint Exception Routine Before it Executes the Instruction.
- Instructions with Load/Store Breakpoints are Executed. The Machine Branches To the Breakpoint Exception Routine After it Executes the Instruction. The Address of the Access is Placed In the Breakpoint Address Register.
- Load/Store Multiple and String Instructions with Load/Store Breakpoints First Finish Execution and Then the Machine Branches to the Breakpoint Exception Routine.
- Load/Store Data Compare is Made On the Load/Store, After Swap in Store Accesses and Before Swap in Load Accesses.
- Internal Breakpoints Operate in Masked or Nonmasked Mode.
- Both “go to x” and “continue” Working Modes are Supported for Instruction Breakpoints.

18.2.1.2 RESTRICTIONS. There are times when the same watchpoint can be detected more than once during the execution of a single instruction. For example, a load/store watchpoint detected on more than one transfer when executing load/store multiple/string or a load/store watchpoint detected on more than one byte when working in byte mode. In these cases, only one watchpoint of the same type is reported for a single instruction. Similarly, only one watchpoint of the same type can be counted in the counters for a single instruction. Since watchpoint events are reported when the instruction that caused the event retires ensuing events can be reported in the same clock. Moreover, if the same event is detected on more than one instruction can just be reported once. The internal counters count correctly in these cases.

18.2.1.3 BYTE AND HALF-WORD WORKING MODES. You can use watchpoints and breakpoints to detect matches on bytes and half-words when the byte/half-word is accessed in a load/store instruction of larger data widths. For example, when loading a table of bytes using a series of load word instructions. To use this feature in word mode, the required match value should be written to the correct half-word of the data comparator and to the mask in the L-data comparator. If you prefer to break on bytes, the byte mask for each L-comparator and the bytes to be matched must be written in the data comparator.

Since bytes and half-words can be accessed using a larger data width instruction, it is impossible for you to predict the exact value of the L-address lines when the requested byte/half-word is accessed. If the matched byte is byte 2 of the word and is accessed using a load word instruction, the L-address value will be of the word (byte 0). Therefore, the core masks the two least-significant bits of the L-address comparators whenever a word access is performed and the least-significant bits whenever a half-word access is performed. Address range is only supported when aligned according to the access size.

18.2.1.3.1 Byte Working Mode Example

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.

18.2.1.3.2 Half-Word Working Mode Example 1

Looking for:
Data size: Half-word.
Address: Greater than 0x00000000 and less than 0x0000000c.
Data value: Greater than 0x4e204e20 and less than 0x9c409c40.

Programming options:
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.

18.2.1.3.3 Half-Word Working Mode Example 2

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 options:
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 examples can only be ignored by the software that handles the breakpoints. The following figure illustrates this partially supported scenario:

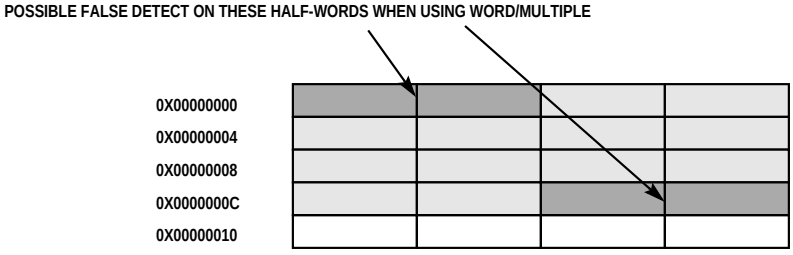


Figure 18-2. Partially Supported Watchpoints/Breakpoint Example

18.2.1.4 CONTEXT DEPENDENT FILTER. The core 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 it is programmed to recognize internal breakpoints (when MSR_{RI} =1), all parts of the code can be debugged, 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 working in masked mode, all internal breakpoints detected when MSR_{RI} =0 are lost and detected watchpoints are not counted by the debug counters. Detected watchpoints are always reported on the external pins, regardless of the value of the MSR_{RI} bit. The core defaults to masked mode after reset. It is input in the nonmasked mode by setting the BRKNOMSK bit in the LCTRL2 register.The BRKNOMSK bit controls all internal breakpoints (I-breakpoints and L-breakpoints). See Table 18-14 for details.

18.2.1.5 IGNORE FIRST MATCH OPTION. To facilitate the debugger utilities of “continue” and “go from x”, the ignore first match option is supported for the instruction breakpoints. When an instruction breakpoint is first enabled, the first instruction will not cause an instruction breakpoint if the IFM bit in the instruction support control (ICTRL) register is set. This is used for “continue” utilities. When IFM is clear, every matched instruction can cause an instruction breakpoint. This is used for “go from x”. The IFM bit is set by the software and cleared by the hardware 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.

18.2.1.6 GENERATING COMPARE TYPES. Using the four compare types–equal to, not equal to, greater than, and less than–it is possible to generate two additional compare types:

- Greater than or equal to
- Less than or equal to

Generating the greater than or equal to compare type can be accomplished by using the greater than compare type and programming the comparator to the value in question minus 1. Likewise, generating the less than or equal to compare type can be accomplished by using the less than compare type and programming the comparator to the value in question plus 1. This method does not work for the following boundary cases:

- Less than or equal to the largest unsigned number (1111...1)
- Greater than or equal to the smallest unsigned number (0000...0)
- Less than or equal to the maximum positive number when in signed mode (0111...1)
- Greater than or equal to the maximum negative number when in signed mode (1000...)

These boundary cases do not require special support because they are considered ‘always true’. They can be programmed using the ignore option of the load/store watchpoint programming. See Table 18-14 for more information.

18.2.2 Basic Watchpoint/Breakpoint Operation

18.2.2.1 INSTRUCTION SUPPORT. There are four instruction address comparators (A, B, C, and D). Each one is 30 bits long and generates two output signals—equal to and less than—which generates equal, not equal to, 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.

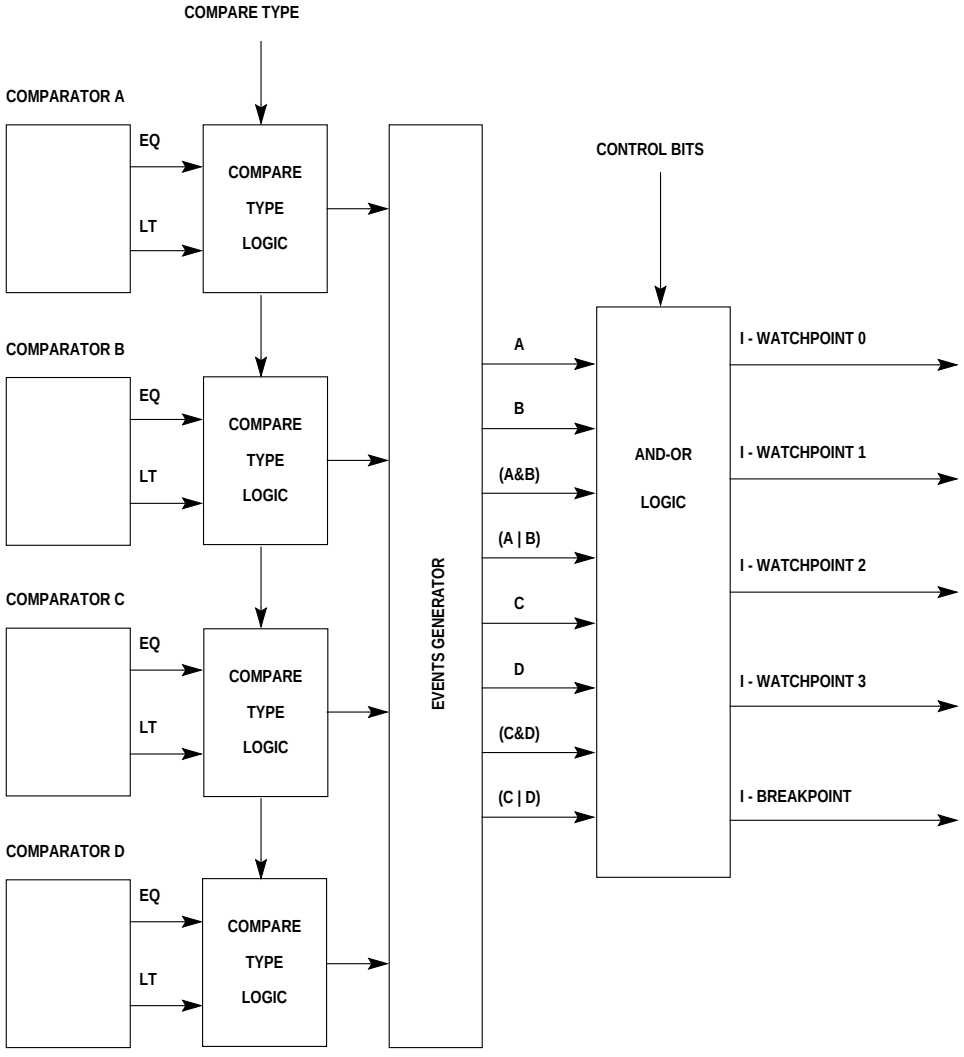


Figure 18-3. Instruction Support General Structure

Table 18-4. Instruction Watchpoints Programming Options

NAME	DESCRIPTION	PROGRAMMING OPTIONS
IW0	First instruction watchpoint	Comparator A Comparators (A & B)
IW1	Second instruction watchpoint	Comparator B Comparator (A | B)
IW2	Third instruction watchpoint	Comparator C Comparators (C & D)
IW3	Fourth instruction watchpoint	Comparator D Comparator (C | D)

18.2.2.2 LOAD/STORE SUPPORT. There are two load/store address comparators (E and F) that compare the 32 address bits and the cycle's attributes (read/write). The two least-significant bits are masked ignored whenever a word is accessed and the least-significant bit is masked whenever a half-word is accessed. Each comparator generates two output signals—equal to and less than. These signals generate one of four events from each comparator—equal to, not equal to, greater than, or less than. For more information, refer to **Section 18.1.1.6 Benefits of Compression**.

There are two load/store data comparators (G and H) that are 32 bits wide and can be programmed to treat numbers as signed or unsigned values. Each data comparator operates as four independent byte comparators that have a mask bit and generate two output signals—equal to and less than (if the mask bit is not set.) Therefore, each 32-bit comparator has eight output signals. These signals generate the "equal to and less than" signals according to the compare size programmed by the user (byte, half-word, word). When operating in byte mode, all signals are significant. In half-word mode only four signals from each comparator are significant, and in word mode only two signals are significant.

One of the following four match events are generated by the equal to and less than signals—equal to, not equal to, greater than, or less than—depending on the compare type. Therefore, from the two 32-bit comparators, eight match indications are generated—Gmatch[0:3] and Hmatch[0:3]. According to the lower bits of the address and the size of the cycle, only match indications detected on bytes with valid information are validated. The rest are negated. If the executed cycle has a smaller size than the compare size (a byte access when the compare size is word or half-word), no match indication will be asserted. Using the match indication signals, four load/store data events are generated as shown in the Table 18-5.

Table 18-5. Load/Store Data Events

EVENT NAME	EVENT FUNCTION
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 user programming.

Table 18-6. Load/Store Watchpoint Programming Options

NAME	DESCRIPTION	I-ADDRESS EVENT PROGRAMMING OPTIONS	L-ADDRESS EVENT PROGRAMMING OPTIONS	L-DATA EVENT PROGRAMMING OPTIONS
LW0	First Load/Store Watchpoint	IW0, IW1, IW2, IW3, Ignore I-address events	Comparator E Comparator F Comparators (E & F) Comparators (E | F) Ignore L-address events	Comparator G Comparator H Comparators (G & H) Comparators (G | H) Ignore L-data events
LW1	Second Load/Store Watchpoint	IW0, IW1, IW2, IW3, Ignore I-address events	Comparator E Comparator F Comparators (E & F) Comparators (E | F) Ignore L-address events	Comparator G Comparator H Comparators (G & H) Comparators (G | H) Ignore L-data events

When programming the load/store watchpoints to ignore L-address events and L-data events, the instruction must be a load/store instruction to trigger the load/store watchpoint event.

18.2.2.3 COUNTER SUPPORT. There are two 16-bit down counters that count one of the instruction watchpoints or one of the load/store watchpoints. Both generate the corresponding breakpoint when they reach zero. When working in masked mode, the counters do not count detected watchpoints when MSR_{RI} =0. Counter values are not predictable if they are counting watchpoints programmed on the instructions that alter the counters. Readings from the active counters must be synchronized by inserting a **sync** instruction before a read is performed. For details, refer to **Section 18.1.1.6 Benefits of Compression**.

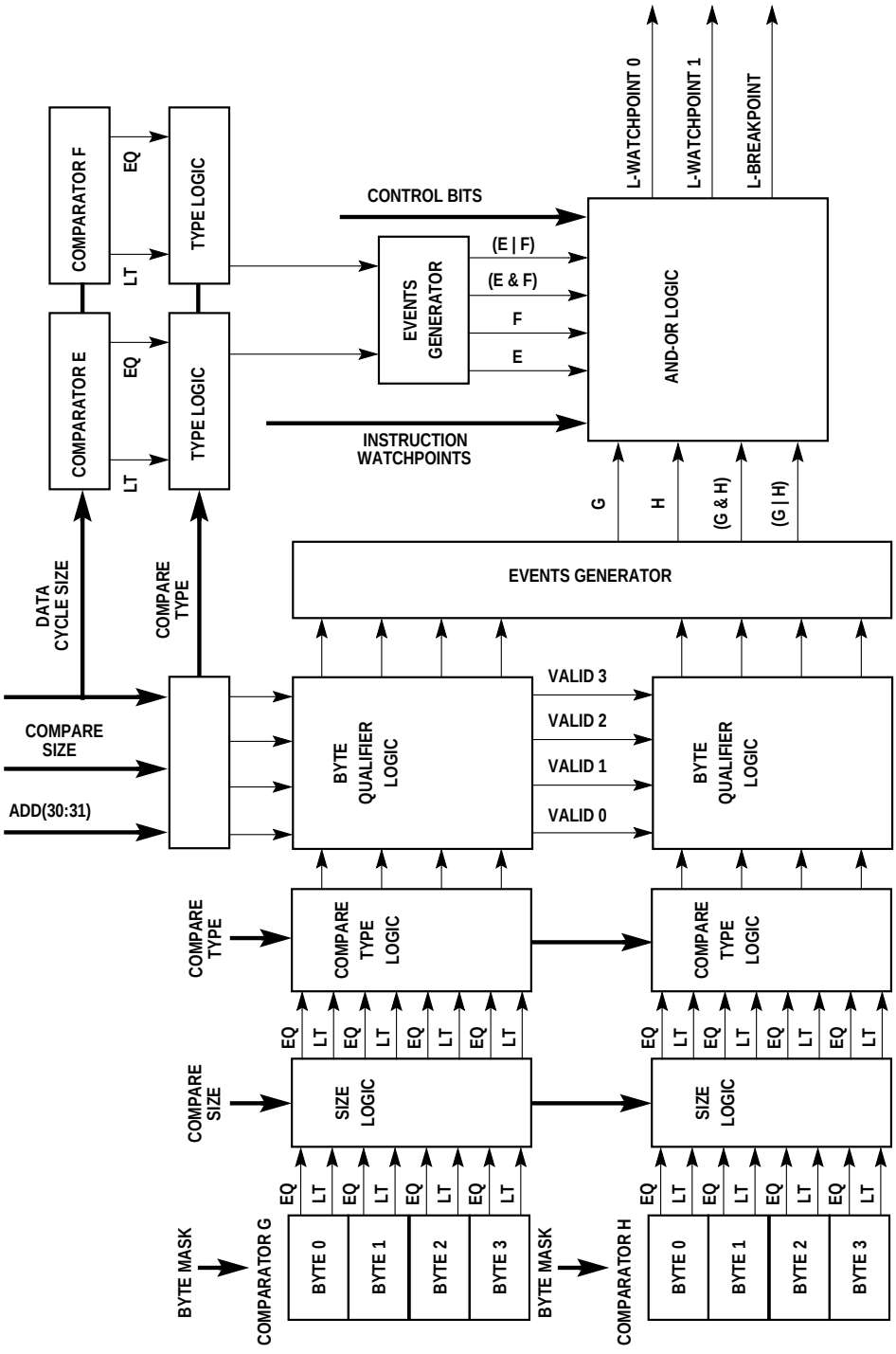


Figure 18-4. Load/Store Support General Structure

NOTE

When programmed to count instruction watchpoints, the last instruction that decrements the counter to zero is treated like any other instruction breakpoint in the sense that it is not executed before the machine branches to the breakpoint exception routine. As a side effect of this behavior, the value of the counter inside the breakpoint exception routine equals 1 and not zero. When programmed to count load/store watchpoints, the last instruction that decrements the counter to zero is treated like any other load/store breakpoint in the sense that it is executed before the machine branches to the breakpoint exception routine. Therefore, the value of the counter inside the breakpoint exception routine equals zero.

18.2.2.4 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 18.3.3.3 Development Port Serial Communications**. The value used by the breakpoint generation logic is the bit-wise OR of the software trap enable bits written using the **mtspr** instruction and the development port trap enable bits that are serially shifted using the development port. The software trap enable bits and development port trap enable bits can be read from the ICTRL and LCTRL2 registers using the **mtspr** instruction. For exact bit placement, refer to Tables 18-18 and 18-20.

18.3 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 core supports this configuration.

The development system interface of the core uses the development port, which is a dedicated serial port that does not need any of the regular system interfaces. System activity can be controlled from the development port when the core is in debug mode. The development port is a relatively inexpensive interface that allows the development system to operate in a lower frequency than the core's frequency. It is also possible to debug the core using monitor debugger software. For more information, refer to **Section 18.4 The Software Monitor Debugger**.

In debug mode, the core fetches all instructions from the development port. Data can be read from or written to the development port. This allows memory and registers to be read and modified by a development tool or 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 18.1.1.6 Benefits of Compression**.

You can work in debug mode directly out of reset or the core can be programmed to enter into the debug mode as a result of a predefined sequence of events. These events can be any interrupt or exception in the core system (including the internal breakpoints), in addition to two levels of development port requests and one peripheral breakpoint request generated internally and externally. Each of these can be programmed as a regular interrupt that causes the machine to branch to its interrupt vector or as a special interrupt that causes debug mode entry. When in debug mode, the **rfi** instruction returns the machine to its regular work mode. The relationship between debug mode logic and the rest of the core is illustrated in the following figure.

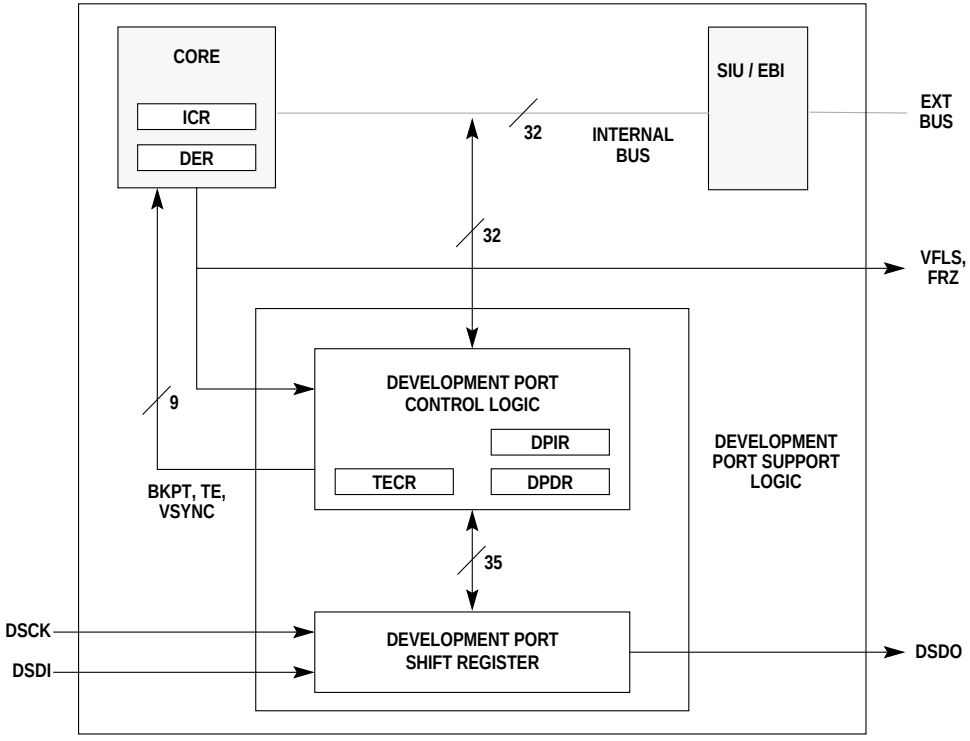


Figure 18-5. Relationship Between the Core and Debug Mode

The development port provides a full-duplex serial interface for communications between the internal development support logic of the core and an external development tool. The development port can operate in two working modes–trap enable mode and debug mode.

18.3.1 Trap Enable Mode

Trap enable mode is used to shift the following control signals into the core internal development support logic:

- An instruction trap enable signal is used to program the instruction breakpoint on-the-fly.
- A load/store trap enable signal is used to program the load/store breakpoint on-the-fly.
- A nonmaskable breakpoint is used to assert the nonmaskable external breakpoint.
- A maskable breakpoint is used to assert the maskable external breakpoint.
- A VSYNC control code is used to assert and negate VSYNC operation.

In debug mode, the development port also controls the debug mode features of the core. For more details, refer to **Section 18.3.3 The Development Port**.

18.3.2 Debug Mode

Debug mode provides the development system with the following functions:

- Controls and maintains all circumstances of processor execution.
  - — The development port can force the core to enter debug mode even when the external interrupts are disabled.
- Debug mode can be entered immediately out of reset, thus allowing the user to debug a system without ROM.
- The events that cause the machine to enter into debug mode can be selectively defined through an enable register.
- Contains a cause register that indicates why debug mode is entered.
- After entering debug mode, program execution continues 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 mode.
- A simple method is provided for memory dump and load via the data register of the development port that is accessed with **mtspr** and **mfspr**.
- The processor enters the privileged state (MSR_{PR} =0) in debug mode, thus allowing execution of any instruction and access to any storage location.
- An OR signal of all interrupt cause register bits enables the development port to detect pending events while already in debug mode. For example, the development port can detect a debug mode access to nonexisting memory space.

Figure 18-6 illustrates the debug mode logic implemented in the core.

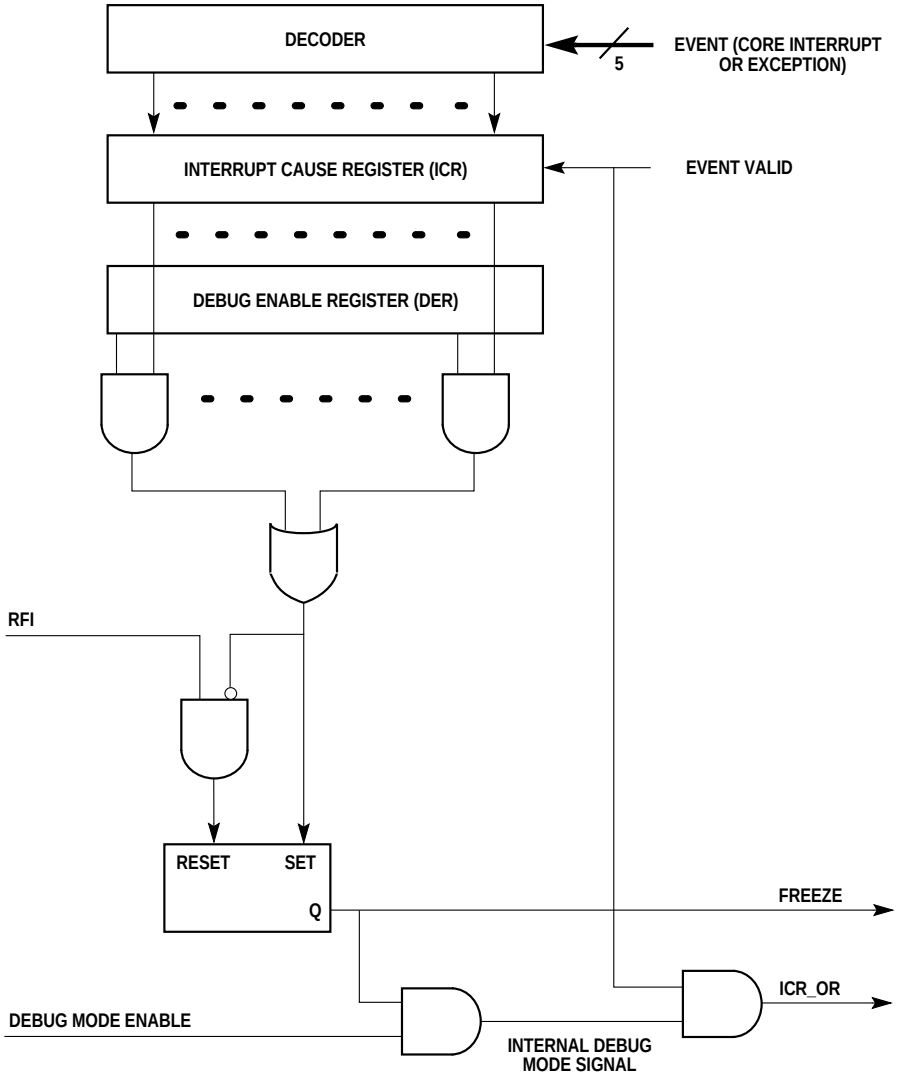


Figure 18-6. Debug Mode Logic Implementation

18.3.2.1 DEBUG MODE ENABLE VS. DEBUG MODE DISABLE. For protection purposes, there are two possible working modes—debug mode enable and debug mode disable. These modes are selected once at reset. Debug mode is enabled by asserting the DSCK pin during reset and the state of this pin is sampled three clocks before $\overline{\text{SRESET}}$ is negated. If the DSCK pin is sampled negated, debug mode is disabled until the subsequent reset that occurs when the DSCK pin is asserted. When debug mode is disabled, the internal watchpoint/breakpoint hardware is still operational and can be used by a software monitor program for debugging purposes. A timing diagram for the enabling debug mode is illustrated in Figure 18-7.

NOTE

Since $\overline{\text{SRESET}}$ negation time is dependent on an external pull-up resistor, any reference to $\overline{\text{SRESET}}$ negation time refers to the time the MPC801 releases $\overline{\text{SRESET}}$. If the rise time of $\overline{\text{SRESET}}$ is long because of a large resistor, the setup time for the debug port signals should be adjusted accordingly.

When debug mode is disabled, all development support registers are accessible when MSR_{PR} =0 and can be used by the monitor debugger software. However, the processor never enters debug mode and the ICR and DER are only used to assert and negate the freeze signal. For more information on the software monitor debugger, refer to **Section 18.4 The Software Monitor Debugger**. Only when the core is in debug mode are all development support registers accessible. Therefore, the development system has full control of the core's development support features. For more information, see Table 18-12.

18.3.2.2 ENTERING DEBUG MODE. Debug mode entry can be the result of a number of events. All events have a programmable enable bit so you can selectively decide that the cause of debug mode entry as well as the events that require regular interrupt handling. Entering debug mode is possible immediately out of reset, thus allowing a system to be debugged without ROM. This occurs 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.

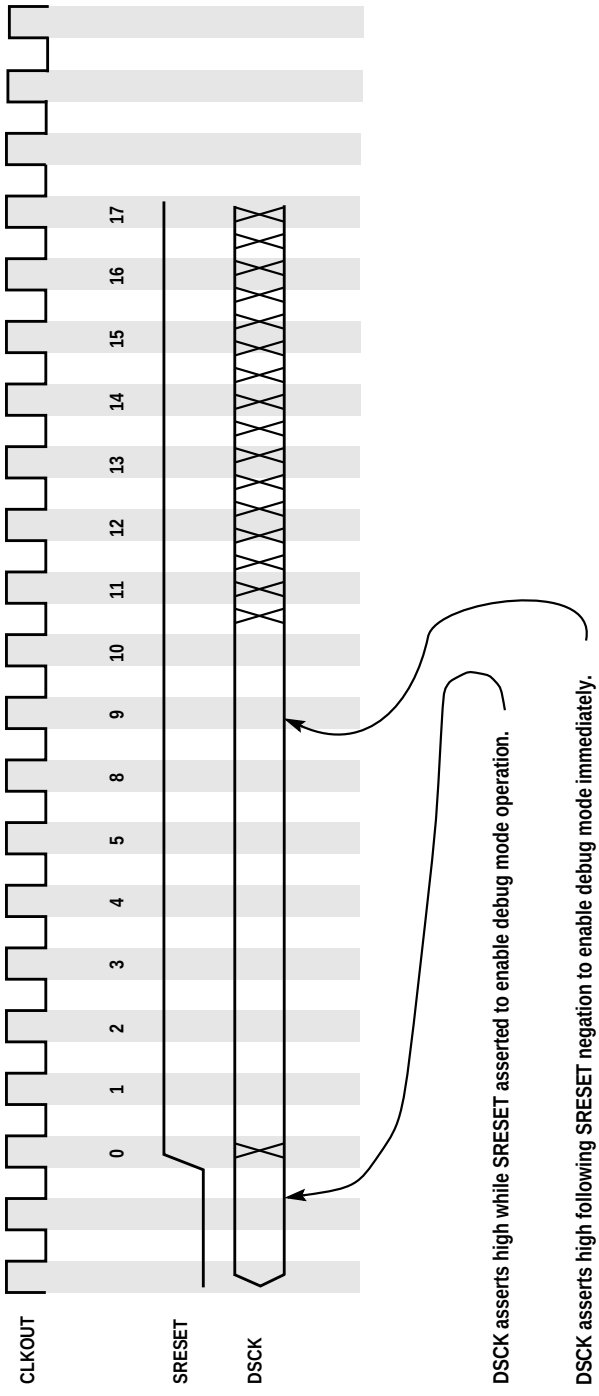


Figure 18-7. Debug Mode Reset Configuration Timing Diagram

To avoid entering debug mode after reset, the DSCK pin must be negated no later than seven clock cycles after $\overline{\text{SRESET}}$ negates to allow the processor to jump to the reset vector and begin normal execution. Entering debug mode immediately after reset, Bit 31 (development port interrupt bit) of the interrupt cause register (ICR) is set. For details, refer to the timing diagram illustrated in Figure 18-7. 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 debug mode is enabled or disabled. The internal freeze signal is connected to all relevant internal modules that can be programmed to stop all operations in response to assertion of the freeze signal. Furthermore, the freeze indication is negated when exiting the debug mode. For more information, refer to **Section 18.4.1 Freeze Indication**.

The following list of events could cause the core to enter debug mode. Each event results in debug mode entry if debug mode is enabled and the corresponding enable bit is set. The reset values of the enable bits allow the debug mode features to be used without having to program the debug enable register (DER). For more information, see Table 18-18.

- System reset as a result of $\overline{\text{SRESET}}$ assertion
- Check stop
- Machine check interrupt
- Implementation specific instruction TLB miss
- Implementation specific instruction TLB error
- Implementation specific data TLB miss
- Implementation specific data TLB error
- External interrupt, recognized when MSR_{EE} =1
- Alignment interrupt
- Program interrupt
- Floating-point unavailable interrupt
- Decrementer interrupt, recognized when MSR_{EE} =1
- System call interrupt
- Trace asserted when in single or branch trace mode
- Implementation dependent software emulation interrupt
- Instruction breakpoint is recognized only when MSR_{RI} =1 and when breakpoints are masked. When breakpoints are not masked, they are always recognized.
- Load/store breakpoint is recognized only when MSR_{RI} = 1 and when breakpoints are masked. When breakpoints are not masked, they are always recognized.
- Peripheral breakpoint from the development port generated by external modules are recognized only when MSR_{RI} =1.
- Development port nonmaskable interrupt occurs as a result of a debug station request. Useful in some catastrophic events like an endless loop when MSR_{RI} =0. As a result of this event, the machine can enter a nonrestartable state.

The processor enters into the debug mode state when at least one of the bits in the ICR is set, the corresponding bit in the DER is enabled, and debug mode is enabled. When debug mode is enabled and an enabled event occurs, the processor waits until its pipeline is empty and then starts fetching the next instructions from the development port. For information on the exact value of SRR0 and SRR1, refer to **Section 7.3.7.3 Definitions**. When the processor is in debug mode, the freeze indication is asserted, thus allowing any properly programmed peripheral to stop. The fact that the core is in debug mode is also broadcasted to the external world using the value b'11' on the VFLS pins. The freeze signal can be asserted by the software when debug mode is disabled. The development port should read the value of the ICR to find out what causes debug mode entry. Reading the ICR clears all of its bits.

18.3.2.3 CHECKSTOP STATE AND DEBUG MODE. The core enters checkstop state if the machine check interrupt is disabled (MSR_{ME} =0) and a machine check interrupt is detected. However, if a machine check interrupt is detected when MSR_{ME} =0, debug mode is enabled, the checkstop enable bit in the DER is set, and the core enters debug mode rather then the checkstop state. The various actions taken by the core when a machine check interrupt is detected are provided in the following table.

Table 18-7. Checkstop State and Debug Mode

MSR_{ME}	DEBUG MODE ENABLE	CHSTPE[2]	MCIE[2]	ACTION PERFORMED BY THE CORE WHEN A MACHINE CHECK INTERRUPT IS DETECTED	INTERRUPT CAUSE REGISTER VALUE
0	0	X	X	Enter Checkstop State	0x20000000
1	0	X	X	Branch to the Machine Check Interrupt	0x10000000
0	1	0	X	Enter 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. The checkstop enable bit of the DER.

2. The machine check interrupt enable bit of the DER.

18.3.2.4 SAVING THE MACHINE STATE IN DEBUG MODE. If entering debug mode is the result of a load/store-type exception, the data address register (DER) and data/storage interrupt status register (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 the save/restore registers 0 and 1 (SRR0 and SRR1).

18.3.2.5 RUNNING IN DEBUG MODE. When running in debug mode, all fetch cycles access the development port, regardless of the cycle's actual address. All load/store cycles access the real memory system according to the cycle's address. The data register of the development port is mapped as a special control register and is accessed using the **mtspr** and **mfspr** instructions, via special load/store cycles.

Exceptions are treated differently in debug mode. When in debug mode, the ICR is updated when an exception is recognized by the event that caused the exception. A special error indication (ICR_OR) is asserted for one clock cycle to notify the development port that an exception has occurred. Execution then continues in debug mode without any change in SRR0 and SRR1. ICR_OR is asserted before the next fetch occurs so the development system can detect the excepting instruction. However, not all exceptions are recognizable in debug mode. Breakpoints and watchpoints are not generated by the hardware when in debug mode, regardless of the MSR_{RI} bit's value. When entering debug mode, the MSR_{EE} bit is cleared by the hardware, thus forcing the hardware to ignore external and decrementer interrupts.

CAUTION

Setting the MSR_{EE} bit while in debug mode is strictly forbidden.

The reason for this restriction is that the external interrupt event is a level signal and because the core only reports exceptions in debug mode and does not perform exception processing, the core hardware does not clear the MSR_{EE} bit. This event, if enabled, is then recognized on every clock. When the ICR_OR signal is asserted the development station must search the ICR to find out what caused the exception. Since the values in SRR0 and SRR1 do not change if an exception is recognized in debug mode, they only change once when entering debug mode. However, saving SRR0 and SRR1 when entering debug mode is unnecessary.

18.3.2.6 EXITING DEBUG MODE. The **rfi** instruction is used to exit from debug mode and return to normal processor operation and negate the freeze signal. The development system may monitor the FRZ signal or status to make sure the MPC801 is out of debug mode. It is the responsibility of the software to read the ICR before performing the **rfi** instruction. Failure to do so forces the core to immediately reenter debug mode and reassert the freeze signal if an asserted bit in the ICR has a corresponding enable bit set in the DER.

18.3.3 The Development Port

The development port provides a full-duplex serial interface for communications between the internal development support logic and an external development tool. The relationship of the development support logic to the rest of the core is illustrated in Table 18-5. Notice that the development port support logic is shown as a separate block for clarity. It will be implemented as part of the system interface unit module.

18.3.3.1 THE DEVELOPMENT PORT PINS. The following development port pin functions are provided:

- Development serial clock
- Development serial data in
- Development serial data out
- Freeze

18.3.3.1.1 Development Serial Clock. The DSCK pin 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 to the DSDO pin. Future references to the DSCK pin imply the internally synchronized value of the clock. The DSCK pin must be driven either high or low at all times and is not allowed to float. With a resistor, a typical target environment would pull this input low.

The clock can be implemented as a free-running or gated clock. The ready and start signals control data shifting, so the clock does not need to be gated with the serial transmissions. The DSCK pin is used at reset to enable debug mode either immediately following reset or to enter debug mode during an event.

18.3.3.1.2 Development Serial Data In. Data to be transferred into the development port shift register is presented at the 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 determine the development port clock mode. Refer to **Section 18.3.3.3 Development Port Serial Communications** for more information.

18.3.3.1.3 Development Serial Data Out. The debug mode logic shifts data out of the development port shift register using the 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 18-10 for details about DSDO data.

18.3.3.1.4 Freeze. The freeze signal means that the processor is in debug mode and 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.

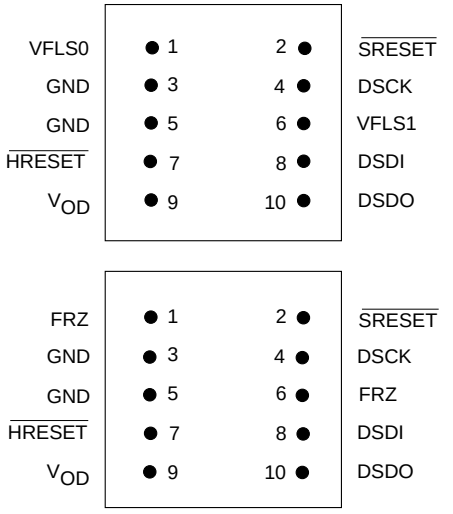


Figure 18-8. Development Port/BDM Connector Pinout Options

18.3.3.2 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). However, these registers are physically implemented as two registers—the development port shift and trap enable control registers. 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.

18.3.3.2.1 Development Port Shift Register. The 35-bit development port shift register has instructions and data 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 18.3.3.3 Development Port Serial Communications**.

The instructions or data are then transferred in parallel to the core and the trap enable control register. When the processor enters debug mode it fetches instructions from the DPIR, which causes an access to the development port shift register. These instructions are serially loaded into the shift register from the DSDI using DSCK or CLKOUT as the shift clock (similar to the way data is transferred to the core). Data is shifted into the shift register and read by the processor when a “move from special purpose register DPDR” instruction is executed. Data is also parallel loaded into the development port shift register from the core by executing a “move to special purpose register DPDR” instruction. It is then serially shifted out to the DSDO using DSCK or CLKOUT as the shift clock.

18.3.3.2.2 Trap Enable Control Register. The 9-bit trap enable control register (TECR) is loaded from the development port shift register. The content of this register drives the six trap enable signals, two breakpoint signals, and the VSYNC signal to the core. The transfer data to trap enable control register commands are used to force the appropriate bits to transfer to this register.

The trap enable control register is not accessed by the core, but supplies signals to the core. The trap enable bits, VSYNC bit, and the breakpoint bits of this register are loaded from the development port shift register as a result of trap enable mode transmissions. The trap enable bits are reflected in the ICTRL and LCTRL2 special registers. Refer to **Section 18.5.2 Development Port Registers** for more information on the support registers.

18.3.3.2.3 Decoding the Development Port Registers. The development port shift register is selected when the core accesses the DPIR or DPDR registers. Accesses to these two special purpose registers occur in debug mode and appear on the internal bus as an address and the assertion of an address attribute signal indicating that a special purpose register is being accessed. The DPIR is read by the core to fetch all instructions when in debug mode and the DPDR is read and written to transfer data between the core and external development tools. The DPIR and DPDR are pseudo-registers, so decoding either of these registers causes the development port shift register to be accessed. The debug mode logic knows whether the core is fetching instructions or reading or writing data. A sequence error is signaled to the external development tool when the core expected result and the general-purpose register result do not match. An example of this would be when an instruction is received instead of the expected data.

18.3.3.3 DEVELOPMENT PORT SERIAL COMMUNICATIONS

18.3.3.3.1 Clock Mode Selection. All of the serial transmissions are clocked transmissions and are synchronous communications. With CLKOUT, the transmission clock can either be synchronous or asynchronous. The development port has two methods for clocking 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 MPC801. The other communication method requires 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 that does not have access to the CLKOUT signal or has either a delayed or skewed CLKOUT signal. The timing diagram in Figure 18-9 illustrates serial communication asynchronous clocked timing.

The second clock mode is called synchronous self-clocked and does not require an input clock. Instead, the port is clocked by the system clock. The DSDI input is required to meet setup and hold time requirements, with respect to the CLKOUT rising edge. The data rate for this mode is always the same as the system clock. The timing diagram in Figure 18-10 illustrates serial communication synchronous self-clocked timing. The selection of clocked or self-clocked mode is made at reset. The state of the DSDI input is latched eight clocks after $\overline{\text{SRESET}}$ is negated. If it is latched low, asynchronous clocked mode is enabled. If it is latched high, then synchronous self-clocked mode is enabled. The timing diagram in Figure 18-11 illustrates the clock mode selection following reset.

Since DSDI is used to select the development port clock scheme, any transitions on DSDI during clock mode select must be prevented from being recognized as the start of a serial transmission. The port will not begin scanning for the start bit of a serial transmission until 16 clocks after $\overline{\text{SRESET}}$ is negated. If DSDI is asserted 16 clocks after $\overline{\text{SRESET}}$ negates, the port waits until DSDI is negated before it starts scanning for the start bit.

18.3.3.3.2 Trap Enable Mode. When not in debug mode, the development port begins communicating by setting DSDO low to show that all activity related to the previous transmission is complete and that a new transmission can begin. The start of a serial transmission from an external development tool to the development port is signaled by a start bit. A mode bit in the transmission defines it as either a trap enable mode or debug mode transmission. If the mode bit is set, the transmission will be 10 bits long and only seven data bits will be shifted into the shift register. These seven bits will be latched into the TECR. A control bit determines whether the data is latched into the TECR's trap enable, VSYNC, or breakpoint bits.

NOTE

The development port shift register is 35 bits wide, but trap enable mode transmissions only use 10 of the 35 bits—the start/ready bits, mode/status bits, control/status bits, and seven least-significant data bits. The data encoding shifted into the development port shift register (through the DSDI pin) is shown in Tables 8-8 and 8-9.

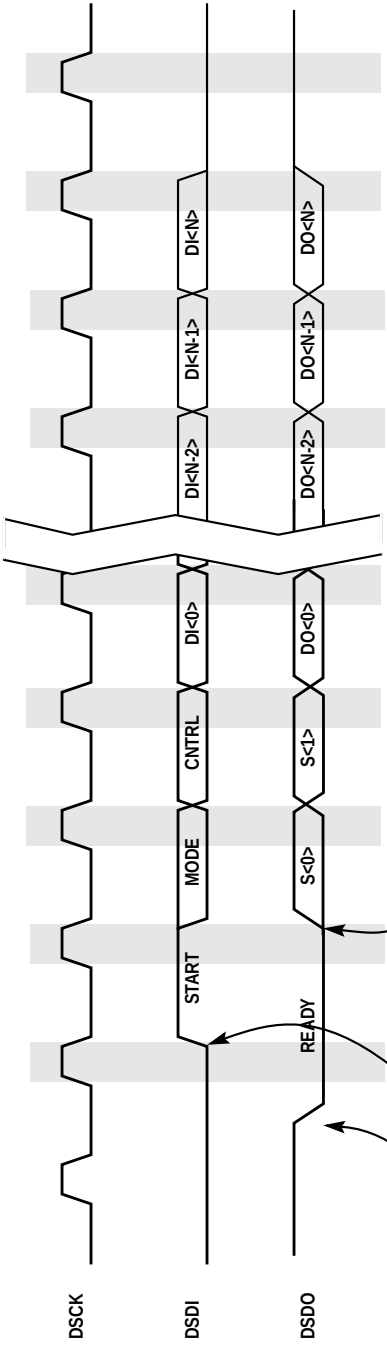


Figure 18-9. Asynchronous Clocked Serial Communications Timing Diagram

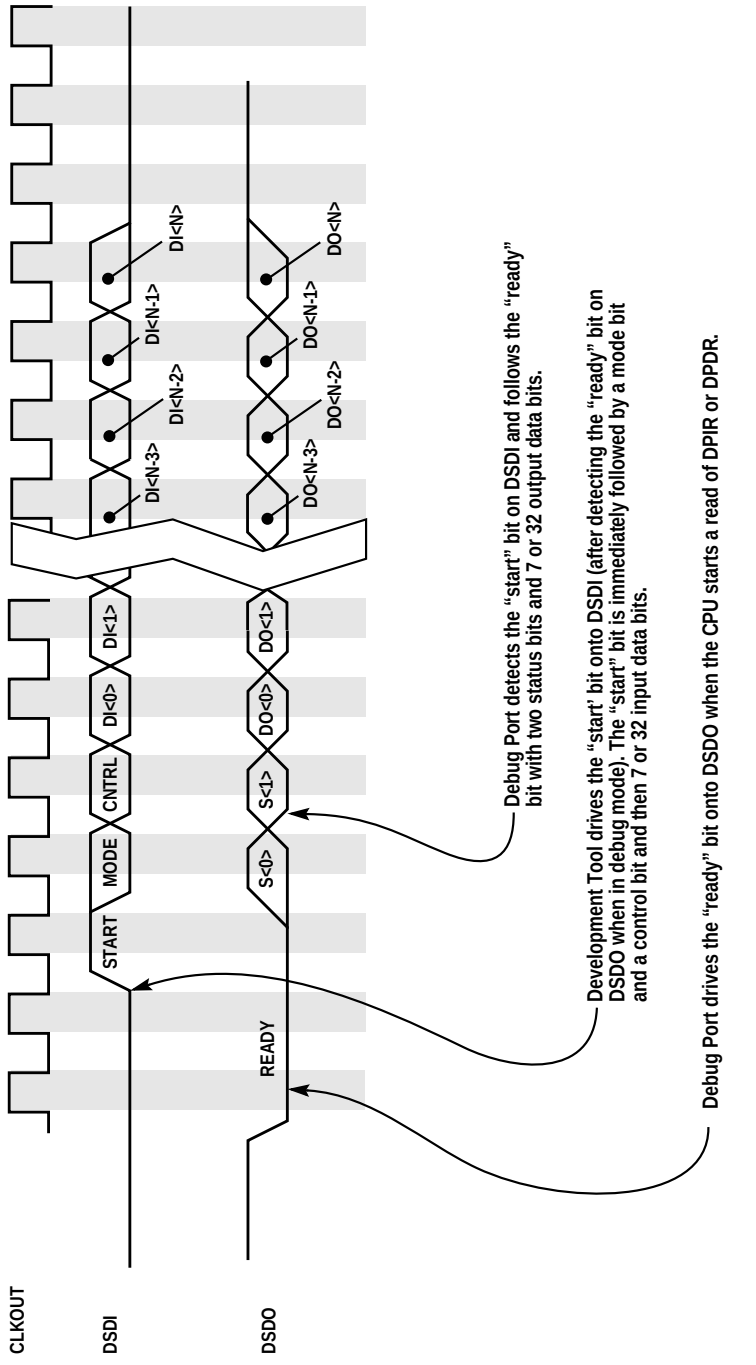


Figure 18-10. Synchronous Self-Clocked Serial Communications Timing Diagram

18

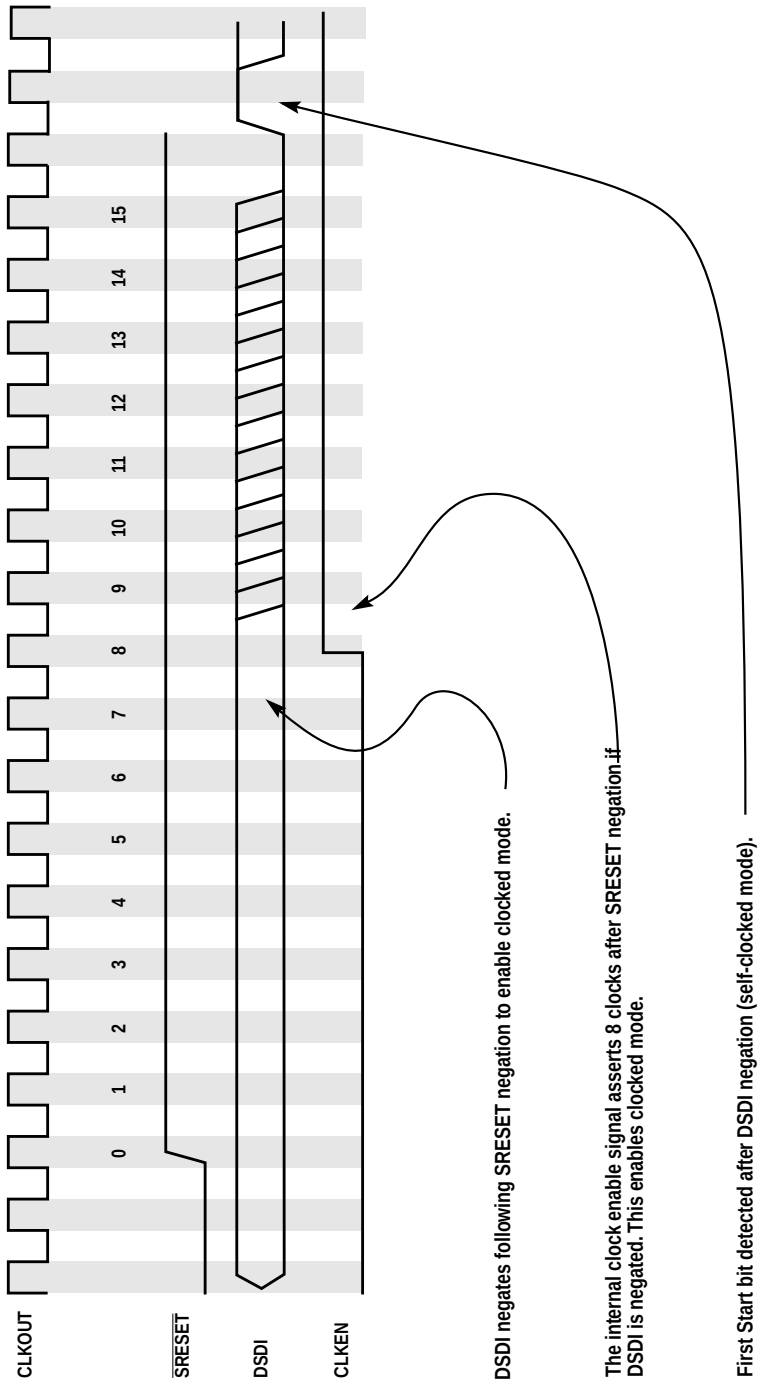


Figure 18-11. Enabling Clock Mode Following Reset Timing Diagram

Table 18-8. Trap Enable Data Shifted Into Development Port Shift Register

START	MODE	CONTROL	FIRST	SECOND	THIRD	FOURTH	FIRST	SECOND	VSYNC	FUNCTION
			INSTRUCTION				DATA			
			WATCHPOINT TRAP ENABLES							
1	1	0	0 = Disabled 1 = Enabled							Transfer Data to Trap Enable Control Register

Table 18-9. Debug Port Command Shifted Into the 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 enable and VSYNC functions are described in **Section 18.2 Watchpoint And Breakpoint Generation** and **Section 18.1 Program Flow Tracking**. The debug port command function allows the development tool to either assert or negate breakpoint requests, reset the processor, or activate or deactivate the fast download procedure.

NOTE

In trap enable mode, there is no data out of the development port. Data from the development port in the trap enable mode is shown in Table 18-10.

Table 18-10. Status/Data Shifted Out of the Development Port Shift Register

READY	STATUS [0:1]		DATA			FUNCTION
			BIT 0	BIT 1	BITS 2–31 OR 2–6, DEPENDING ON THE INPUT MODE	
(0)	0	0	DATA			Valid Data From Core
(0)	0	1	Freeze Status	Download Procedure In Progress	1s	Sequencing Error
(0)	1	0			1s	Core 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.

In trap enable mode the "Valid Data from CPU" and "CPU Interrupt" status cannot occur. When not in debug mode, 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 considered a no operation (NOP) function. The null output encoding indicates that the previous transmission did not have any associated errors. When not in debug mode, ready is asserted at the end of each transmission. If debug mode is not enabled and transmission errors are guaranteed not to occur, the status output is not needed.

18.3.3.3.3 Debug Mode. When in debug mode the development port starts communicating by setting DSDO low to show that the core is trying to read an instruction from the DPIR or data from the DPDR. When the core writes data to the port to be shifted out, the ready bit is not set. The port waits for the core to read the next instruction before asserting ready. This allows duplex operation of the serial port while allowing the port to control all transmissions from the external development tool. After detecting this ready status, the external development tool begins transmitting to the development port with a start bit (logic high) on the DSDI pin.

In debug mode, the 35 bits of the development port shift register are interpreted as a start/ready bit, mode/status bit, control/status bit, and 32 bits of data. All instructions and data for the core are transmitted with the mode bit cleared, thus indicating a 32-bit data field. The encoding of data shifted into the development port shift register through the DSDI pin is shown in Table 18-11. Unless otherwise specified, the data values in the last two functions are reserved.

Table 18-11. Debug Instructions/Data Shifted Into the Development Port Shift Register

START	MODE	CONTROL	INSTRUCTION / DATA (32 BITS)		FUNCTION
			BITS 0–6	BITS 7–31	
1	0	0	Core Instruction		Transfer Instruction to Core
1	0	1	Core Data		Transfer Data to Core
1	1	0	Trap Enable Bits	Not Exist	Transfer Data to Trap Enable Control Register
1	1	1	0011111	Not Exist	Negate Breakpoint Requests to Core
1	1	1	0	Not Exist	NOP

NOTE: See Table 18-8 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 core is trying to read an instruction or data from the port. The external development tool waits until it sees DSDO go low before it begins sending the next transmission. The control bit differentiates between instructions and data that allows the development port to detect when an instruction was entered when the core was expecting data and vice versa. If this occurs, a sequence error indication is shifted out in the next serial transmission. The trap enable function allows the development port to transfer data to the trap enable control register. The debug port command function allows the development tool to either negate breakpoint requests, reset the processor, or activate or deactivate the fast download procedure. The NOP function provides a null operation to use when there is data or a response to be shifted out of the data register. The next appropriate instruction or command will be determined by the value of the response or data shifted out.

The encoding of data shifted out of the development port shift register in debug mode is the same as for trap enable mode, as shown in Table 18-10. The valid data encoding is used when data has been transferred from the core to the development port shift register. This results when an instruction to move the contents of a general-purpose register to the DPDR occurs. The valid data encoding has the highest priority of all status outputs and will be reported even if an interrupt occurs at the same time. Since it is not possible for a sequencing error to occur that has valid data, there is no priority conflict with the sequencing error status. Also, any interrupt that is recognized at the same time that there is valid data, is not related to the execution of an instruction. Therefore, a valid data status will be output and the interrupt status will be saved for the next transmission.

The sequencing error encoding indicates that the external development tool inputs are not what the development port and/or the core was expecting. There are two possible causes of this error:

- The processor was trying to read instructions and data was shifted into the development port.
- The processor was trying to read data and an instruction was shifted into the development port.

Nonetheless, the port terminates the read cycle with a bus error. In turn, this bus error causes the core to signal that an interrupt exception has occurred. Since a status of sequencing error has higher priority than an exception, the port reports the sequencing error first and the core interrupt on the next transmission. The development port ignores the command, instruction, or data shifted in while the sequencing error or core interrupt is shifted out. The next transmission, after all error status is reported to the port, should either be a new instruction, trap enable, or command.

When an interrupt encoding occurs, it indicates that the core has encountered an interrupt while executing the previous instruction in debug mode. Interrupts can occur as a result of instruction execution either because of memory access faults or from unmasked external interrupts. 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 core to the development port shift register.

A 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:mfsprRX, DPDR
DATA word to be moved to memory
stwuRX, 0x4(RY)
until here

•••

Restore RX,RY
```

Figure 18-12. Example of Download Procedure Code

For large blocks of data, this sequence can take a significant amount of time to complete. But, using the fast download procedure of the debug port reduces this time because the need to transfer the instructions in the debug port loop is eliminated. The only transactions needed are those used to transfer the data to be placed in the system memory. Figures 18-13 and 18-14 illustrate the benefit of using the fast download procedure.

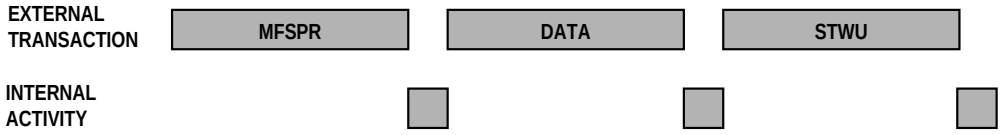


Figure 18-13. Slow Download Procedure Loop

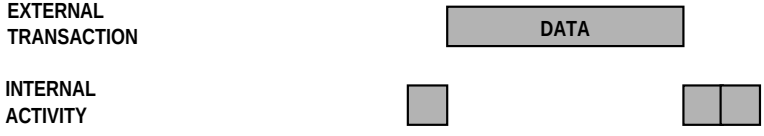


Figure 18-14. Fast Download Procedure Loop

The sequence of the instructions used in the fast download procedure is illustrated in Figure 18-12, 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.

18.4 THE SOFTWARE MONITOR DEBUGGER

When in debug mode disable, a software monitor debugger can use all of the development support features defined in the core. When debug mode is disabled, all events result in regular interrupt handling (the processor resumes execution in the corresponding interrupt handler). The ICR and DER only influence the assertion and negation of the freeze signal.

18.4.1 Freeze Indication

The internal freeze signal is connected to all relevant internal modules that can be programmed to stop all operations when the freeze signal is asserted. So that a software monitor debugger can signal when the debug software has executed, the internal freeze signal must be asserted or negated when debug mode is disabled. The FRZ signal indicates to the external world when the freeze signal is asserted or negated. The ICR and DER signals control whether or not the freeze signal is asserted or negated while it is in debug mode disable, as illustrated in Figure 18-6.

To assert the freeze signal, the software must program the relevant bits in the DER, but to negate it, the software must read the ICR to clear it and perform an **rfi** instruction. If the ICR is not cleared before the **rfi** instruction is performed, the freeze signal is not negated and it can 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. This process enables the software to accurately control the when the freeze line is asserted or negated.

18.5 PROGRAMMING THE DEVELOPMENT SUPPORT REGISTERS

These registers reside in the control register space and can be accessed using the **mtspr** and **mfspr** instructions. The addresses of these registers are in Table 6-9.

18.5.1 Protecting the Development Port Registers

The development support registers are protected according to the standards in the following table. Take note of the ICR and DPDR registers' special behavior.

Table 18-12. Development Support Register Protection

OPERATION	MSR_{PR}	DEBUG MODE ENABLE	IN DEBUG MODE	RESULT
Read Register	0	0	X	A read is performed and when reading ICR, it is also cleared.
	0	1	0	A read is performed and when reading ICR, it is not cleared.
	0	1	1	A read is performed and when reading ICR, it is also cleared.
	1	X	X	A read is not performed, a program interrupt is generated, and when reading ICR, it is not cleared.
Write Register	0	0	X	A write is performed, a write to ICR is ignored, and a write to DPDR is ignored.
	0	1	0	A write is ignored.
	0	1	1	A write is performed and a write to ICR is ignored.
	1	X	X	A write is not performed, but a program interrupt is generated.

NOTE: Ignored means the register is not modified and no interrupt is generated.

18.5.2 Development Port Registers

18.5.2.1 COMPARATOR A–D VALUE REGISTER. This bit has an undefined reset value.

CMPA-D

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

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

18.5.2.2 COMPARATOR E–F VALUE REGISTER. This bit has an undefined reset value.

CMPE-F

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

18.5.2.3 COMPARATOR G–H VALUE REGISTER. This bit has an undefined reset value.

CMPG-H

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

18.5.2.4 BREAKPOINT ADDRESS REGISTER. This bit has an undefined reset value.

BAR

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

18.5.2.5 INSTRUCTION SUPPORT CONTROL REGISTER

CMPG-H

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CTA			CTB			CTC			CTD			IW0		IW1	
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	IW2		IW3		SIW0EN	SIW1EN	SIW2EN	SIW3EN	DIW0EN	DIW1EN	DIW2EN	DIW3EN	IFM	ISCT_SER		

CTA—Compare Type of Comparator A

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTB—Compare Type of Comparator B

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTC—Compare Type of Comparator C

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTD—Compare Type of Comparator D

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

IW0—Instruction First Watchpoint Programming

0x = Not active (reset value).
10 = Match from Comparator A.
11 = Match from Comparators (A & B).

IW1—Instruction Second Watchpoint Programming

0x = Not active (reset value).
10 = Match from Comparator B.
11 = Match from Comparators (A | B).

IW2—Instruction Third Watchpoint Programming

0x = Not active (reset value).
10 = Match from Comparator C.
11 = Match from Comparators (C & D).

IW3—Instruction Fourth Watchpoint Programming

0x = Not active (reset value).
10 = Match from Comparator D.
11 = Match from Comparators (C | D).

SIW0EN—Software Trap Enable Selection of the First Instruction Watchpoint

0 = Trap disabled (reset value).
1 = Trap enabled.

SIW1EN—Software Trap Enable Selection of the Second Instruction Watchpoint

0 = Trap disabled (reset value).
1 = Trap enabled.

SIW2EN—Software Trap Enable Selection of the Third Instruction Watchpoint

0 = Trap disabled (reset value).
1 = Trap enabled.

SIW3EN—Software Trap Enable Selection of the Fourth Instruction Watchpoint

0 = Trap disabled (reset value).
1 = Trap enabled.

DIW0EN—Development Port Trap Enable Selection of the First Instruction Watchpoint
This is a read-only bit.

0 = Trap disabled (reset value).
1 = Trap enabled.

DIW1EN—Development Port Trap Enable Selection of the Second Instruction Watchpoint
This is a read-only bit.

0 = Trap disabled (reset value).
1 = Trap enabled.

DIW2EN—Development Port Trap Enable Selection of the Third Instruction Watchpoint
This is a read-only bit.

0 = Trap disabled (reset value).
1 = Trap enabled.

DIW3EN—Development Port Trap Enable Selection of the Fourth Instruction Watchpoint
This is a read-only bit.

0 = Trap disabled (reset value).
1 = Trap enabled.

IFM—Ignore First Match Only for Instruction Breakpoints

0 = Do not ignore first match, used for “Go To x” (reset value).
1 = Ignore first match (used for “Continue”).

ISCT_SER—Instruction Fetch Show Cycle and Core Serialize Control
Changing the Instruction show cycle programming starts to take effect only from the second instruction after the actual **mtspr** instruction to ICTRL.

000 = Core is fully serialized and show cycle will be performed for all fetched instructions (reset value). Has a reset value of 0x00000000.
001 = Core is fully serialized and show cycle will be performed for all changes in the program flow.
010 = Core is fully serialized and show cycle will be performed for all indirect changes in the program flow.
011 = Core is fully serialized and no show cycles will be performed for fetched instructions.
100 = Illegal.
101 = Core is not serialized (normal mode) and show cycle will be performed for all changes in the program flow. If the fetch of the target of a direct branch is aborted by the core, the target is not always visible on the external pins. This does not affect program trace.
110 = Core is not serialized (normal mode) and show cycle will be performed for all indirect changes in the program flow.
111 = Core is not serialized (normal mode) and no show cycles will be performed for fetched instructions.

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.

18.5.2.6 LOAD/STORE SUPPORT COMPARATORS CONTROL REGISTER

LCTRL1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CTE			CTF			CTG			CTH			CRWE		CRWF	
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CSG		CSH		SUSG	SUSH	CGBMSK			CHBMSK					RESERVED	

CTE—Compare Type, Comparator E

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTF—Compare Type, Comparator F

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTG—Compare Type, Comparator G

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTH—Compare Type, Comparator H

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CRWE—Select Match on Read/Write of Comparator E

0x = “Don’t care” (reset value).
10 = Match on read.
11 = Match on write.

CRWF—Select Match on Read/Write of Comparator F

0x = “Don’t care” (reset value).
10 = Match on read.
11 = Match on write.

CSG—Compare Size, Comparator G

00 = Reserved.
01 = Word.
10 = Half-Word.
11 = Byte.

CSH—Compare Size, Comparator H

00 = Reserved.
01 = Word.
10 = Half-Word.
11 = Byte.

SUSG—Signed/Unsigned Operating Mode for Comparator G

0 = Signed.
1 = Unsigned.

SUSH—Signed/Unsigned Operating Mode for Comparator H

0 = Signed.
1 = Unsigned.

CGBMSK—Byte Mask for Comparator G

0000 = All bytes are not masked.
0001 = Last byte of the word is masked.
•
•
•
1111 = All bytes are masked.

CHBMSK—Byte Mask for Comparator H

0000 = All bytes are not masked.
0001 = Last byte of the word is masked.
•
•
•
1111 = All bytes are masked.

Bits 30–31—Reserved

These bits are reserved and should be set to 0.

18.5.2.7 LOAD/STORE SUPPORT AND–OR CONTROL REGISTER. The reset value of this register is 0x00000000.

LCTRL2

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	LW0EN	LW0IA		LW0IA DC	LW0LA		LW0LA DC	LW0LD		LW0LD DC	LW1EN	LW1IA		LW1IA DC	LW1LA	
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	LW1LA DC	LW1LD		LW1LD DC	BRKNO MSK	RESERVED							DLW0E N	DLW1E N	SLW0E N	SLW1E N

LW0EN—First Load/Store Watchpoint Enable

0 = Watchpoint not enabled (reset value).
1 = Watchpoint enabled.

LW0IA—First Load/Store Watchpoint I-Address Watchpoint Selection

00 = First instruction watchpoint.
01 = Second instruction watchpoint.
10 = Third instruction watchpoint.
11 = Fourth instruction watchpoint.

LW0IADC—First Load/Store Watchpoint Care/Don’t Care I-Address Events

0 = “Don’t care.”
1 = “Care.”

LW0LA—First Load/Store Watchpoint L-Address Events Selection

00 = Match from comparator E.
01 = Match from comparator F.
10 = Match from comparators (E & F).
11 = Match from comparators (E | F).

LW0LADC—First Load/Store Watchpoint Care/Don’t Care L-Address Events

0 = “Don’t care.”
1 = “Care.”

LW0LD—First Load/Store Watchpoint L-Data Events Selection

00 = Match from comparator G.
01 = Match from comparator H.
10 = Match from comparators (G & H).
11 = Match from comparators (G | H).

LW0LDDC—First Load/Store Watchpoint Care/Don’t Care L-Data Events

0 = “Don’t care.”
1 = “Care.”

LW1EN—Second Load/Store Watchpoint Enable

0 = Watchpoint not enabled (reset value).
1 = Watchpoint enabled.

LW1IA—Second Load/Store Watchpoint I-Address Watchpoint Selection

00 = First instruction watchpoint.
01 = Second instruction watchpoint.
10 = Third instruction watchpoint.
11 = Fourth instruction watchpoint.

LW1IADC—Second Load/Store Watchpoint Care/Don't Care I-Address Event

0 = "Don't care."
1 = "Care."

LW1LA—Second Load/Store Watchpoint L-Address Events Selection

00 = Match from comparator E.
01 = Match from comparator F.
10 = Match from comparators (E & F).
11 = Match from comparators (E | F).

LW1LADC—Second Load/Store Watchpoint Care/Don't Care L-Address Events

0 = "Don't care."
1 = "Care."

LW1LD—Second Load/Store Watchpoint L-Data Events Selection

00 = Match from comparator G.
01 = Match from comparator H.
10 = Match from comparators (G & H).
11 = Match from comparator (G | H).

LW1LDDC—Second Load/Store Watchpoint Care/Don't Care L-Data Events

0 = "Don't care."
1 = "Care."

BRKNOMSK—Internal Breakpoints Nonmask Bit Controls Both Instruction Breakpoints and Load/Store Breakpoints

0 = Masked mode, breakpoints are recognized only when MSR_{RI} =1 (reset value).
1 = Nonmasked mode, breakpoints are always recognized.

Bits 21–27—Reserved

These bits are reserved and should be set to 0.

DLW0EN—Development Port Trap Enable Selection of the First Load/Store Watchpoint (Read-Only Bit)

0 = Trap disabled (reset value).
1 = Trap enabled.

DLW1EN—Development Port Trap Enable Selection of the Second Load/Store Watchpoint (Read-Only Bit)

0 = Trap disabled (reset value).
1 = Trap enabled.

SLW0EN—Software Trap Enable Selection of the First Load/Store Watchpoint

0 = Trap disabled (reset value).
1 = Trap enabled.

SLW1EN—Software Trap Enable Selection of the Second Load/Store Watchpoint

0 = Trap disabled (reset value).
1 = Trap enabled.

Each watchpoint programming consists of three control register fields—LWxIA, LWxLA, and LWxLD. All three conditions must be detected to assert a watchpoint.

18.5.2.8 BREAKPOINT COUNTER A VALUE AND CONTROL REGISTER. Reset value for bits 0-15 is undefined and for bits 16-31 it is 0x0000.

COUNTA

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CNTV															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED														CNTC	

CNTV—Counter Preset Value

Bits 16–29—Reserved

These bits are reserved and should be set to 0.

CNTC—Counter Source Select

00 = Not active (reset value).
01 = Instruction first watchpoint.
10 = Load/store first watchpoint.
11 = Reserved.

18.5.2.9 BREAKPOINT COUNTER B VALUE AND CONTROL REGISTER

COUNTB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CNTV															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED														CNTC	

CNTV—Counter Preset Value

Bits 16–29—Reserved
These bits are reserved and should be set to 0.

CNTC—Counter Source Select
- 00 = Not active (reset value).
- 01 = Instruction second watchpoint.
- 10 = Load/store second watchpoint.
- 11 = Reserved.

18.5.3 Debug Mode Registers

18.5.3.1 INTERRUPT CAUSE REGISTER. The reset value for this register is 0x00000000.

ICR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	RST	CHSTP	MCI	RESERVED		EXTI	ALI	PRI	FPUVI	DECI	RESERVED		SYSI	TR	RES
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RES	SEI	ITLBMS	ITLBER	DTLBMS	DTLBER	RESERVED						LBRK	IBRK	EBRK	DPI

Bits 0, 4, and 5—Reserved
These bits are reserved and should be set to 0.

RST—Reset Interrupt
This bit is set when the system reset pin is asserted. This pin is not implemented in the core.

CHSTP—Check Stop
This bit is set when the machine check interrupt is asserted and MSR_{ME} =0. 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.

MCI—Machine Check Interrupt
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.

EXTI—External Interrupt

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.

ALI—Alignment Interrupt

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.

PRI—Program Interrupt

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.

FPUVI—Floating-Point Unavailable Interrupt

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.

DECI—Decrementer Interrupt

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.

Bits 11–12 and 15–16—Reserved

These bits are reserved and should be set to 0.

SYSI—System Call Interrupt

This bit is set when the system call interrupt is asserted. Results in debug mode entry if debug mode is enabled and the corresponding enable bit is set.

TR—Trace Interrupt

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.

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.

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.

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.

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.

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.

Bits 22–27—Reserved

These bits are reserved and should be set to 0.

LBRK—Load/Store Breakpoint Interrupt

This bit is set as a result of the assertion of an load/store breakpoint. Results in debug mode entry if debug mode is enabled and the corresponding enable bit is set.

IBRK—Instruction Breakpoint Interrupt

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.

EBRK—External Breakpoint Interrupt

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.

DPI—Development Port Interrupt

This bit is set by the development port as a result of a debug station nonmaskable request or when entering debug mode immediately out of reset. Results in debug mode entry if debug mode is enabled and the corresponding enable bit is set.

The reason for entering debug mode is provided by the interrupt cause register. All bits are set by the hardware, cleared when the register is read, and cleared to zero when exiting reset. Any attempt to write to this register is ignored.

18.5.3.2 DEBUG ENABLE REGISTER. This register has a reset value of 0x2002000.

DER

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	RSTE	CHSTPE	MCIE	RESERVED		EXTIE	ALIE	PRIE	FPUVIE	DECIE	RESERVED		SYSIE	TRE	RES
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RES	SEIE	ITLBMSE	ITLBERE	DTLBMSE	DTLBERE	RESERVED						LBRKE	IBRKE	EBRKE	DPIE

Bits 0, 4, and 5—Reserved

These bits are reserved and should be set to 0.

RSTE—Reset Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

CHSTPE—Check Stop Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

MCIE—Machine Check Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

EXTIE—External Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

ALIE—Alignment Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

PRIE—Program Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

FPUVIE—Floating-Point Unavailable Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

DECIE—Decrementer Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

Bits 11–12 and 15–16—Reserved

These bits are reserved and should be set to 0.

SYSIE—System Call Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

TRE—Trace Interrupt Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

SEIE—Software Emulation Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

ITLBMSE—Implementation Specific Instruction TLB Miss Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

ITLBERE—Implementation Specific Instruction TLB Error Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

DTLBMSE—Implementation Specific Data TLB Miss Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

DTLBERE—Implementation Specific Data TLB Error Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

Bits 22–27—Reserved

These bits are reserved and should be set to 0.

LBRKE—Load/Store Breakpoint Interrupt Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

IBRKE—Instruction Breakpoint Interrupt Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

EBRKE—External Breakpoint Interrupt Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

DPIE—Development Port Nonmaskable Request Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

The debug enable register enables allows the enabling of events that cause the processor to enter into debug mode.

18.5.4 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 development system. An access to this register is initiated using the **mtspr** and **mfspr** instructions and it is implemented using a special bus cycle on the internal bus. For details, see Table 6-9.

SECTION 19
IEEE 1149.1 TEST ACCESS PORT

The MPC801 provides a dedicated user-accessible test access port 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 the development of this proposed standard under the sponsorship of the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The MPC801 implementation supports circuit board test strategies based on this standard.

The test access port (TAP) consists of five dedicated signal pins, a 16-state TAP controller, and two test data registers. A boundary scan register links all the device signal pins into a single shift register. The test logic, which is implemented using static logic design, is independent of the device system logic. The MPC801 gives you the capability to:

- Perform boundary scan operations to check circuit board electrical continuity.
- Bypass the MPC801 for a given circuit board test by effectively reducing the boundary scan register to a single cell.
- Sample the MPC801 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 19.5 Nonscan Chain Operation** for details.

The MPC801's JTAG includes a TAP controller, a 4-bit instruction register, and two test registers (a 1-bit bypass register and a 316-bit boundary scan register). The TAP controller consists of the following signals:

- TCK—A test clock input to synchronize the test logic.
- TMS—A test mode select input (with an internal pull-up resistor) that is sampled on the rising edge of TCK to sequence the TAP controller's state machine.
- TDI—A test data input (with an internal pull-up resistor) that is sampled on the rising edge of TCK.
- TDO—A three-stateable test data output that is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCK.
- $\overline{\text{TRST}}$—An asynchronous reset with an internal pull-up resistor that provides TAP controller initialization and other logic required by the standard.

An overview of the MPC801 scan chain implementation is illustrated in the figure below.

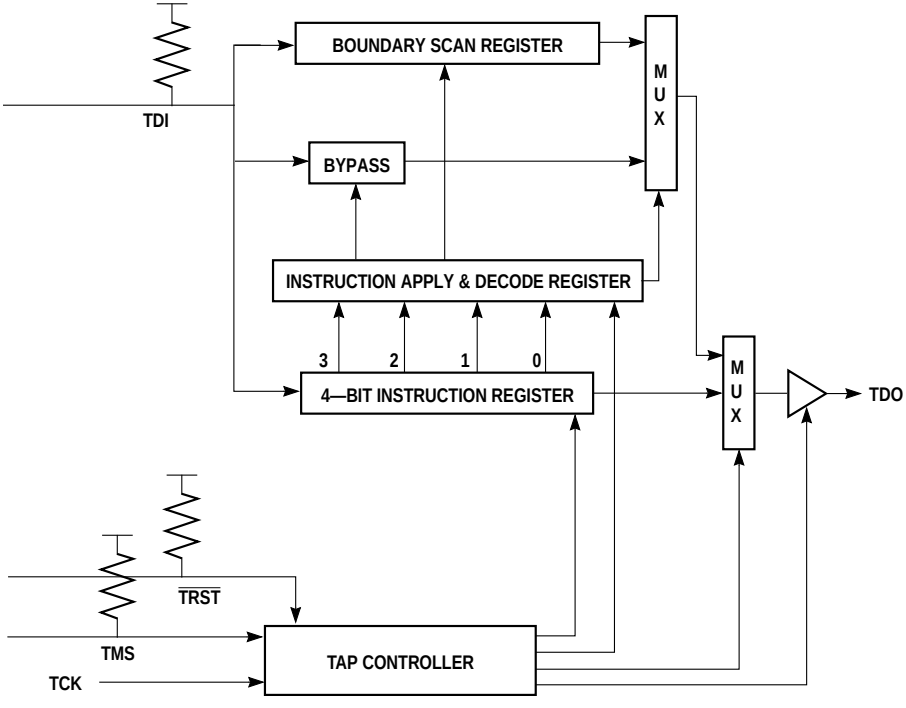


Figure 19-1. Test Logic Block Diagram

19.1 THE TAP CONTROLLER

The TAP controller is responsible for interpreting the sequence of logical values on the TMS signal. It is a synchronous state machine that controls the operation of the JTAG logic. The value shown adjacent to each bubble in the figure below represents the value of the TMS signal sampled on the rising edge of the TCK signal.

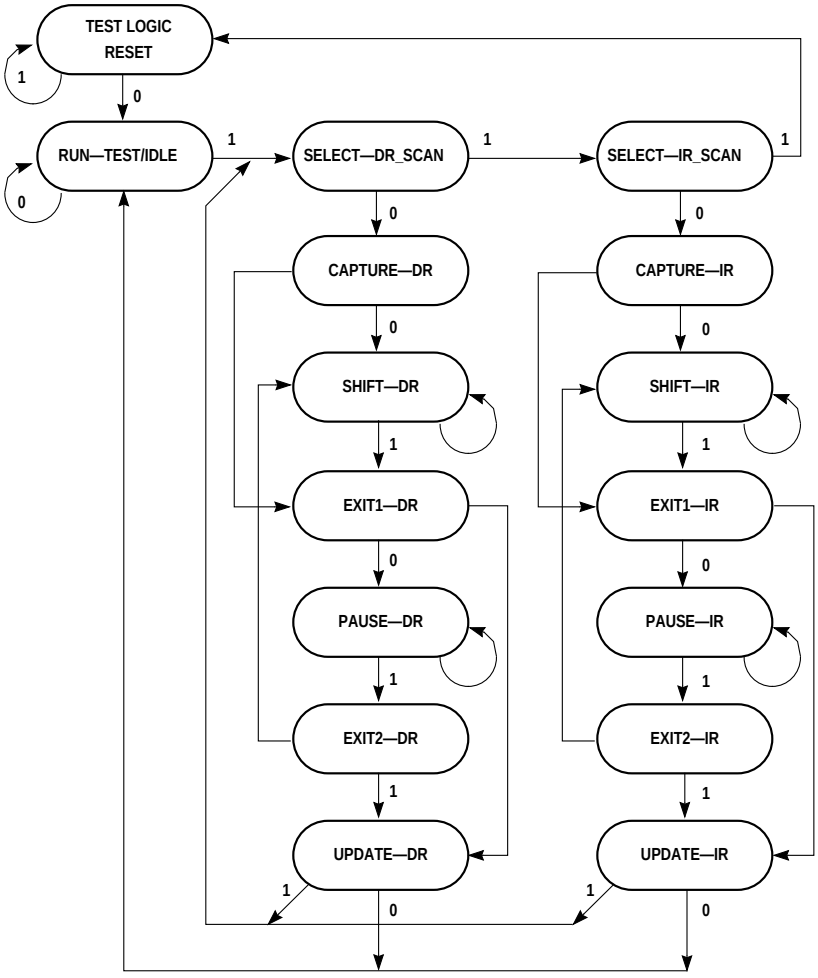


Figure 19-2. TAP Controller State Machine

19.2 THE BOUNDARY SCAN REGISTER

The MPC801 scan chain implementation has a 316-bit boundary scan register that contains bits for all device signal, clock pins, and associated control signals. However, the XTAL, EXTAL, and XFC pins are associated with analog signals and are not included in the boundary scan register. An IEEE-1149.1-compliant boundary scan register has been included on the MPC801. This 316-bit boundary scan register can be connected between the TDI and TDO pins 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. Figures 19-3 through 19-6 illustrate the various cell types.

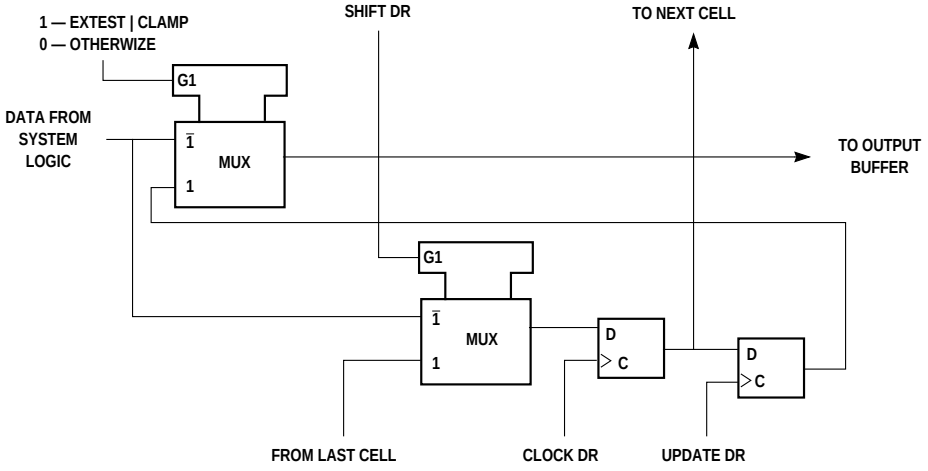


Figure 19-3. Output Pin Cell (O.Pin)

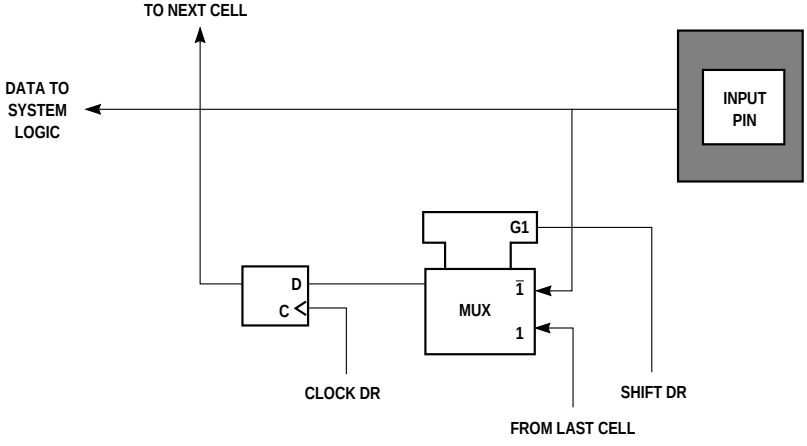


Figure 19-4. Observe-Only Input Pin Cell (I.Obs)

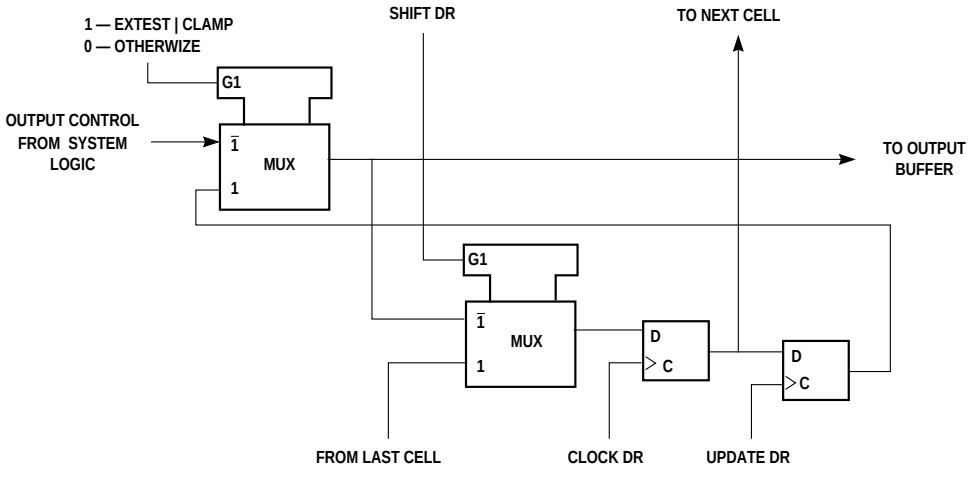


Figure 19-5. Output Control Cell (IO.CTL)

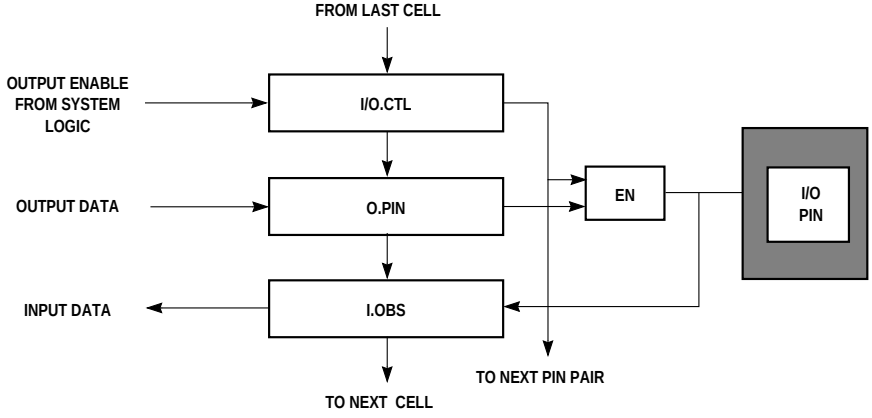


Figure 19-6. General Arrangement of Bidirectional Pin Cells

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 19-6 and these bits are controlled by the cell illustrated in Figure 19-5.

It is important to know the boundary scan bit order and the pins that are associated with them. The bit order starting with the TDO output and ending with the TDI input is shown in Table 19-1. 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 316. The second column references one of the three MPC801 cell types depicted in Figures 19-3 through 19-6 that describe the cell structure for each type. The third column lists the pin name for all pin-related cells and defines the name of the bidirectional control register bits. The fourth column lists the pin type and the last column indicates the associated boundary scan register control bit for the bidirectional output pins.

Table 19-1. Boundary Scan Bit Definition

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
0	I.OBS	PB[26]	I/O	—
1	O.PIN	PB[26]	I/O	g56.ctl
2	IO.CTL	G56.CTL	—	—
3	I.OBS	PB[27]	I/O	—
4	O.PIN	PB[27]	I/O	g57.ctl
5	IO.CTL	G57.CTL	—	—
6	I.OBS	PB[28]	I/O	—
7	O.PIN	PB[28]	I/O	g58.ctl
8	IO.CTL	G58.CTL	—	—
9	I.OBS	PB[29]	I/O	—
10	O.PIN	PB[29]	I/O	g59.ctl
11	IO.CTL	G59.CTL	—	—
12	I.OBS	PB[30]	I/O	—
13	O.PIN	PB[30]	I/O	g60.ctl
14	IO.CTL	G60.CTL	—	—
15	I.OBS	PB[31]	I/O	—
16	O.PIN	PB[31]	I/O	g61.ctl
17	IO.CTL	G61.CTL	—	—
18	I.OBS	A[6]	I/O	—
19	O.PIN	A[6]	I/O	g203.ctl
20	I.OBS	A[7]	I/O	—

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
21	O.PIN	A[7]	I/O	g203.ctl
22	IO.CTL	G203.CTL	—	—
23	I.OBS	A[8]	I/O	—
24	O.PIN	A[8]	I/O	g202.ctl
25	IO.CTL	G202.CTL	—	—
26	I.OBS	A[9]	I/O	—
27	O.PIN	A[9]	I/O	g202.ctl
28	I.OBS	A[10]	I/O	—
29	O.PIN	A[10]	I/O	g202.ctl
30	I.OBS	A[11]	I/O	—
31	O.PIN	A[11]	I/O	g202.ctl
32	I.OBS	A[12]	I/O	—
33	O.PIN	A[12]	I/O	g202.ctl
34	I.OBS	A[13]	I/O	—
35	O.PIN	A[13]	I/O	g202.ctl
36	I.OBS	A[14]	I/O	—
37	O.PIN	A[14]	I/O	g202.ctl
38	I.OBS	A[15]	I/O	—
39	O.PIN	A[15]	I/O	g202.ctl
40	I.OBS	A[16]	I/O	—
41	O.PIN	A[16]	I/O	g201.ctl
42	IO.CTL	G201.CTL	—	—
43	I.OBS	A[17]	I/O	—
44	O.PIN	A[17]	I/O	g201.ctl
45	I.OBS	A[27]	I/O	—
46	O.PIN	A[27]	I/O	g201.ctl
47	I.OBS	A[19]	I/O	—
48	O.PIN	A[19]	I/O	g201.ctl
49	I.OBS	A[20]	I/O	—

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
50	O.PIN	A[20]	I/O	g201.ctl
51	I.OBS	A[21]	I/O	—
52	O.PIN	A[21]	I/O	g201.ctl
53	I.OBS	A[29]	I/O	—
54	O.PIN	A[29]	I/O	g201.ctl
55	I.OBS	A[23]	I/O	—
56	O.PIN	A[23]	I/O	g201.ctl
57	I.OBS	A[24]	I/O	—
58	O.PIN	A[24]	I/O	g200.ctl
59	IO.CTL	G200.CTL	—	—
60	I.OBS	A[25]	I/O	—
61	O.PIN	A[25]	I/O	g200.ctl
62	I.OBS	A[30]	I/O	—
63	O.PIN	A[30]	I/O	g200.ctl
64	I.OBS	A[18]	I/O	—
65	O.PIN	A[18]	I/O	g200.ctl
66	I.OBS	A[28]	I/O	—
67	O.PIN	A[28]	I/O	g200.ctl
68	I.OBS	A[22]	I/O	—
69	O.PIN	A[22]	I/O	g200.ctl
70	I.OBS	A[26]	I/O	—
71	O.PIN	A[26]	I/O	g200.ctl
72	I.OBS	A[31]	I/O	—
73	O.PIN	A[31]	I/O	g200.ctl
74	I.OBS	TSIZ0	I/O	—
75	O.PIN	TSIZ0	I/O	g204.ctl
76	I.OBS	TSIZ1	I/O	—
77	O.PIN	TSIZ1	I/O	g204.ctl
78	IO.CTL	G204.CTL	—	—

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
79	I.OBS	SPARE1	I/O	—
80	O.PIN	SPARE1	I/O	g87.ctl
81	IO.CTL	G87.CTL	—	—
82	O.PIN	$\overline{WE0}$_B_$\overline{BSAB0}$_B	O	—
83	O.PIN	$\overline{WE1}$_B_$\overline{BSAB1}$_B	O	—
84	O.PIN	$\overline{WE2}$_B_$\overline{BSAB2}$_B	O	—
85	O.PIN	$\overline{WE3}$_B_$\overline{BSAB3}$_B	O	—
86	O.PIN	$\overline{GPLA0}$_B_$\overline{GPLB0}$_B	O	—
87	O.PIN	$\overline{OE}$_B_$\overline{GPLAB1}$_B	O	—
88	O.PIN	$\overline{GPLAB2}$_B_$\overline{CS2}$_B	O	—
89	O.PIN	$\overline{GPLAB3}$_B_$\overline{CS3}$_B	O	—
90	O.PIN	$\overline{CS4}$_B	O	—
91	O.PIN	$\overline{CS5}$_B	O	—
92	O.PIN	$\overline{CS6}$	O	—
93	O.PIN	$\overline{CS7}$	O	—
94	O.PIN	$\overline{CS3}$_B	O	—
95	O.PIN	$\overline{CS2}$_B	O	—
96	O.PIN	$\overline{CS1}$_B	O	—
97	O.PIN	$\overline{CS0}$_B	O	—
98	I.OBS	WR_B	I/O	—
99	O.PIN	WR_B	I/O	g96.ctl
100	IO.CTL	G96.CTL	—	—
101	I.OBS	$\overline{GPLB4}$_B_UPWAITB_B	I/O	—
102	O.PIN	$\overline{GPLB4}$_B_UPWAITB_B	I/O	g24.ctl
103	IO.CTL	G24.CTL	—	—
104	O.PIN	$\overline{GPLA5}$_B	O	—
105	I.OBS	$\overline{GPLA4}$_B_UPWAITA_A	I/O	—
106	O.PIN	$\overline{GPLA4}$_B_UPWAITA_A	I/O	g25.ctl
107	IO.CTL	G25.CTL	—	—

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
108	O.PIN	BDIP_B_GPLB5_B	O	—
109	I.OBS	BI_B	I/O	—
110	O.PIN	BI_B	I/O	g23.ctl
111	IO.CTL	G23.CTL	—	—
112	I.OBS	TA_B	I/O	—
113	O.PIN	TA_B	I/O	g22.ctl
114	IO.CTL	G22.CTL	—	—
115	I.OBS	TEA_B	I/O	—
116	O.PIN	TEA_B	I/O	g89.ctl
117	IO.CTL	G89.CTL	—	—
118	I.OBS	TS_B	I/O	—
119	O.PIN	TS_B	I/O	g97.ctl
120	IO.CTL	G97.CTL	—	—
121	I.OBS	BR_B	I/O	—
122	O.PIN	BR_B	I/O	g21.ctl
123	IO.CTL	G21.CTL	—	—
124	I.OBS	BG_B	I/O	—
125	O.PIN	BG_B	I/O	g20.ctl
126	IO.CTL	G20.CTL	—	—
127	I.OBS	BB_B	I/O	—
128	O.PIN	BB_B	I/O	g11.ctl
129	IO.CTL	G11.CTL	—	—
130	I.OBS	SPARE4	I/O	—
131	O.PIN	SPARE4	I/O	g86.ctl
132	IO.CTL	G86.CTL	—	—
133	I.OBS	FRZ_B_IRQ6_B	I/O	—
134	O.PIN	FRZ_B_IRQ6_B	I/O	g90.ctl
135	IO.CTL	G90.CTL	—	—
136	I.OBS	CR_B_IRQ3_B	I	—

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
137	I.OBS	$\overline{\text{BURST}}$_B	I/O	—
138	O.PIN	$\overline{\text{BURST}}$_B	I/O	g205.ctl
139	IO.CTL	G205.CTL	—	—
140	I.OBS	RSV_B_$\overline{\text{IRQ2}}$_B	I/O	—
141	O.PIN	RSV_B_$\overline{\text{IRQ2}}$_B	I/O	g17.ctl
142	IO.CTL	G17.CTL	—	—
143	I.OBS	LWP1_VF1	I/O	—
144	O.PIN	LWP1_VF1	I/O	g15.ctl
145	IO.CTL	G15.CTL	—	—
146	I.OBS	LWP0_VF0	I/O	—
147	O.PIN	LWP0_VF0	I/O	g150.ctl
148	IO.CTL	G150.CTL	—	—
149	I.OBS	IWP2_VF2	I/O	—
150	O.PIN	IWP2_VF2	I/O	g15.ctl
151	I.OBS	IWP1_VFLS1	I/O	—
152	O.PIN	IWP1_VFLS1	I/O	g150.ctl
153	I.OBS	IWP0_VFLS0	I/O	—
154	O.PIN	IWP0_VFLS0	I/O	g150.ctl
155	I.OBS	PTR_AT3	I/O	—
156	O.PIN	PTR_AT3	I/O	g13.ctl
157	IO.CTL	G13.CTL	—	—
158	I.OBS	AT2	I/O	—
159	O.PIN	AT2	I/O	g41.ctl
160	IO.CTL	G41.CTL	—	—
161	I.OBS	DSCK_AT1	I/O	—
162	O.PIN	DSCK_AT1	I/O	g16.ctl
163	IO.CTL	G16.CTL	—	—
164	I.OBS	DSDI_IRQ5_B	I/O	—
165	O.PIN	DSDI_IRQ5_B	I/O	g14.ctl

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
166	IO.CTL	G14.CTL	—	—
167	I.OBS	$\overline{KR}$_B_$\overline{IRQ4}$_B	I/O	—
168	O.PIN	$\overline{KR}$_B_$\overline{IRQ4}$_B	I/O	g95.ctl
169	IO.CTL	G95.CTL	—	—
170	I.OBS	$\overline{AS}$_B	I/O	—
171	O.PIN	$\overline{AS}$_B	I/O	g82.ctl
172	IO.CTL	G82.CTL	—	—
173	I.OBS	BADDR[30]	I/O	—
174	O.PIN	BADDR[30]	I/O	g83.ctl
175	IO.CTL	G83.CTL	—	—
176	I.OBS	MODCK1_$\overline{STS}$_B	I/O	—
177	O.PIN	MODCK1_$\overline{STS}$_B	I/O	g92.ctl
178	IO.CTL	G92.CTL	—	—
179	I.OBS	MODCK2_DSDO	I/O	—
180	O.PIN	MODCK2_DSDO	I/O	g91.ctl
181	IO.CTL	G91.CTL	—	—
182	I.OBS	BADDR[29]	I/O	—
183	O.PIN	BADDR[29]	I/O	g84.ctl
184	IO.CTL	G84.CTL	—	—
185	I.OBS	BADDR[28]	I/O	—
186	O.PIN	BADDR[28]	I/O	g85.ctl
187	IO.CTL	G85.CTL	—	—
188	I.OBS	CLK4IN	I	—
189	O.PIN	TEXP	O	—
190	I.OBS	$\overline{HRESET}$_B	I/O	—
191	O.PIN	$\overline{HRESET}$_B	I/O	g94.ctl
192	IO.CTL	G94.CTL	—	—
193	I.OBS	$\overline{SRESET}$_B	I/O	—
194	O.PIN	$\overline{SRESET}$_B	I/O	g93.ctl

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
195	IO.CTL	G93.CTL	—	—
196	I.OBS	RSTCONF_B	I	—
197	I.OBS	PORESET_B	I	—
198	O.PIN	CLKOUT	O	—
199	I.OBS	DP0_IRQ3_B	I/O	—
200	O.PIN	DP0_IRQ3_B	I/O	g18.ctl
201	I.OBS	DP3_IRQ6_B	I/O	—
202	O.PIN	DP3_IRQ6_B	I/O	g18.ctl
203	IO.CTL	G18.CTL	—	—
204	I.OBS	DP2_IRQ5_B	I/O	—
205	O.PIN	DP2_IRQ5_B	I/O	g18.ctl
206	I.OBS	DP1_IRQ4_B	I/O	—
207	O.PIN	DP1_IRQ4_B	I/O	g18.ctl
208	I.OBS	D[31]	I/O	—
209	O.PIN	D[31]	I/O	g103.ctl
210	IO.CTL	G103.CTL	—	—
211	I.OBS	D[30]	I/O	—
212	O.PIN	D[30]	I/O	g103.ctl
213	I.OBS	D[29]	I/O	—
214	O.PIN	D[29]	I/O	g103.ctl
215	I.OBS	D[28]	I/O	—
216	O.PIN	D[28]	I/O	g103.ctl
217	I.OBS	D[7]	I/O	—
218	O.PIN	D[7]	I/O	g103.ctl
219	I.OBS	D[26]	I/O	—
220	O.PIN	D[26]	I/O	g103.ctl
221	I.OBS	D[25]	I/O	—
222	O.PIN	D[25]	I/O	g103.ctl
223	I.OBS	D[24]	I/O	—

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
224	O.PIN	D[24]	I/O	g103.ctl
225	I.OBS	D[6]	I/O	—
226	O.PIN	D[6]	I/O	g102.ctl
227	IO.CTL	G102.CTL	—	—
228	I.OBS	D[22]	I/O	—
229	O.PIN	D[22]	I/O	g102.ctl
230	I.OBS	D[21]	I/O	—
231	O.PIN	D[21]	I/O	g102.ctl
232	I.OBS	D[20]	I/O	—
233	O.PIN	D[20]	I/O	g102.ctl
234	I.OBS	D[19]	I/O	—
235	O.PIN	D[19]	I/O	g102.ctl
236	I.OBS	D[18]	I/O	—
237	O.PIN	D[18]	I/O	g102.ctl
238	I.OBS	D[5]	I/O	—
239	O.PIN	D[5]	I/O	g102.ctl
240	I.OBS	D[16]	I/O	—
241	O.PIN	D[16]	I/O	g102.ctl
242	I.OBS	D[15]	I/O	—
243	O.PIN	D[15]	I/O	g101.ctl
244	IO.CTL	G101.CTL	—	—
245	I.OBS	D[14]	I/O	—
246	O.PIN	D[14]	I/O	g101.ctl
247	I.OBS	D[3]	I/O	—
248	O.PIN	D[3]	I/O	g101.ctl
249	I.OBS	D[2]	I/O	—
250	O.PIN	D[2]	I/O	g101.ctl
251	I.OBS	D[11]	I/O	—
252	O.PIN	D[11]	I/O	g101.ctl

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
253	I.OBS	D[10]	I/O	—
254	O.PIN	D[10]	I/O	g101.ctl
255	I.OBS	D[9]	I/O	—
256	O.PIN	D[9]	I/O	g101.ctl
257	I.OBS	D[1]	I/O	—
258	O.PIN	D[1]	I/O	g101.ctl
259	I.OBS	D[27]	I/O	—
260	O.PIN	D[27]	I/O	g100.ctl
261	IO.CTL	G100.CTL	—	—
262	I.OBS	D[23]	I/O	—
263	O.PIN	D[23]	I/O	g100.ctl
264	I.OBS	D[17]	I/O	—
265	O.PIN	D[17]	I/O	g100.ctl
266	I.OBS	D[4]	I/O	—
267	O.PIN	D[4]	I/O	g100.ctl
268	I.OBS	D[13]	I/O	—
269	O.PIN	D[13]	I/O	g100.ctl
270	I.OBS	D[12]	I/O	—
271	O.PIN	D[12]	I/O	g100.ctl
272	I.OBS	D[8]	I/O	—
273	O.PIN	D[8]	I/O	g100.ctl
274	I.OBS	D[0]	I/O	—
275	O.PIN	D[0]	I/O	g100.ctl
276	I.OBS	$\overline{\text{IRQ0}}$_B	I	—
277	I.OBS	$\overline{\text{IRQ1}}$_B	I	—
278	I.OBS	$\overline{\text{IRQ7}}$_B	I	—
279	I.OBS	SPARE3	I/O	—
280	O.PIN	SPARE3	I/O	g10.ctl
281	IO.CTL	G110.CTL	—	—

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
282	I.OBS	PB[16]	I/O	—
283	O.PIN	PB[16]	I/O	g40.ctl
284	IO.CTL	G40.CTL	—	g40.ctl
285	I.OBS	PB[17]	I/O	—
286	O.PIN	PB[17]	I/O	g47.ctl
287	IO.CTL	G47.CTL	—	g47.ctl
288	I.OBS	PB[18]	I/O	—
289	O.PIN	PB[18]	I/O	g48.ctl
290	IO.CTL	G48.CTL	—	g48.ctl
291	I.OBS	PB[19]	I/O	—
292	O.PIN	PB[19]	I/O	g49.ctl
293	IO.CTL	G49.CTL	—	g49.ctl
294	I.OBS	PB[20]	I/O	—
295	O.PIN	PB[20]	I/O	g50.ctl
296	IO.CTL	G50.CTL	—	g50.ctl
297	I.OBS	PB[21]	I/O	—
298	O.PIN	PB[21]	I/O	g51.ctl
299	IO.CTL	G51.CTL	—	g51.ctl
300	I.OBS	PB[22]	I/O	—
301	O.PIN	PB[22]	I/O	g52.ctl
302	IO.CTL	G52.CTL	—	g52.ctl
303	I.OBS	PB[23]	I/O	—
304	O.PIN	PB[23]	I/O	g53.ctl
305	IO.CTL	G53.CTL	—	g53.ctl
306	I.OBS	PB[24]	I/O	—
307	O.PIN	PB[24]	I/O	g54.ctl
308	IO.CTL	G54.CTL	—	g54.ctl
309	I.OBS	PB[25]	I/O	—
310	O.PIN	PB[25]	I/O	g55.ctl

Table 19-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
311	IO.CTL	G55.CTL	—	g55.ctl
312	I.OBS	SPARE2	I/O	—
313	O.PIN	SPARE2	I/O	g88.ctl
314	IO.CTL	G88.CTL	—	g88.ctl

19.3 THE INSTRUCTION REGISTER

The MPC801 JTAG implementation includes the public instructions (**extest**, **sample/ preload**, and **bypass**) and also supports the **clamp** instruction. One additional public instruction (**hi-z**) is capable of disabling all device output drivers. The MPC801 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 19-2.

Table 19-2. 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**
0	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. 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.

19.3.1 The External Test Instruction

The external test (**extest**) instruction selects the 316-bit boundary scan register and asserts an internal reset for the MPC801 system logic to force a known beginning internal state while performing external boundary scan operations. By using the TAP controller, the register is capable of scanning user-defined values into the output buffers, capturing values presented to the input pins, and controlling the output drive of three-stateable output or bidirectional pins. For more details on the function and use of **extest**, refer to the IEEE 1149.1 standard.

19.3.2 The sample/preload Instruction

The **sample/preload** instruction initializes the boundary scan register output cells before **extest** is selected. This initialization ensures that known data will appear on the outputs when entering the **extest** instruction. The **sample/preload** instruction also provides an opportunity to obtain a snapshot of system data and control signals.

NOTE

Since there is no internal synchronization between the TCK and CLKOUT pins, you must provide some form of external synchronization between the JTAG operation TCK frequency and system operation CLKOUT frequency to achieve meaningful results.

19.3.3 The bypass Instruction

The **bypass** instruction creates a shift register path from the TDI pin to the bypass register and, finally, to the TDO pin, thus circumventing the 316-bit boundary scan register. This instruction is used to enhance test efficiency when a component other than the MPC801 is the device being tested. It selects the single-bit bypass register as illustrated in Figure 19-7.

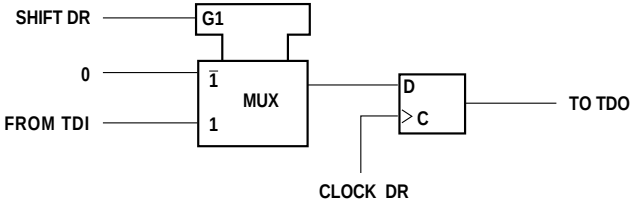


Figure 19-7. Bypass Register

When the bypass register is selected by the current instruction, the shift register stage is set to a logic zero on the rising edge of the TCK pin in the capture-DR controller state. Therefore, the first bit to be shifted out after selecting the bypass register is always a logic zero.

19.3.4 The clamp Instruction

The **clamp** instruction selects the single-bit bypass register as illustrated in Figure 19-7 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 by using the **sample/preload** instruction.

19.3.5 The hi-z Instruction

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) and the instruction selects the bypass register.

19.4 MPC801 RESTRICTIONS

The control afforded by the output enable signals using the boundary scan register and the **extest** instruction requires a compatible circuit board test environment to avoid device-destructive configurations. You must avoid situations in which the MPC801 output drivers are enabled into actively driven networks. The $MPC801_{CC}$ features a low-power stop mode. The interaction of the scan chain interface with low-power stop mode has the following characteristics:

- The TAP controller must be in the test-logic-reset state to either enter or remain in the low-power stop mode. Leaving the TAP controller in the test-logic-reset state negates its ability to achieve low-power, but does not otherwise affect device functionality.
- The TCK input is not disabled in low-power stop mode. To consume minimal power, the TCK input should be externally connected to V_{CC} or ground while in low-power or normal mode (nonscan chain).
- 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 $\overline{\text{PORESET}}$.

19.5 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 that the scan chain test logic must be kept transparent to the system logic by forcing TAP into the test-logic-reset controller state by using one of two methods. The first method is to connect the $\overline{\text{TRST}}$ pin to logic 0 (or one of the reset pins). The second method is to assure the TMS pin is sampled as a logic one for five consecutive TCK rising edges. If the TMS pin remains connected to V_{CC} or does not change state, then the TAP controller cannot leave the test-logic-reset state, regardless of the TCK pin's state.

19.6 MOTOROLA MPC801 BSDL DESCRIPTION

The BSDL file for exercising the scan chain logic on the MPC801 is available at the Motorola web site (`www.mot.com/netcomm`) in the MPCxxx Embedded PowerPC area.

SECTION 20
ELECTRICAL CHARACTERISTICS

This section contains detailed information on the power considerations, DC/AC electrical characteristics, and AC timing specifications for the MPC801.

NOTE

The MPC801 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.

20.1 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
Operating Temperature Range	T_A	0 to 70° or -40° to 85°	˚C
Storage Temperature Range	T_{STG}	-55° to +150°	˚C

NOTES

1. Functional operating conditions are given in **Section 20.3 Power Considerations**. Absolute maximum ratings are stress ratings only and functional operation at the maximum 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 to power-on as well as normal operation.
3. "5 Volt Friendly" inputs are inputs that tolerate 5 volts.

This device contains circuitry that protects against damage from high static voltage or electrical fields. However, it is advised that precautions be taken to avoid application of any voltages higher than the maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (either GND or V_{CC}).

20.2 THERMAL CHARACTERISTICS

CHARACTERISTIC	SYMBOL	VALUE	UNIT
Thermal Resistance for BGA	θ_{JC}	-30[1]	°C/W
	θ_{JA}	30[2]	°C/W
	θ_{JA}	15[3]	°C/W

NOTES:

1. Assumes natural convection, a multilayer board with thermal vias, 1 watt MPC801 dissipation, and a board temperature rise of 20°C above ambient.
2. Assumes natural convection, a multilayer board with thermal vias,1 watt MPC801 dissipation, and a board temperature rise of 10°C above ambient.

   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.

20.3 POWER CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from

$$T_J = T_A + (P_D \bullet q_{JA}) \quad (1)$$

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:

$$P_D = K \prod (T_J + 273\infty C) \quad (2)$$

Solving equations (1) and (2) for K gives

$$K = P_D \bullet (T_A + 273\infty C) + q_{JA} \bullet P_D^2 \quad (3)$$

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.

20.3.1 Layout Practices

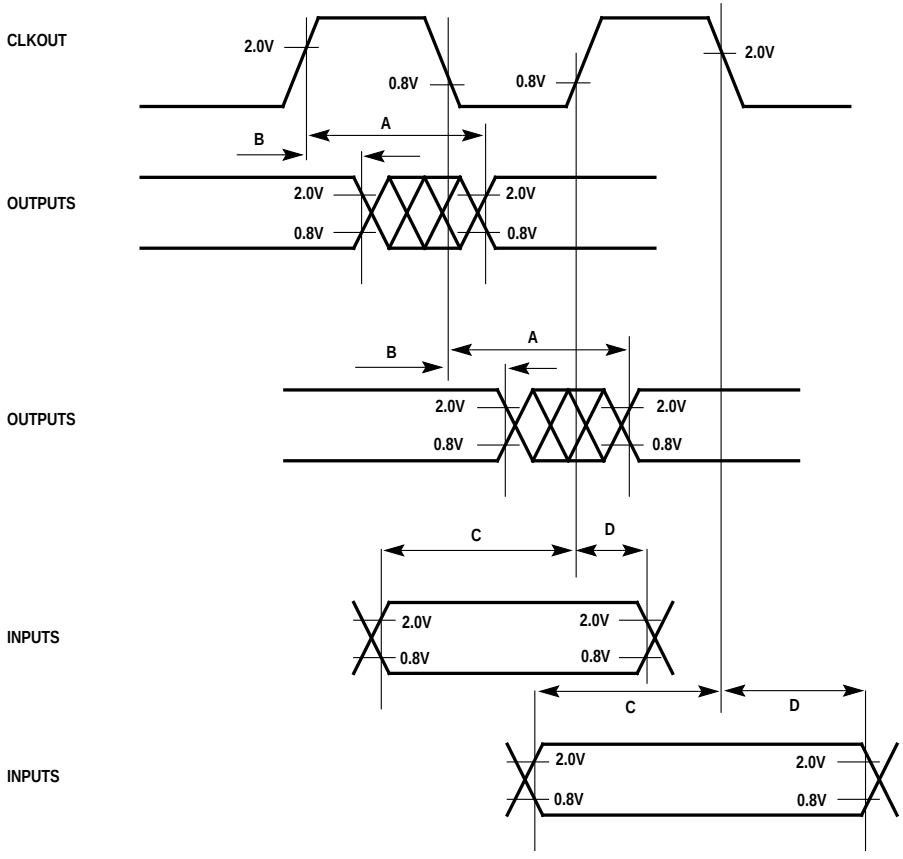
Each V_{CC} pin on the MPC801 should be provided with a low-impedance path to the board's supply. Each GND pin should likewise be provided with a low-impedance path to ground. The power supply pins drive distinct groups of logic on chip. The V_{CC} power supply should be bypassed to ground using at least four 0.1μF 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 MPC801 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.

20.4 DC ELECTRICAL SPECIFICATIONS (V_{CC} = 3.0 – 3.6V)

CHARACTERISTIC	SYMBOL	MIN	MAX	UNIT
Input High Voltage (all inputs except EXTAL and EXTCLK)	VIH	2.0	5.5	V
Input Low Voltage	VIL	GND	0.8	V
EXTAL, EXTCLK Input High Voltage	VIHC	0.7*(VCC)	VCC+0.3	V
Input Leakage Current, Vin = 5.5 V	I_{in}	—	TBD	mA
Hi-Z (Off State) Leakage Current, Vin = 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, IOH = -2.0 mA, VDDH = 3.0 V Except XTAL, XFC, and Open-Drain Pins	VOH	2.4	—	V
Output Low Voltage IOL = 2.0 mA CLKOUT IOL = 3.2 mA A[6:31], TSIZ0, TSIZ1, D[0:31], DP[0:3]/IRQ[3:6], RD/WR, BURST, RSV/IRQ2, IWP[0:1]/VFLS[0:1], AT2, IWP2/VF2, IWP0/VF0, LWP1/VF1, DSDI/IRQ5, PTR/AT3, SPISEL/PB31, SPICLK/PB30, SPIMOSI/PB29, SPIMISO/PB28, I2CSDA/PB27, I2CSCL/PB26, UGPIO1/PB25, UGPIO2/PB24, UCTS1/PB23, UCTS2/PB22, URXD1/PB21, URXD2/PB20, URTS1/PB19, URTS2/PB18, UTXD1/PB17, UTXD2/PB16 IOL = 5.3 mA BDIP/GPLB(5), BR, BG, FRZ/IRQ6, CS[0:5], CS(6), CS(7), WE0/BS_AB0, WE1/BS_AB1, WE2/BS_AB2, WE3/BS_AB3, GPLA0/GPLB0, OE/GPLA1/GPLB1, GPLA[2:3]/GPLB[2:3]/CS[2:3], UPWAITA/GPLA4, UPWAITB/GPLB4, GPLA5, DSCK/AT1, MODCK1/STS, MODCK2/DSDO , BADDR[28:30] IOL = 8.9 mA TS, TA, TEA, BI, BB, HRESET, SRESET	VOL	—	0.5	V

20.5 MPC801 AC ELECTRICAL SPECIFICATIONS CONTROL TIMING



A = Maximum Output Delay Specification
B = Minimum Output Hold Time
C = Minimum Input Setup Time Specification
D = Minimum Input Hold Time Specification

Table 20-1. Bus Operation Timing

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 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	—	ns
B7	CLKOUT To A(6:31), RD/$\overline{WR}$, $\overline{BURST}$, D(0:31), DP(0:3) Invalid	0.25TC + 1	10	—	5	—	ns
B8	CLKOUT To TSIZ(0:1),$\overline{RSV}$, AT(0:3),$\overline{BDIP}$, $\overline{PTR}$, BADDR(28:30) Invalid	0.25TC + 1	10	—	5	—	ns
B9	CLKOUT To $\overline{BR}$, $\overline{BG}$, FRZ, VFLS(0:1), VF(0:2), IWP(0:2), LWP(0:1), $\overline{STS}$ Invalid [1]	0.25TC + 1	10	—	5	—	ns
B10	CLKOUT To A(6:31), RD/$\overline{WR}$, $\overline{BURST}$, D(0:31), DP(0:3) Valid	0.25TC + TCC	10	19	—	13	ns
B11	CLKOUT To TSIZ(0:1),$\overline{RSV}$, AT(0:3), $\overline{BDIP}$, PTR Valid	0.25TC + TCC	10	19	—	13	ns
B12	CLKOUT To $\overline{BR}$, $\overline{BG}$, VFLS(0:1), VF(0:2), IWP(0:2), FRZ, LWP(0:1), $\overline{STS}$ Valid. [1]	0.25TC + TCC	10	19	—	13	ns
B13	CLKOUT To A(6:31), RD/$\overline{WR}$, $\overline{BURST}$, D(0:31), TSIZ(0:1), RSV, AT(0:3), PTR Hi-Z	0.25TC + TCC	10	19	5	13	ns
B14	CLKOUT To $\overline{TS}$, $\overline{BB}$ Assertion	0.25TC + TCC	10	19	5	13	ns
B15	CLKOUT To $\overline{TA}$, $\overline{BI}$ Assertion (When Driven By The Memory Controller)		—	11	—	10	ns
B16	CLKOUT To $\overline{TS}$, $\overline{BB}$ Negation	0.25TC + TCC	10	19	5	13	ns
B17	CLKOUT To $\overline{TA}$, $\overline{BI}$ Negation (When Driven By The Memory Controller)		—	11	—	11	ns
B18	CLKOUT To $\overline{TS}$, $\overline{BB}$ Hi-Z	0.25TC +14	10	24	5	21	ns
B19	CLKOUT To $\overline{TA}$, $\overline{BI}$ Hi-Z (When Driven By The Memory Controller)		—	15	—	15	ns
B20	CLKOUT To $\overline{TEA}$ Assertion		—	11	—	11	ns
B21	CLKOUT To $\overline{TEA}$ Hi-Z		—	15	—	15	ns
B22a	CLKOUT Falling Edge to $\overline{CS}$ Asserted-GPCM-ACS=10, TRLX=0	TCC + 1	—	10	—	8	ns
B22b	CLKOUT Falling Edge to $\overline{CS}$ Asserted-GPCM-ACS=11, TRLX=0, EBDF=1	0.25TC + TCC + 1	10	20	5	13	ns
B22c	CLKOUT Falling Edge to $\overline{CS}$ Asserted-GPCM-ACS=11, TRLX=0, EBDF=1	0.375TC + TCC + 1	14	25	7	16	ns
B22	$\overline{TA}$, $\overline{BI}$, $\overline{BB}$, $\overline{BG}$, $\overline{BR}$ Valid To CLKOUT (Setup Time) [2 3]		9	—	7	—	ns

Table 20-1. Bus Operation Timing (Continued)

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
B22a	$\overline{TEA}$, $\overline{KR}$, $\overline{RETRY}$, $\overline{CR}$ Valid To CLKOUT (Setup Time)		11	—	10	—	ns
B23	CLKOUT To $\overline{TA}$, $\overline{TEA}$, $\overline{BI}$, $\overline{BB}$, $\overline{BG}$, $\overline{BR}$ Valid (Hold Time).[2]		2	—	2	—	ns
B23a	CLKOUT To $\overline{KR}$, $\overline{RETRY}$, $\overline{CR}$ Valid (Hold Time)		2	—	2	—	ns
B24	D(0:31), DP(0:3) Valid To CLKOUT Rising Edge (Setup Time).[4]		6	—	6	—	ns
B25	CLKOUT Rising Edge To D(0:31), DP(0:3) Valid (Hold Time).[4]		2	—	1	—	ns
B26	D(0:31), DP(0:3) Valid To CLKOUT Falling Edge (Setup Time)[5]		4	—	4	—	ns
B27	CLKOUT Falling Edge To D(0:31), DP(0:3) Valid (Hold Time)[5]		2	—	2	—	ns
B28a	CLKOUT Falling Edge To $\overline{WE}$(0:3) Negated-GPCM-Write Access TRLX=0, CSNT=1, EBDF=0	0.25TC + TCC + 1	10	20	5	13	ns
B28b	CLKOUT Falling Edge To $\overline{CS}$ Negated-GPCM-Write Access TRLX=0, CSNT=1, ACS=10, or ACS=11, EBDF=0	0.25TC + TCC + 1	—	20	—	13	ns
B28c	CLKOUT Falling Edge To $\overline{WE}$(0:3) Negated-GPCM-Write Access TRLX=0, CSNT=1, EBDF=0	0.375TC + TCC + 1	14	25	7	16	ns
B28d	CLKOUT Falling Edge To $\overline{CS}$ Negated-GPCM-Write Access TRLX=0, CSNT=1, ACS=10, or ACS=11, EBDF=0	0.25TC + TCC + 1	—	25	—	16	ns
B28	CLKOUT Rising Edge To $\overline{CS}$ Asserted -GPCM-ACS = 00	0.25TC+TCC+1	10	20	5	13	ns
B29a	$\overline{WE}$(0:3) Negated To D(0:31), DP(0:3) High Z-GPCM- Write Access, TRLX = '0', CSNT = '1', EBDF=0		18	—	8	—	ns
B29a	CLKOUT Falling Edge To $\overline{CS}$ Asserted -GPCM-ACS = 10, TRLX = 0	TCC + 1	—	10	—	8	ns
B29b	CLKOUT Falling Edge To $\overline{CS}$ Asserted -GPCM-ACS = 11, TRLX = 0	0.25TC+TCC+1	10	20	5	13	ns
B29b	$\overline{CS}$ Negated To D(0:31), DP(0:3) High Z-GPCM-Write Access, ACS=00, TRLX=0, CSNT=0		8	—	3	—	ns
B29c	$\overline{CS}$ Negated To D(0:31), DP(0:3) High Z-GPCM-Write Access, TRLX=0, CSNT=1, ACS=10 or ACS=11, EBDF=0		18	—	8	—	ns
B29d	$\overline{WE}$(0:3) Negated To D(0:31), DP(0:3) High Z-GPCM- Write Access,TRLX = '1', CSNT = '1', EBDF=0		58	—	28	—	ns
B29e	$\overline{CS}$ Negated To D(0:31), DP(0:3) High Z-GPCM-Write Access, TRLX=1, CSNT=1, ACS=10 or ACS=11, EBDF=0		58	—	28	—	ns
B29f	$\overline{WE}$(0:3) Negated To D(0:31), DP(0:3) High Z-GPCM- Write Access, TRLX = '0', CSNT = '1', EBDF=1		12	—	5	—	ns

Table 20-1. Bus Operation Timing (Continued)

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
B29g	$\overline{CS}$ Negated To D(0:31), DP(0:3) Hi-Z-GPCM-Write Access, TRLX=0, CSNT=1, ACS=10 or ACS=11, EBDF=1		52	—	24	—	ns
B29h	$\overline{WE}$(0:3) Negated To D(0:31), DP(0:3) Hi-Z-GPCM-Write Access,TRLX = '1', CSNT = '1', EBDF=1		52	—	24	—	ns
B29i	$\overline{CS}$ Negated To D(0:31), DP(0:3) Hi-Z-GPCM-Write Access, TRLX=1, CSNT=1, ACS=10 or ACS=11, EBDF=1		—	—	—	—	—
B29	CLKOUT Rising Edge To $\overline{CS}$ Negated -GPCM-Read Access	TCC + 1	3	10	2	8	ns
B30	A(6:31) To $\overline{CS}$ Asserted -GPCM- ACS = 10, TRLX = 0		8	—	3	—	ns
B30a	A(6:31) To $\overline{CS}$ Asserted -GPCM- ACS = 11, TRLX = 0		18	—	8	—	ns
B31	CLKOUT Rising Edge To $\overline{OE}$, $\overline{WE}$(0:3) Asserted		—	11	—	9	ns
B32	CLKOUT Rising Edge To $\overline{OE}$ Negated		3	11	2	9	ns
B33	A(6:31) To $\overline{CS}$ Asserted -GPCM- ACS = 10, TRLX = 1		48	—	23	—	ns
B33a	A(6:31) To $\overline{CS}$ Asserted -GPCM- ACS = 11, TRLX = 1		58	—	28	—	ns
B34	CLKOUT Rising Edge To $\overline{WE}$(0:3) Negated -GPCM-Write Access CSNT = '0'		—	11	—	9	ns
B34a	CLKOUT Falling Edge To $\overline{WE}$(0:3) Negated -GPCM-Write Access TRLX = '0', CSNT = '1'.	0.25TC + TCC + 1	10	20	5	13	ns
B34b	CLKOUT Falling Edge To $\overline{CS}$ Negated-GPCM-Write Access TRLX = '0', CSNT = '1', ACS = '10' Or ACS='11'	0.25TC + TCC + 1	—	20	—	13	ns
B35	$\overline{WE}$(0:3) Negated To D(0:31), DP(0:3) Hi-Z-GPCM-Write Access, CSNT = '0'		8	—	3	—	ns
B35a	$\overline{WE}$(0:3) Negated To D(0:31), DP(0:3) Hi-Z-GPCM-Write Access,TRLX = '0', CSNT = '1'		18	—	8	—	ns
B35b	$\overline{CS}$ Negated To D(0:31), DP(0:3) Hi-Z -GPCM-Write Access, ACS = '00', TRLX = '0' & CSNT = '0'		8	—	3	—	ns
B35c	$\overline{CS}$ Negated To D(0:31), DP(0:3) Hi-Z -GPCM-Write Access,TRLX = '0', CSNT = '1', ACS = '10' or ACS='11'		18	—	8	—	ns
B35d	$\overline{WE}$(0:3) Negated To D(0:31), DP(0:3) Hi-Z -GPCM-Write Access,TRLX = '1', CSNT = '1'		58	—	28	—	ns
B35e	$\overline{CS}$ Negated To D(0:31), DP(0:3) Hi-Z -GPCM-Write Access,TRLX = '1', CSNT = '1', ACS = '10' Or ACS='11'		58	—	28	—	ns
B36	$\overline{CS}$, $\overline{WE}$(0:3) Negated To A(6:31) Invalid -GPCM-Write Access.[6]		8	—	3	—	ns

Table 20-1. Bus Operation Timing (Continued)

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
B36a	$\overline{WE}$(0:3) Negated To A(6:31) Invalid -GPCM- Write Access, TRLX='0', CSNT = '1'. $\overline{CS}$ Negated To A(6:31) Invalid -GPCM- Write Access, TRLX='0', CSNT = '1', ACS = 10, ACS = ='11'		18	—	8	—	ns
B36b	$\overline{WE}$(0:3) Negated To A(6:31) Invalid -GPCM- Write Access, TRLX='1', CSNT = '1'. $\overline{CS}$ Negated To A(6:31) Invalid -GPCM- Write Access, TRLX='1', CSNT = '1', ACS = 10, ACS = ='11'		58	—	28	—	ns
B37	CLKOUT Falling Edge To $\overline{CS}$ Valid–As Requested By Control Bit CST4 In The Corresponding Word In The UPM	TCC + 1	—	10	—	8	ns
B37a	CLKOUT Falling Edge To $\overline{CS}$ Valid–As Requested By Control Bit CST1 In The Corresponding Word In The UPM	0.25TC + TCC + 1	10	20	5	13	ns
B37b	CLKOUT Rising Edge To $\overline{CS}$ Valid–As Requested By Control Bit CST2 In The Corresponding Word In The UPM	TCC + 1	—	10	—	8	ns
B37c	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
B38	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
B38a	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
B38b	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
B38c	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
B39	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
B39a	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
B40	A(6:31) To $\overline{CS}$ Valid–As Requested By Control Bit CST4 In The Corresponding Word In The UPM		8	—	3	—	ns
B40a	A(6:31) To $\overline{CS}$ Valid–As Requested By Control Bit CST1 In The Corresponding Word In The UPM		18	—	8	—	ns
B40b	A(6:31) To $\overline{CS}$ Valid–As Requested By Control Bit CST2 In The Corresponding Word In The UPM		28	—	13	—	ns
B41	A(6:31) To $\overline{BS}$ Valid–As Requested By Control Bit BST4 In The Corresponding Word In The UPM		8	—	3	—	ns
B41a	A(6:31) To $\overline{BS}$ Valid–As Requested By Control Bit BST1 In The Corresponding Word In The UPM		18	—	8	—	ns

Table 20-1. Bus Operation Timing (Continued)

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
B41b	A(6:31) To $\overline{BS}$ Valid–As Requested By Control Bit BST2 In The Corresponding Word In The UPM		28	—	13	—	ns
B42	A(6:31) To $\overline{GPL}$ Valid–As Requested By Control Bit GxT4 In The Corresponding Word In The UPM		8	—	3	—	ns
B43	UPWAIT Valid To CLKOUT Falling Edge [7]		6	—	—	—	ns
B44	CLKOUT Falling Edge To UPWAIT Valid [7]		1	—	—	—	ns
B45	$\overline{AS}$ Valid To CLKOUT Rising Edge		9	—	7	—	ns
B46	A(6:31), TSIZ(0:1), RD/$\overline{WR}$, $\overline{BURST}$, Valid To CLKOUT Rising Edge		9	—	7	—	ns
B47	$\overline{TS}$ Valid To CLKOUT Rising Edge (Setup Time)		9	—	7	—	ns
B48	CLKOUT Rising Edge To $\overline{TS}$ Valid (Hold Time)		2	—	2	—	ns
B49	$\overline{AS}$ Negation To Memory Controller Signals Negation		—	TBD	—	TBD	ns

1. The timing for $\overline{BR}$ output is relevant when the MPC801 is selected to operate with an external bus arbiter. The timing for $\overline{BG}$ output is relevant when the MPC801 is selected to operate with an 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 drives them.
3. The timing required for $\overline{BR}$ input is relevant when the MPC801 is selected to operate with an internal bus arbiter. The timing for $\overline{BG}$ input is relevant when the MPB801 is selected to operate with an external bus arbiter.
4. The D[0:31] and DP[0:3] input timings B18 and B19 refer 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 B21 refer to the falling edge of the CLKOUT. This timing is valid only under the control of the UPM in the memory controller.
6. The timing B30 refers to $\overline{CS}$ when ACS = '00' and to $\overline{WE}$[0:3] when CSNT = '0'.
7. The UPWAIT signal is considered asynchronous to the CLKOUT and is synchronized internally. The timings specified in B37 and B38 are specified to enable the freeze of the UPM output signals.
8. The $\overline{AS}$ signal is considered asynchronous to the CLKOUT. The B39 timing is specified to allow the behavior illustrated in Figure 8-18.

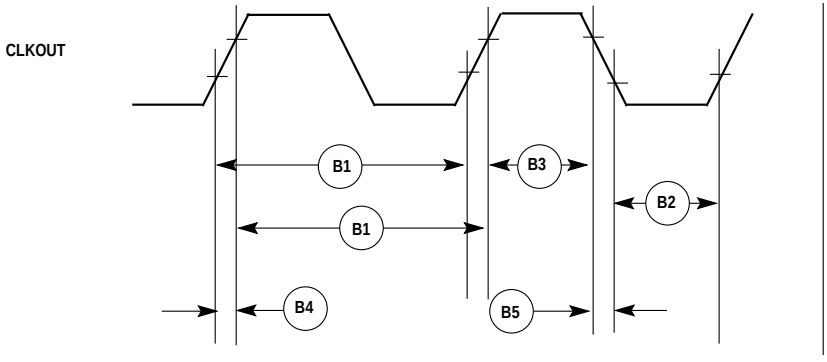


Figure 20-1. External Clock Timing Diagram

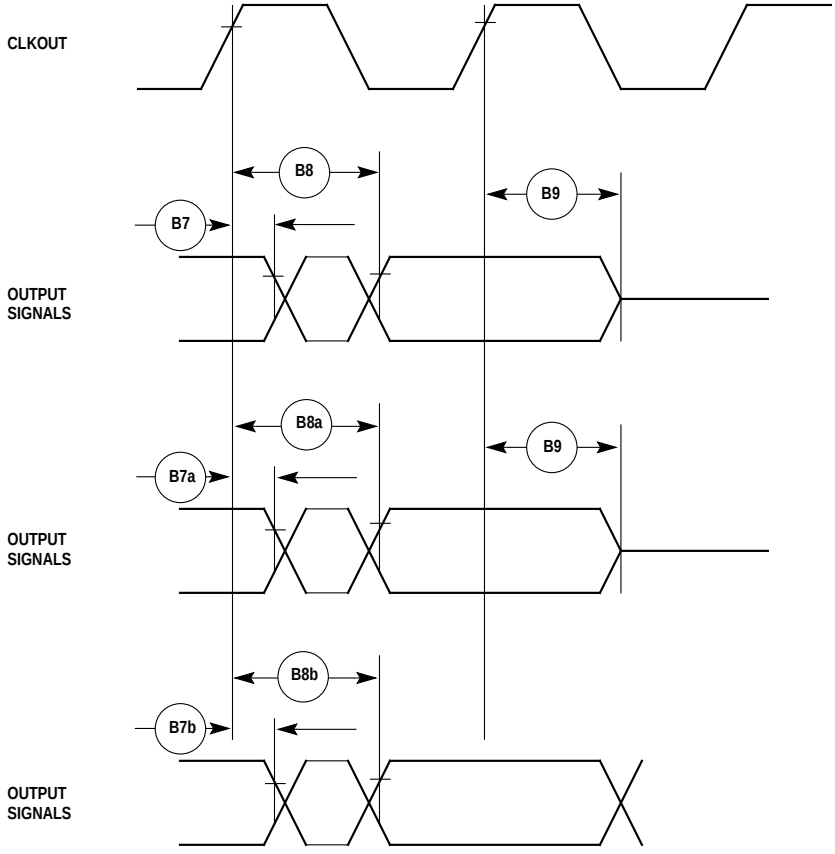


Figure 20-2. Synchronous Output Signals Timing Diagram

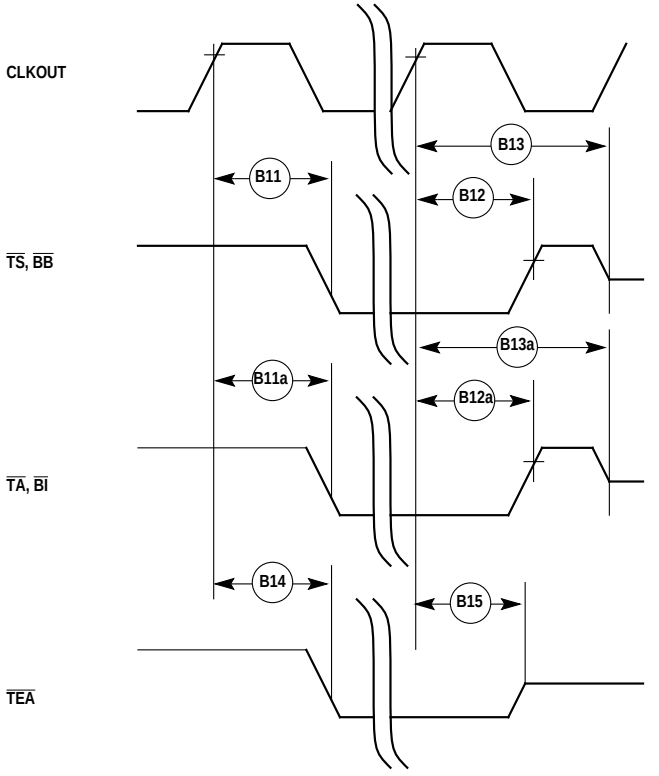


Figure 20-3. Synchronous Active Pull-Up And Open-Drain Outputs Signals Timing Diagram

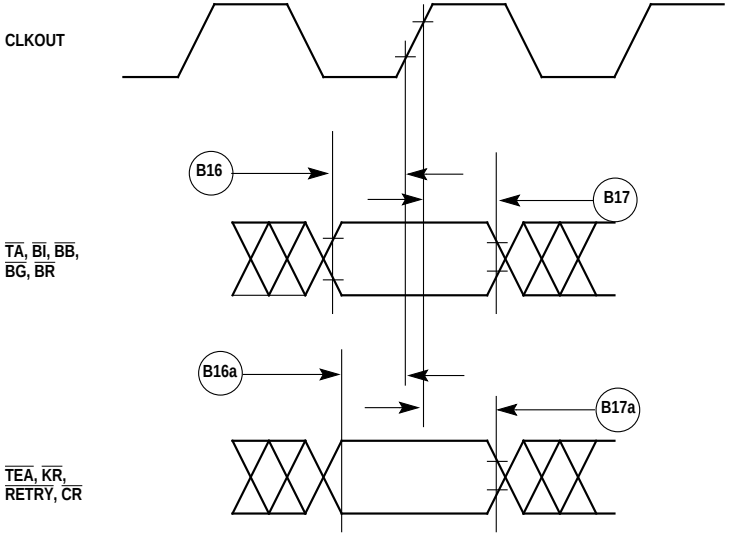


Figure 20-4. Synchronous Input Signals Timing Diagram

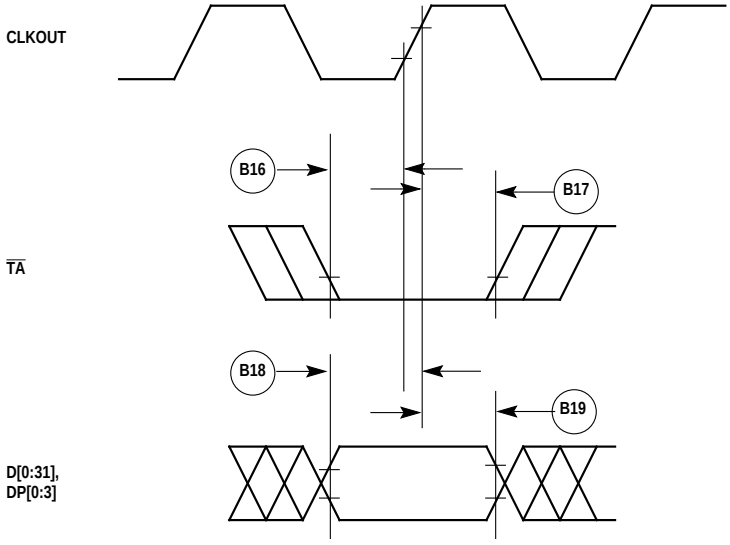


Figure 20-5. Input Data In Normal Case Timing Diagram

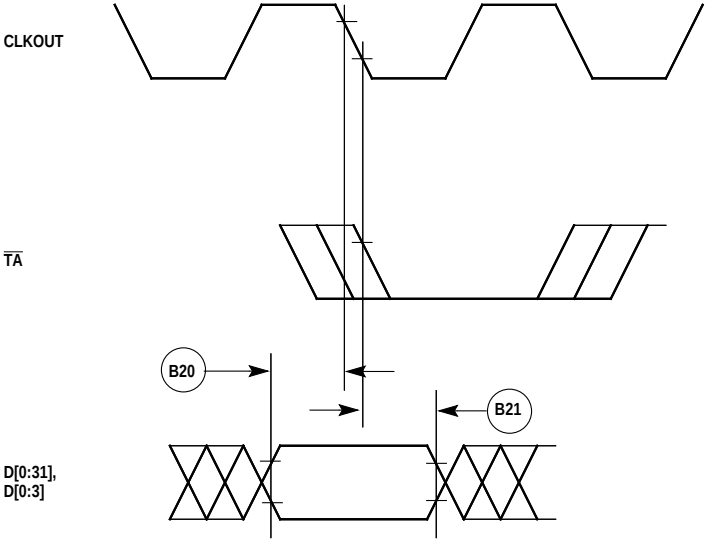


Figure 20-6. Input Data Timing When Controlled By The UPM In The Memory Controller

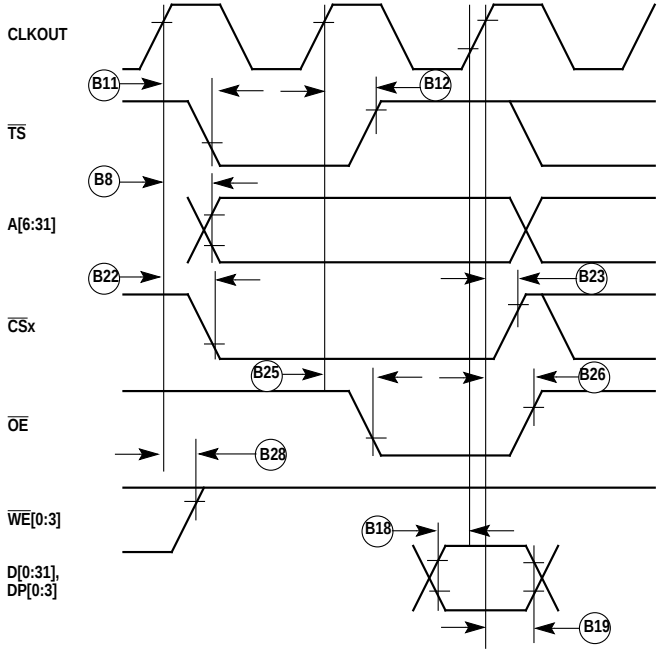


Figure 20-7. External Bus Read Timing Diagram (GPCM Controlled–ACS = ‘00’)

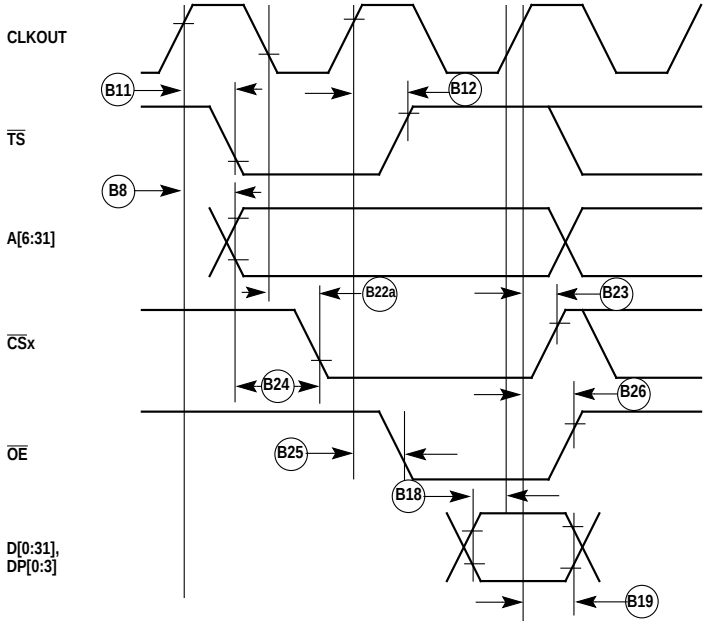


Figure 20-8. External Bus Read Timing Diagram (GPCM Controlled–TRLX = ‘0’ ACS = ‘10’)

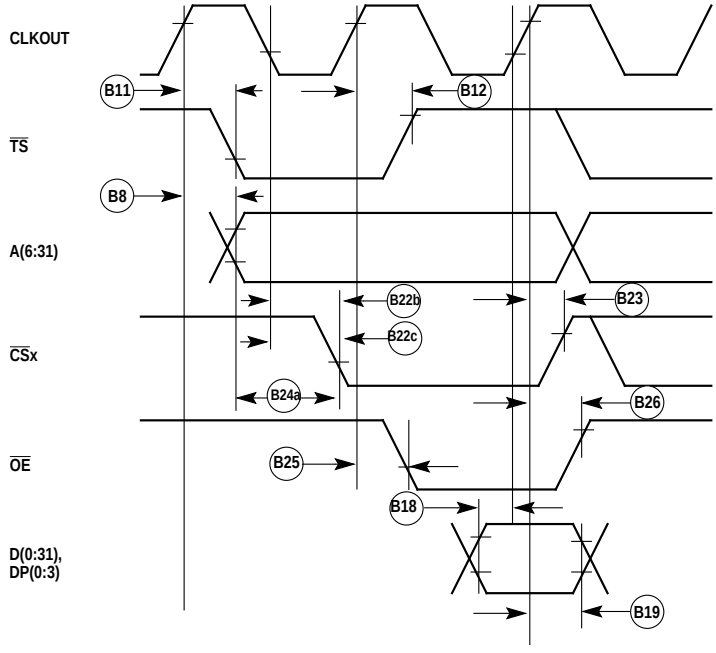


Figure 20-9. External Bus Read Timing Diagram (GPCM Controlled–TRLX = ‘0’ ACS = ‘11’)

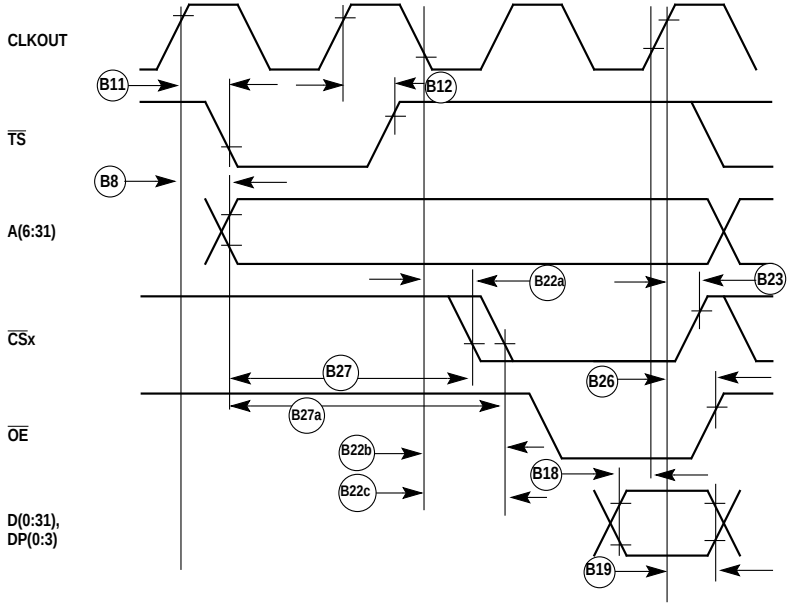


Figure 20-10. External Bus Read Timing Diagram (GPCM Controlled–TRLX = ‘1’, ACS = ‘10’, ACS = ‘11’)

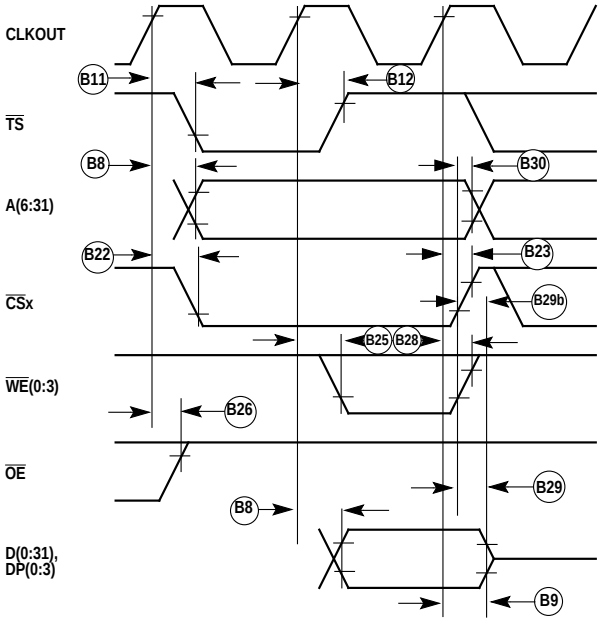


Figure 20-11. External Bus Write Timing Diagram (GPCM Controlled–TRLX = ‘0’, CSNT = ‘0’)

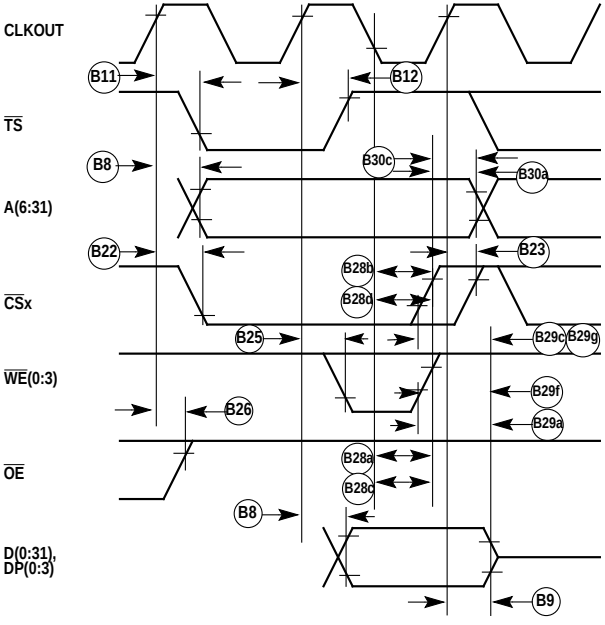


Figure 20-12. External Bus Write Timing Diagram (GPCM Controlled–TRLX = ‘0’, CSNT = ‘1’)

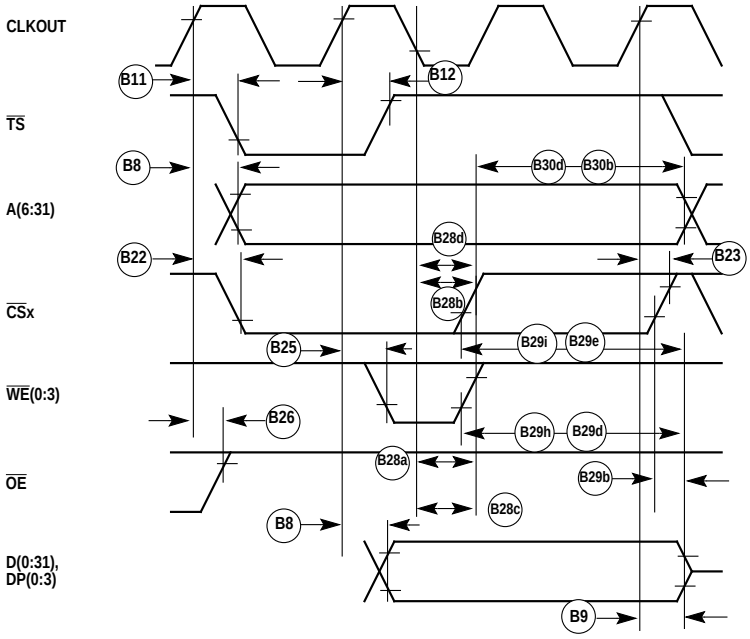


Figure 20-13. External Bus Write Timing Diagram (GPCM Controlled–TRLX = '1', CSNT = '1')

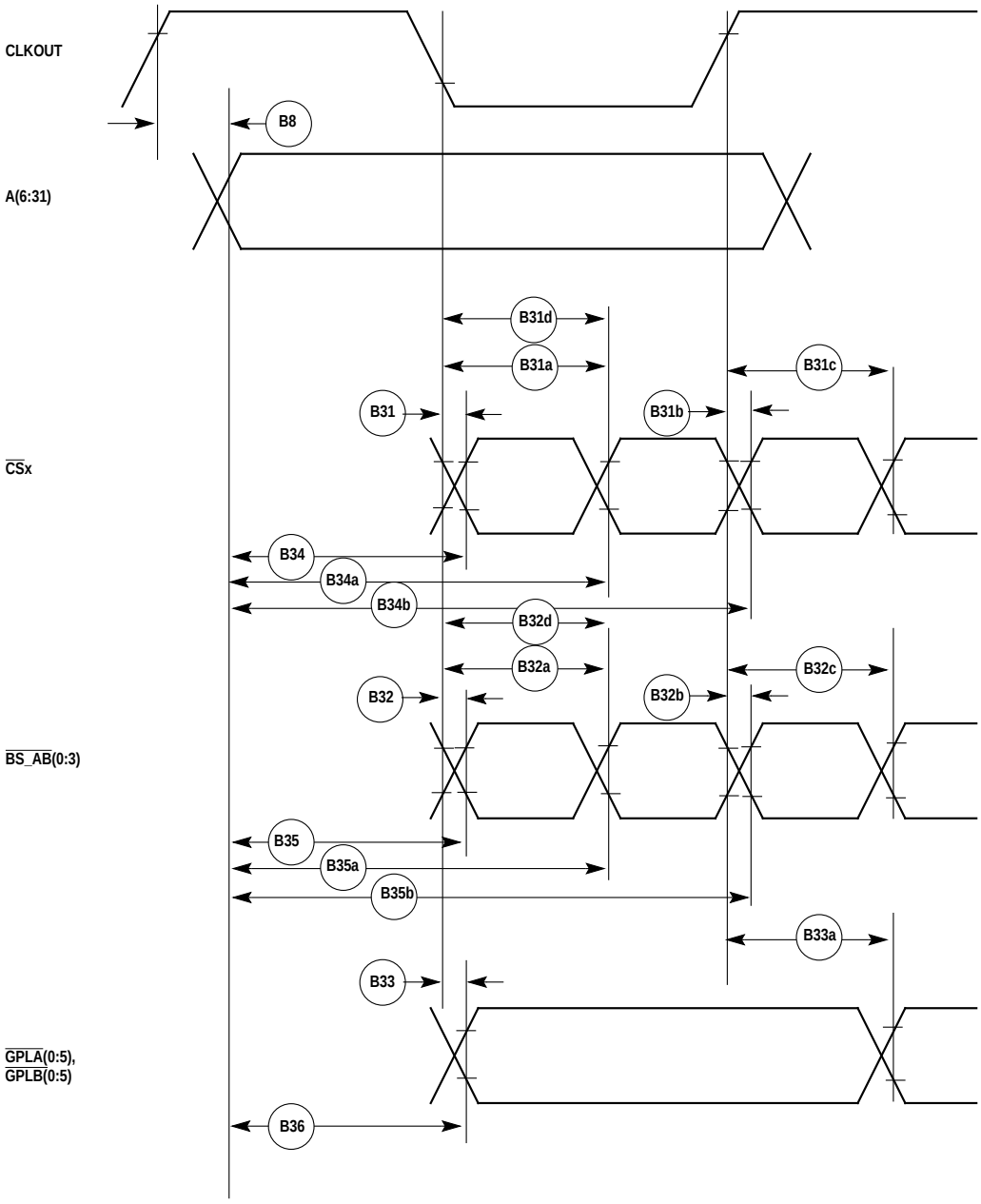


Figure 20-14. External Bus Timing Diagram (UPM Controlled Signals)

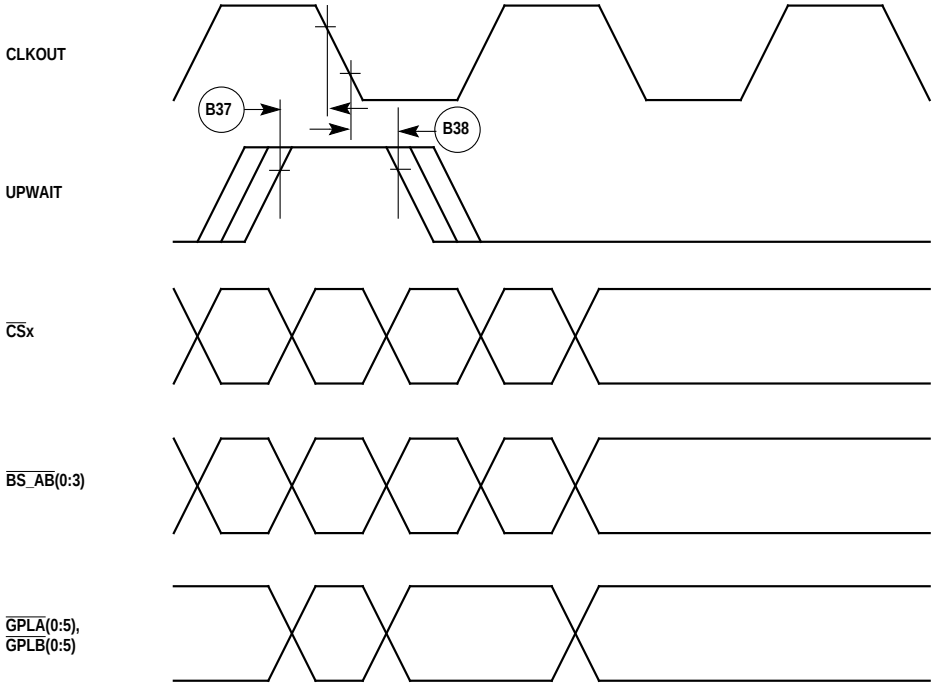


Figure 20-15. Asynchronous UPWAIT Asserted Detection In UPM Handled Cycles Timing Diagram

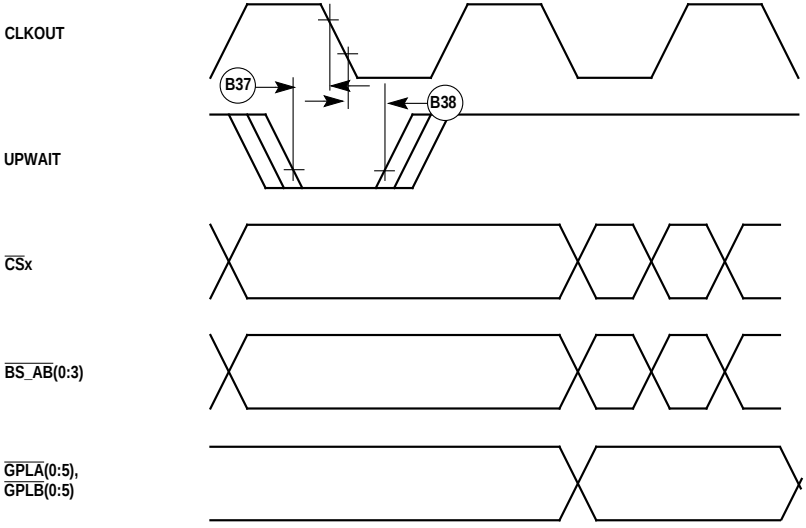


Figure 20-16. Asynchronous UPWAIT Negated Detection In UPM Handled Cycles Timing Diagram

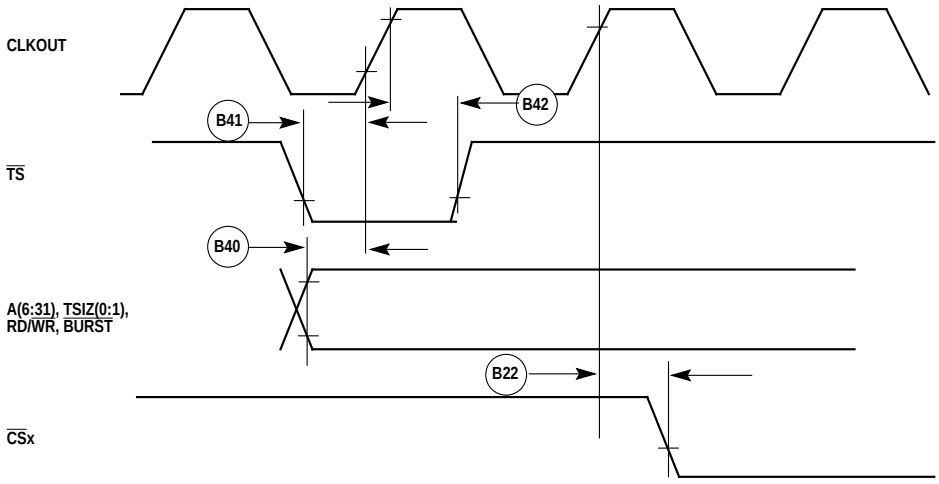


Figure 20-17. Synchronous External Master Access Timing Diagram (GPCM Handled ACS = ‘00’)

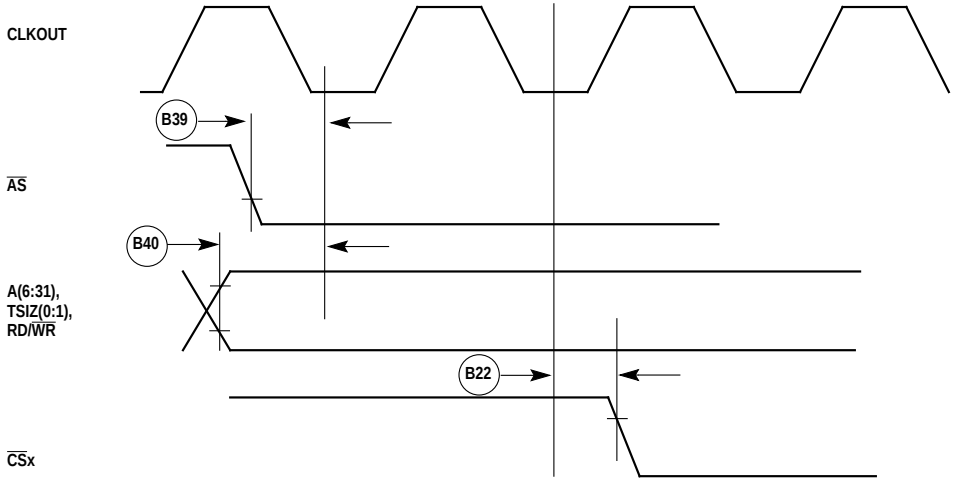


Figure 20-18. Asynchronous External Master Memory Access Timing Diagram (GPCM Controlled–ACS = ’00’)

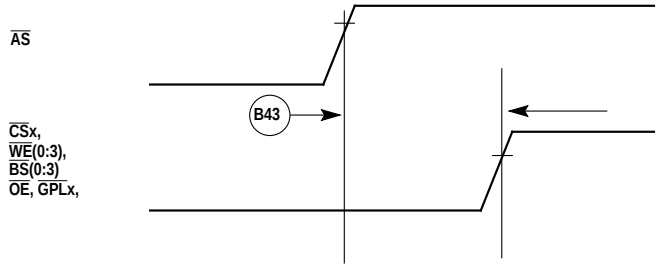


Figure 20-19. Asynchronous External Master Timing Diagram (Control Signals Negation Time)

Table 20-2. Interrupt Timing

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 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

1. The timings I39 and I40 describe the 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 I41, I42, and I43 are specified to allow the IRQ line detection circuitry to function correctly. It has no direct relation with the total system interrupt latency that the MPC801 supports.

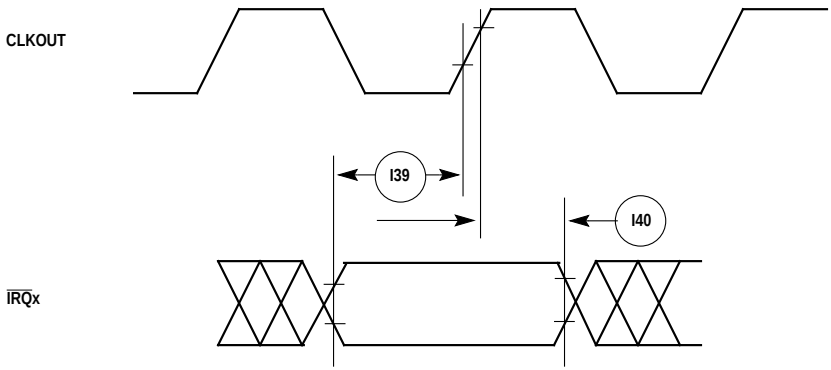


Figure 20-20. Interrupt Detection Timing Diagram for External Level-Sensitive Lines

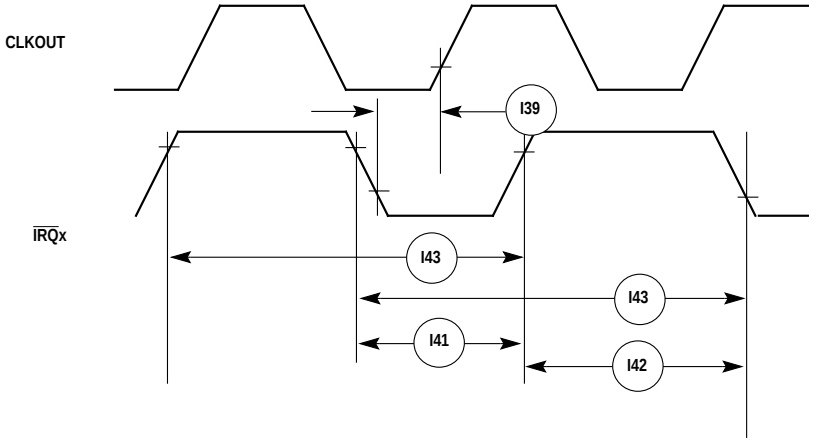


Figure 20-21. Interrupt Detection Timing Diagram for External Edge-Sensitive Lines

Table 20-3. Debug Port Timing

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 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

20

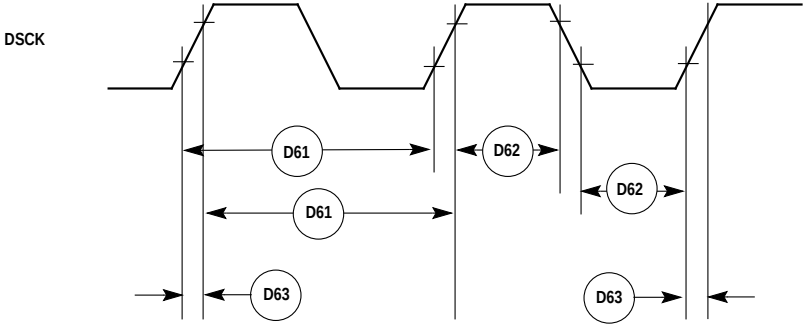


Figure 20-22. Debug Port Clock Input Timing Diagram

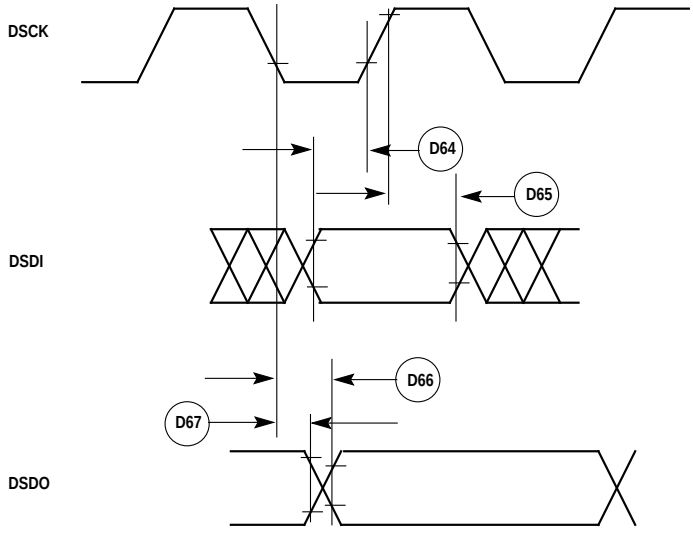


Figure 20-23. Debug Port Timing Diagram

Table 20-4. Reset Timing

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
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							
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

Table 20-4. Reset Timing (Continued)

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
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	8*TCC	320	—	200	—	ns

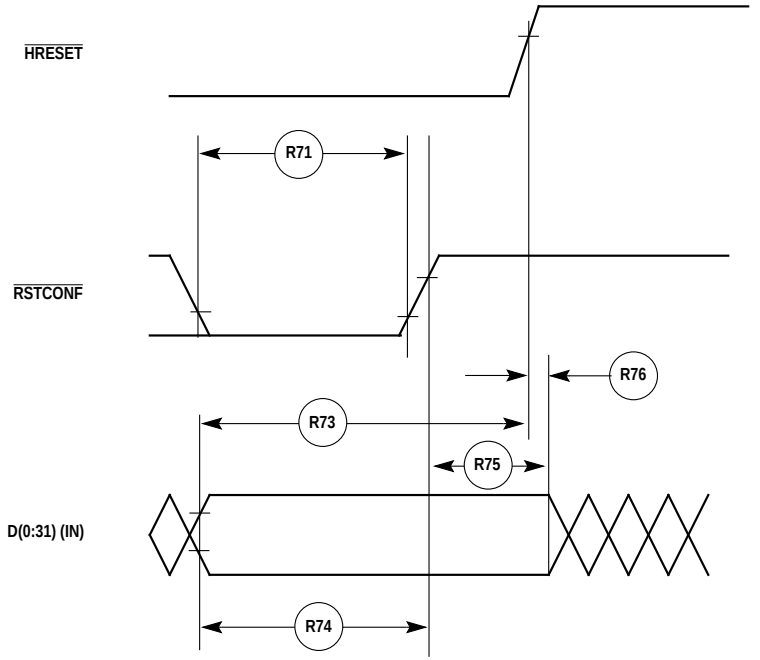


Figure 20-24. Reset Timing Diagram (Configuration From Data Bus)

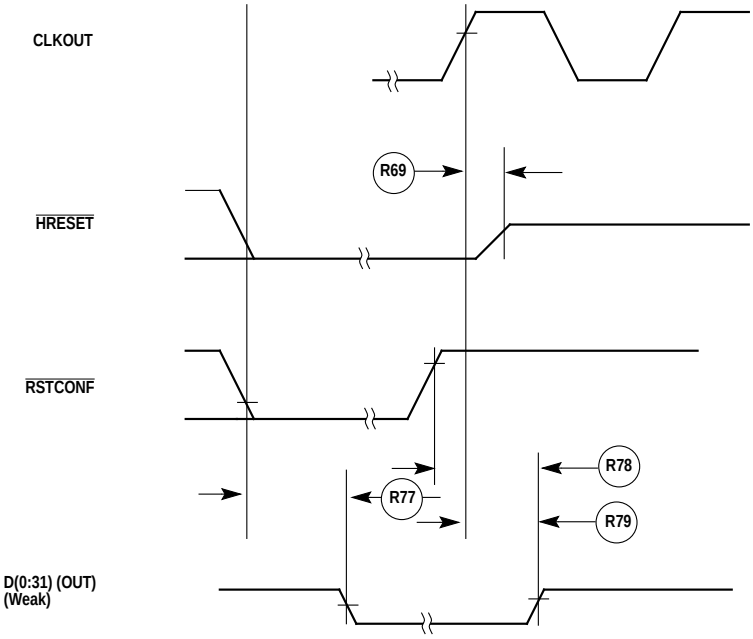


Figure 20-25. Reset Timing Diagram–MPC801 Data Bus Weak Drive During Configuration

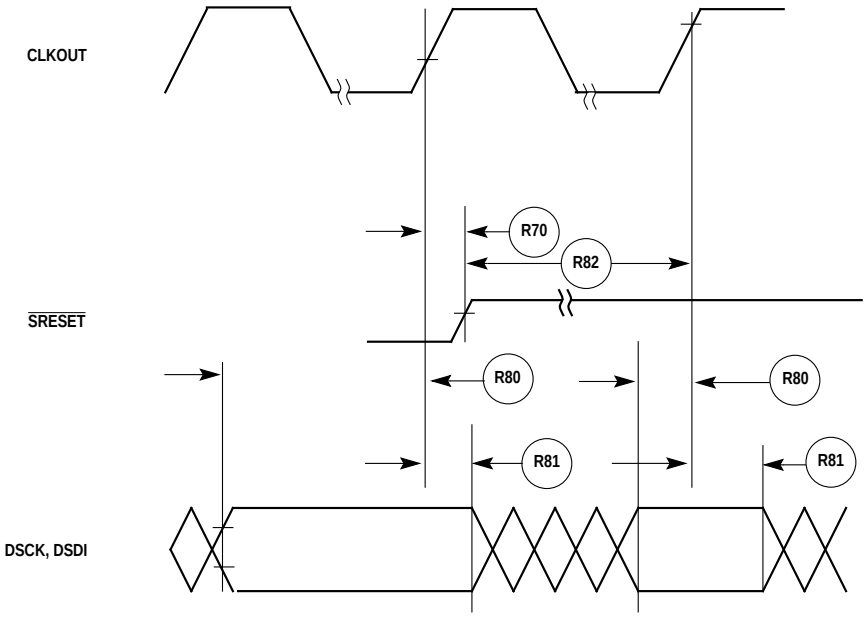


Figure 20-26. Reset Timing Diagram–Debug Port Configuration

20.6 IEEE 1149.1 ELECTRICAL SPECIFICATIONS

Table 20-5. JTAG Timing

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 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		—	27	—	27	ns
J88	TCK Low To TDO Data Invalid		0	—	0	—	ns
J89	TCK Low To TDO High Impedance		—	20	—	20	ns
J90	$\overline{\text{TRST}}$ Assert Time		100	—	100	—	ns
J91	$\overline{\text{TRST}}$ 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

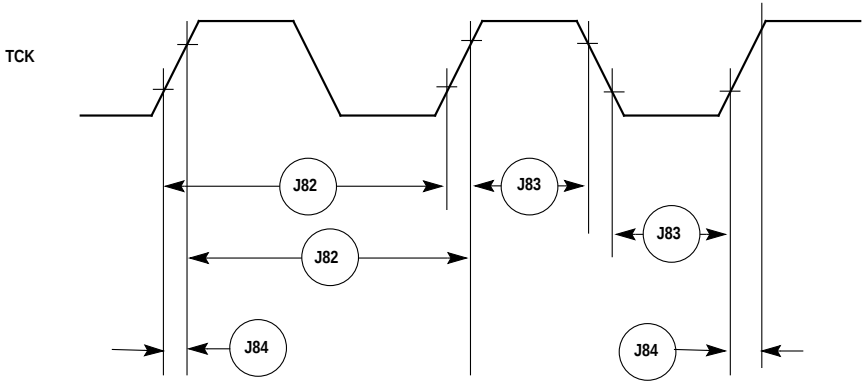


Figure 20-27. JTAG Test Clock Input Timing Diagram

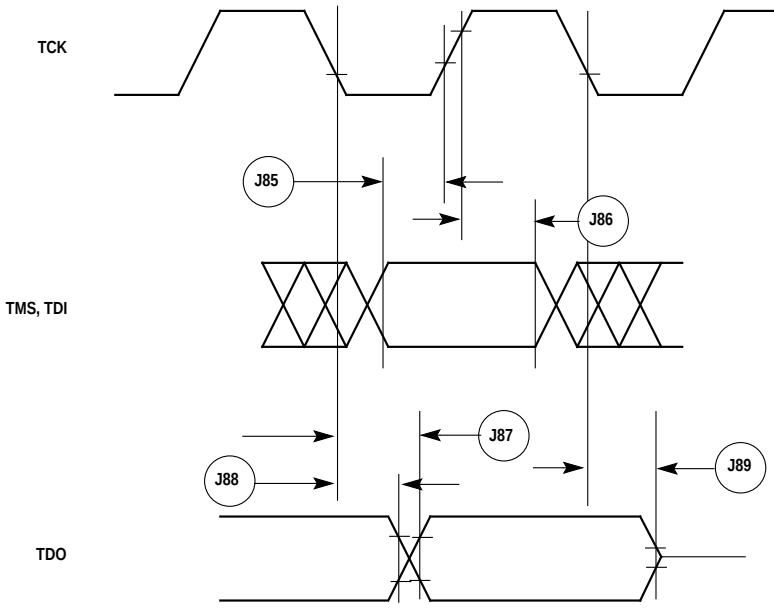


Figure 20-28. JTAG–Test Access Port Timing Diagram

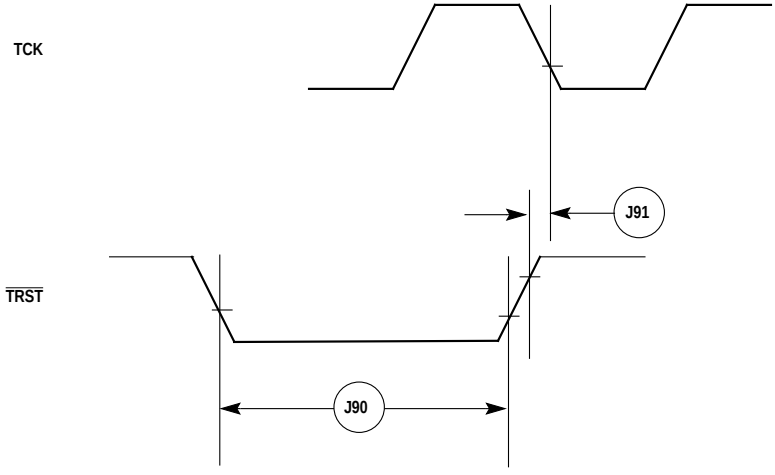


Figure 20-29. JTAG–$\overline{\text{TRST}}$ Timing Diagram

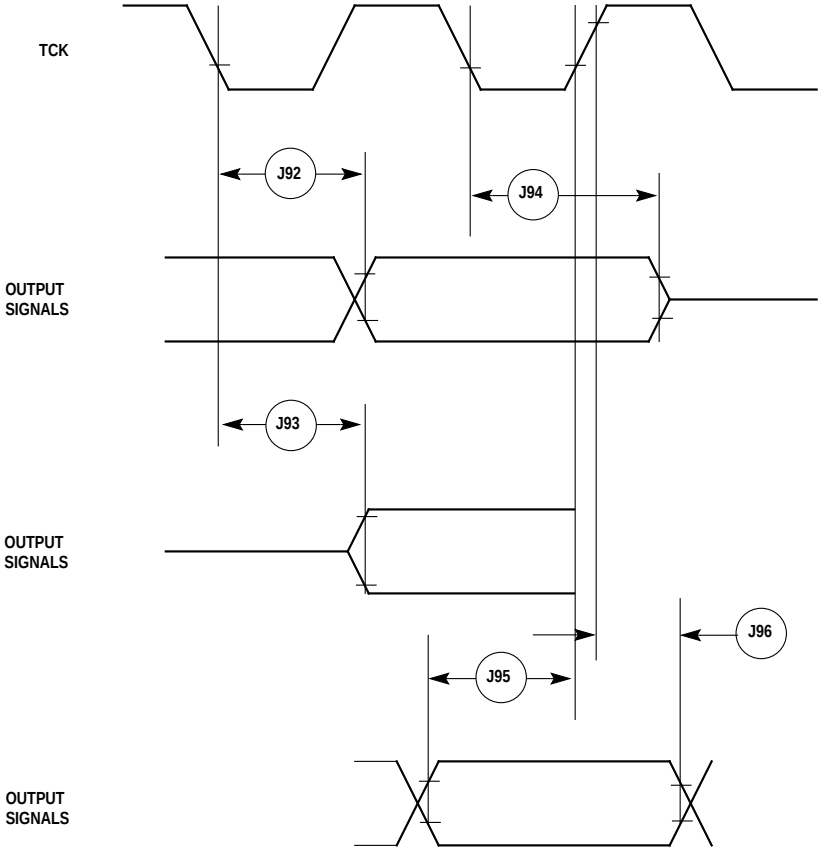


Figure 20-30. Boundary Scan (JTAG) Timing Diagram

SECTION 21
COMMUNICATION ELECTRICAL CHARACTERISTICS

21.1 PIO AC ELECTRICAL SPECIFICATIONS

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
1	Data-In Setup Time To Clock Low		20	—	15	—	ns
2	Data-In Hold Time From Clock Low		10	—	7.5	—	ns
3	Clock High To Data-Out Valid (CPU Writes Data, Control, Or Direction)		—	25	—	25	ns

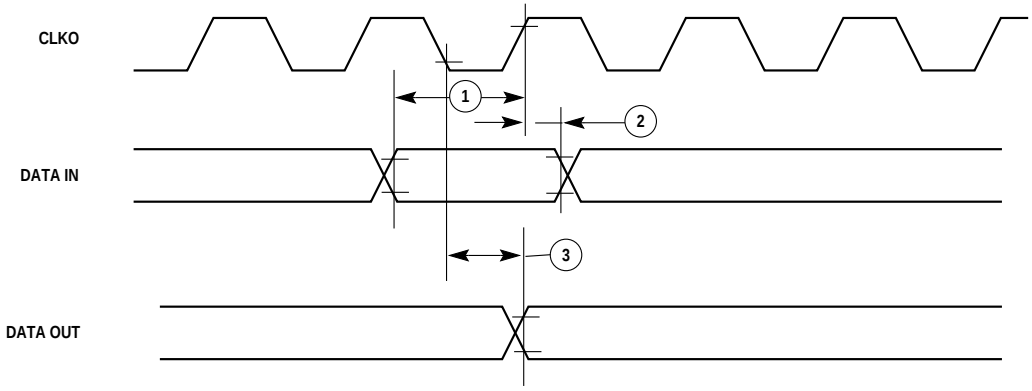


Figure 21-1. Parallel I/O Data-In/Data-Out Timing Diagram

21.2 UART BRG CLOCK AC ELECTRICAL SPECIFICATIONS

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
50[1]	BRG[2] Rise and Fall Time		—	10	—	10	ns
51[1]	BRG[2] Duty Cycle		40	60	40	60	%
52[1]	BRG[2] Cycle		40	—	40	—	ns

NOTES:

1. These specifications are valid when dividing factor≠1 and has a 50% duty cycle.
2. Baud rate generator from UART is allowable when there is dedicated usage of the UGPIO signal.

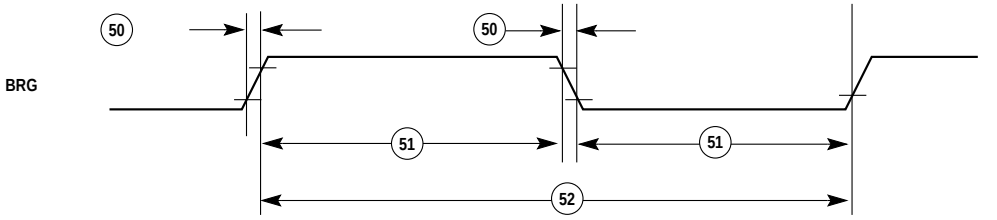


Figure 21-2. Baud Rate Generator from UART–Timing

21.3 UART—EXTERNAL CLOCK AC ELECTRICAL SPECIFICATIONS

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
100[1]	CLK Width High [2]		CLKO	—	CLKO	—	ns
101	CLK Width Low [2]		CLKO+5	—	CLKO+5	—	ns
102	CLK Rise/Fall Time [2]		—	15	—	15	ns
103	UTXD Active Delay (From CLK Falling Edge) [2]		0	50	0	50	ns
104	$\overline{\text{URTS}}$ Active/Inactive Delay (From CLK Rising Edge) [2, 4]		0	50	0	50	ns
105	$\overline{\text{UCTS}}$ Setup Time To CLK Rising Edge [2]		5	—	5	—	ns
106	URXD Setup Time To CLK Rising [2, 3]		5	—	5	—	ns
107	URXD Hold Time From CLK Rising Edge [2, 3]		5	—	5	—	ns

NOTES:

1. The ratio SyncCLK/CLK must be greater or equal to 2.25/1.

2. EXTCLK is available when UGPIO is assigned this functionality.

3. Receive specification is referred to synchronous mode X=1.

4. Specification refers to $\overline{\text{URTS}}$ activation in serial mode.

21.4 UART—INTERNAL CLOCK AC ELECTRICAL SPECIFICATIONS

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 MHZ		UNIT
			MIN	MAX	MIN	MAX	
100 [1]	CLK Frequency		0	8.3	0	13	MHz
102	CLK Rise/Fall Time		—	—	—	—	ns
103	UTXD Active Delay (From CLK Falling Edge) [2]		0	30	0	30	ns
104	$\overline{\text{URTS}}$ Active/Inactive Delay (From CLK Rising Edge) [2, 4]		0	30	0	30	ns
105	$\overline{\text{UCTS}}$ Setup Time To CLK Rising Edge [2]		40	—	40	—	ns
106	URXD Setup Time To CLK Rising Edge [2, 3]		40	—	40	—	ns
107	URXD Hold Time Ffrom CLK Rising Edge [2, 3]		0	—	0	—	ns

NOTES:

1. The ratio SyncCLK/CLK must be greater than or equal to 3/1.

2. The CLK is visible when UGPIO is configured accordingly.

3. Receive specification is referred to synchronous mode X=1.

4. Specification refers to $\overline{\text{URTS}}$ activation in serial mode.

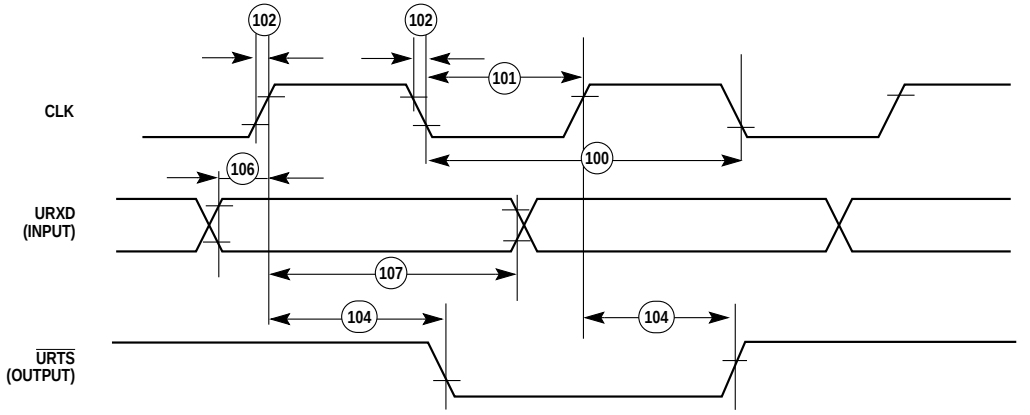


Figure 21-3. UART Receive

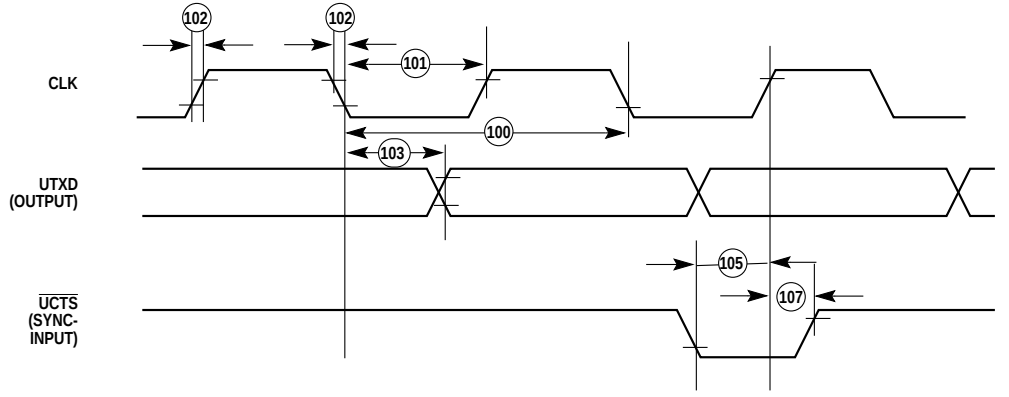


Figure 21-4. UART Transmit

21.5 SPI MASTER AC ELECTRICAL SPECIFICATIONS

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 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 TimeOutput		—	15	—	15	ns

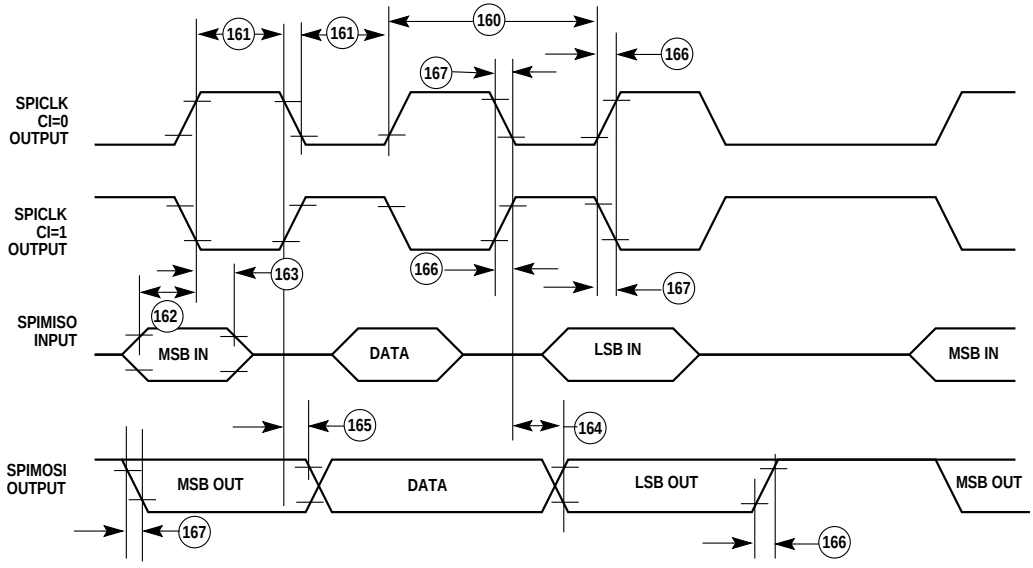


Figure 21-5. SPI Master (CP=0)

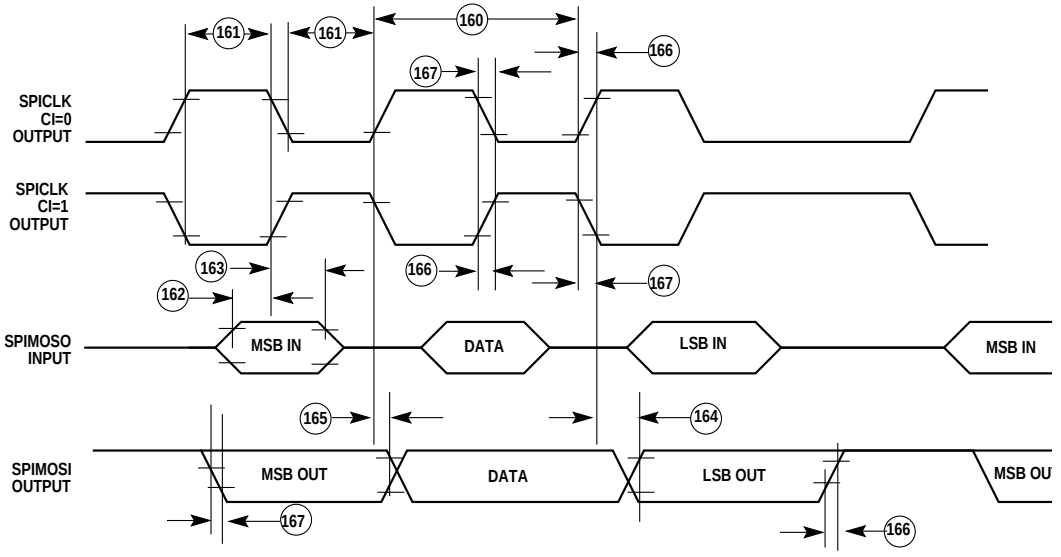


Figure 21-6. SPI Master (CP=1)

21.6 SPI SLAVE AC ELECTRICAL SPECIFICATIONS

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 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
178	Slave SPI MISO Disable Time		—	50	—	50	ns
179	Slave Data Valid (After SPICLK edge)		—	50	—	50	ns
180	Slave Data Hold Time (Outputs)		0	—	0	—	ns
181	Rise Time (Input)		—	15	—	15	ns
182	Fall Time (Input)		—	15	—	15	ns

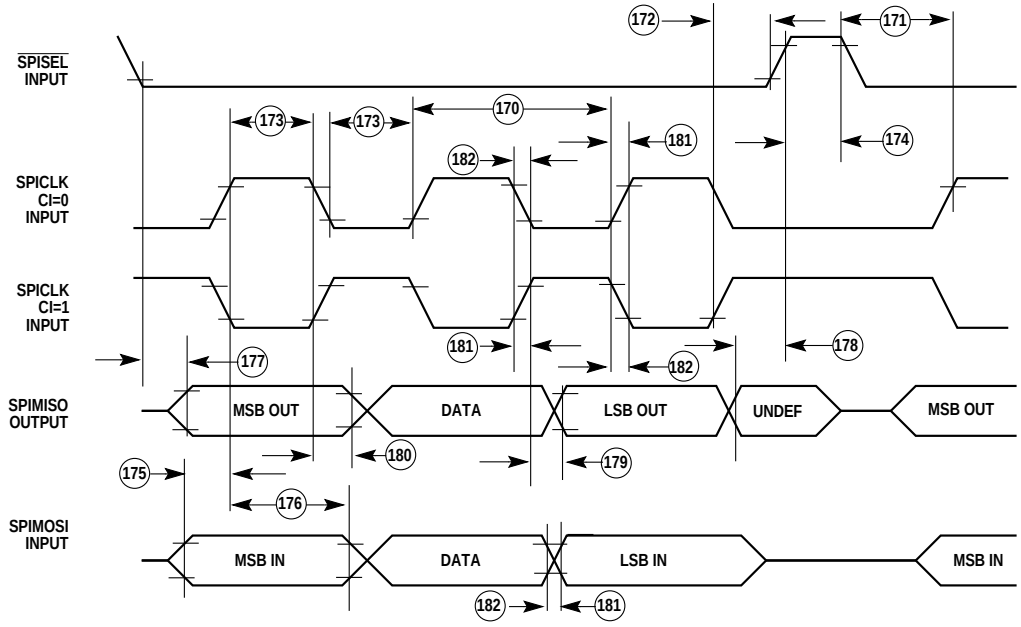


Figure 21-7. SPI Slave (CP=0)

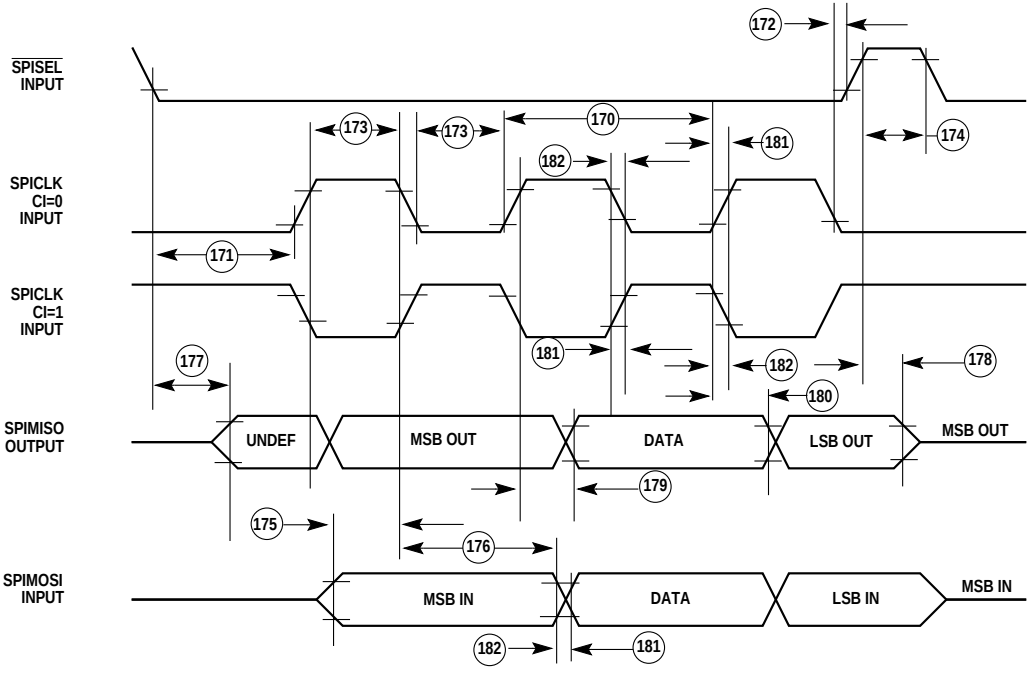


Figure 21-8. SPI Slave (CP=1)

21.7 I²C AC ELECTRICAL SPECIFICATIONS–SCL < 100 KHZ

NUM	CHARACTERISTIC	EXPRESSION	25 MHZ		40 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 than or equal to 4/1.

21.8 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)	—	sec
203	Low Period of SCL		1/(2.2 * fSCL)	—	sec
204	High Period of SCL		1/(2.2 * fSCL)	—	sec
205	Start Condition Setup Time		1/(2.2 * fSCL)	—	sec
206	Start Condition Hold Time		1/(2.2 * fSCL)	—	sec
207	Data Hold Time		0	—	sec
208	Data Setup Time		1/(40 * fSCL)	—	sec
209	SDL/SCL Rise Time		—	1/(10 * fSCL)	sec
210	SDL/SCL Fall Time		—	1/(33 * fSCL)	sec
211	Stop Condition Setup Time		1/(2.2 * fSCL)	—	sec

NOTE: SCL frequency is given by SCL = BRGCLK_frequency/((BRG register + 3) * pre_scaler * 2).
The ratio SyncClk/(BRGCLK/pre_scaler) must be greater than or equal to 4/1.

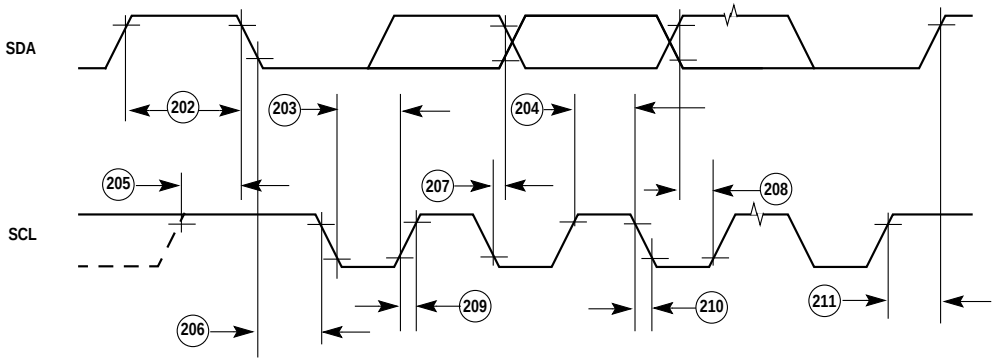


Figure 21-9. I²C Bus Timing

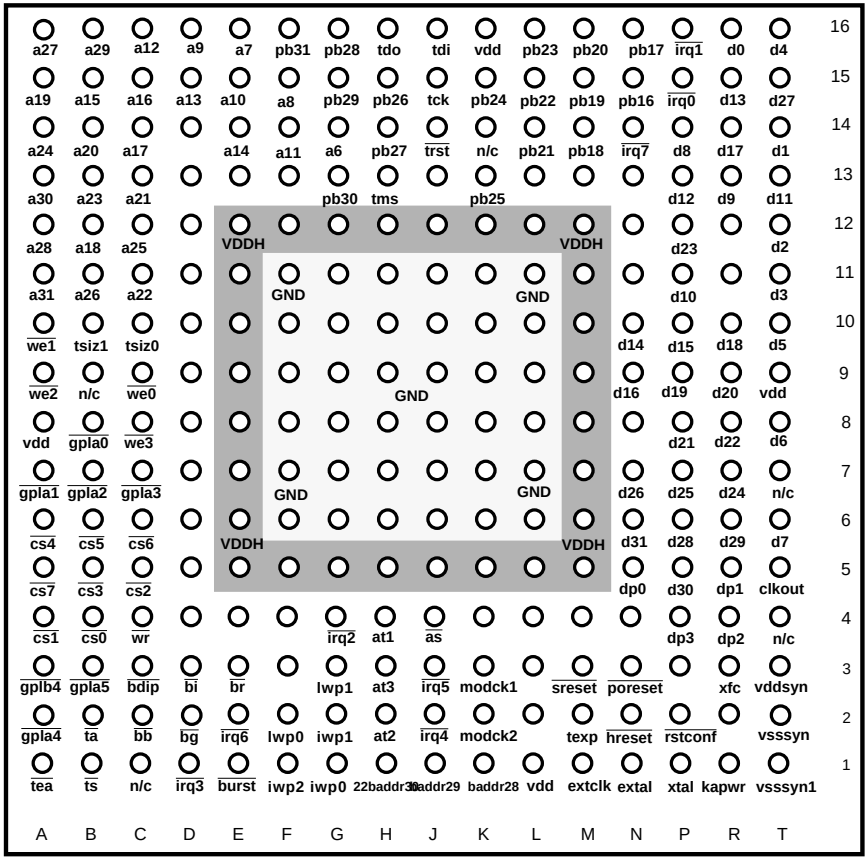
SECTION 22
MECHANICAL SPECIFICATIONS

22.1 ORDERING INFORMATION

To order this part, call your local sales office and provide them with the following information.

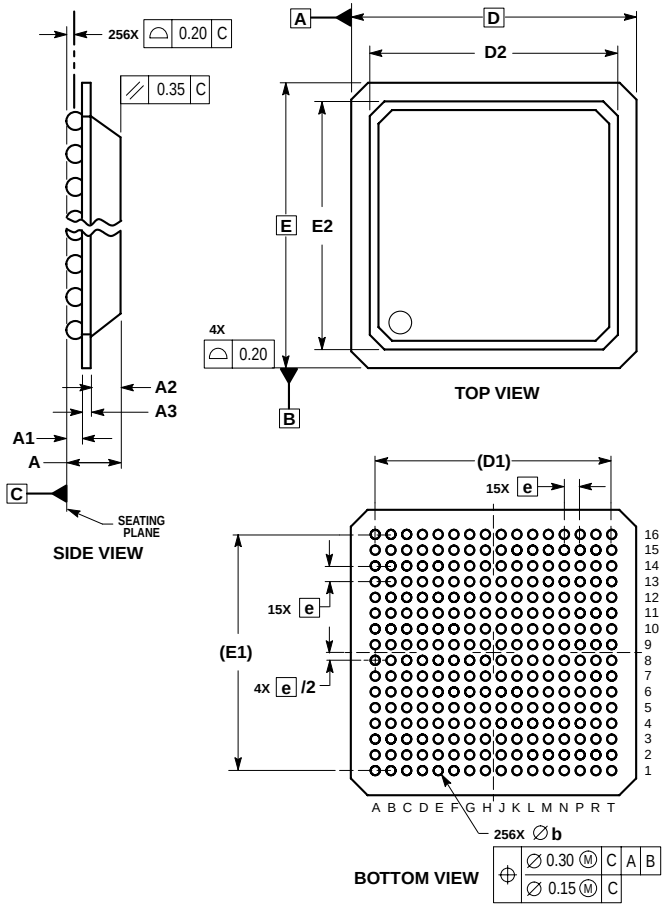
PACKAGE TYPE	FREQUENCY (MHZ)	TEMPERATURE	ORDER NUMBER
PBGA	25/40	90° C Junction	MPC801UM/AD

22.2 PIN ASSIGNMENTS—PBGA—TOP VIEW



22.3 PACKAGE DIMENSIONS

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*, which is available at your local Motorola sales office.



NOTES:
1. DIMENSIONING AND TOLERANCING PER ASME Y14.5M, 1994.
2. DIMENSIONS IN MILLIMETERS.
3. DIMENSION b IS MEASURED AT THE MAXIMUM SOLDER BALL DIAMETER, PARALLEL TO PRIMARY DATUM C.
4. PRIMARY DATUM C AND THE SEATING PLANE ARE DEFINED BY THE SPHERICAL CROWNS OF THE SOLDER BALLS.

	MILLIMETERS	
DIM	MIN	MAX
A	1.91	2.35
A1	0.50	0.70
A2	1.12	1.22
A3	0.29	0.43
b	0.60	0.90
D	23.00 BSC	
D1	19.05 REF	
D2	19.00	20.00
E	23.00 BSC	
E1	19.05 REF	
E2	19.00	20.00
e	1.27 BSC	

SECTION 23
TERMINOLOGY

This section provides the definitions to some of the key terms, concepts, and acronyms used in this document.

A

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)*.

ASID—Address space ID

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.

B

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).

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.

C

CAS—Column address strobe.

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.

D

datastream—A sequence of information to be processed by the core.

DEC—Decrementer.

E

EBI—External bus interface.

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).

F

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.

H

half-word—A half-word consists of 2 bytes or 16 bits.

I

I²C—Inter-integrated circuit.

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.

instruction issue—Driving valid instruction bits inside the core.

instruction stream—A sequence of operands to be executed by the core.

interrupt—The act of changing the machine state register and other parts of the machine state in response to an exception.

IrLAP—Infra-red link access protocol

K

K—Kilobyte.

Kb—Kilobit.

L

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.

LRU—Least recently used.

M

M—Megabyte.

MAC—Multiply and accumulate.

Mb—Megabit.

memory controller—A functional logic section of the MPC801. It's primary function is to provide the controls for the external bus memories and I/O devices.

MMU—Memory management unit.

N

NMI—Nonmaskable interrupt.

no operation (NOP)—An instruction whose sole function is to increment the Program Counter, but which affects no changes to any registers or memory.

P

PCI—Peripheral component interconnect bus.

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.

PIT—Periodic interrupt timer.

R

RAS—Row address strobe.

RTC—Real-time clock.

RX—Receiver.

S

scan chain—The peripheral buffers of a device, linked in JTAG test mode, that are addressed in a shift register fashion.

SCC—Serial communication controller.

sequential instruction—Any instruction that is not a flow control instruction and not ISYNC.

SIU—System interface unit.

SMC—Serial management controller.

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.

SPI—Serial peripheral interface.

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).

SWT—Software watchdog timer.

T

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.

TB—Timebase.

TDM—Time-division multiplex.

TLB—Translation lookaside buffer.

transaction—A bus transaction consists of an address transfer (address phase) and data transfers (data phase).

TSA—Time-slot assigner.

TX—Transmitter.

U

UART—Universal asynchronous receiver/transmitter.

UPM—User-programmable machine.

V

VLSI—Very large-scale integration chip consolidation.

W

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
QUICK REFERENCE GUIDE TO MPC801 REGISTERS

This appendix contains address information about the core control registers, internally mapped registers, and UPM RAM which has an indirect address access.

A.1 CORE CONTROL REGISTERS

The core and system interface unit control registers are implemented within the MPC801 and can be accessed with 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 801 modules that are not dedicated to core operation. During $\overline{\text{HRESET}}$, 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 $\overline{\text{HRESET}}$, the $\overline{\text{RSTCONF}}$ 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 A-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

A

Table A-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 A-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

Table A-3. Added Special-Purpose Registers (Continued)

SPR			NAME	DESCRIPTION
DECIMAL	SPR 5:9	SPR 0:4		
789	11000	10101	MI_TWC (MI_L1DL2P)	Instr. MMU Tblewalk Cntl
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 RAM Read 0
818	11001	10010	MI_DBRAM1	Data MMU RAM 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 (MD_L1P)	MMU TableWalk Base
797	11000	11101	MD_TWC (MD_L1DL2P)	Data MMU TbleWalk Cntl
798	11000	11110	MD_RPN	Data MMU Real Pg Num
799	11000	11111	M_TW (M_SAVE)	MMU TableWalk Special
824	11001	11000	MD_DBCAM	Data MMU CAM Read
825	11001	11001	MD_DBRAM0	Data MMU RAM Read0
826	11001	11010	MD_DBRAM1	Data MMU RAM Read1

Table A-4. Other Control Registers

DESCRIPTION	NAME
Machine State Register	MSR
Condition Register	CR

A.2 INTERNALLY MAPPED REGISTERS

The MPC801 internal memory resources are mapped within a contiguous block of storage. The size of the internal space in the MPC801 is 16K. The location of this block within the global 4G real storage space can be mapped on a 64K resolution through an implementation specific special register, and the internal memory map register (IMMR). The following table defines the internal memory map of the MPC801.

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 register are set to $FF00 must be initialized, the SDMA configuration register (SDCR) at location $FF000030 can be written to. To access the dual port RAM, you should address locations $FF002000 through $FF004000.

Table A-5. MPC801 Internal Memory Map

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
GENERAL SYSTEM INTERFACE UNIT			
000	SIUMCR	SIU Module Configuration Register	32
004	SYPCR	System Protection Control Register	32
008	RES	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	RES	Reserved	—
UART1 CONTROLLER			
040	CNTREG1	Control Register	16
044	BAUDREG1	Baud Control Register	16
048	GLOBREG1	Global Register	16
04C	RXREG1	Receiver Register	16
050	TXREG1	Transmitter Register	16
054	RXREG1	Receiver Register (Special Read Mode)	16
058 to 05F	RES	Reserved	—
UART2 CONTROLLER			
060	CNTREG2	Control Register	16
064	BAUDREG2	Baud Control Register	16
068	GLOBREG2	Global Register	16
06C	RXREG2	Receiver Register	16
070	TXREG2	Transmitter Register	16
074	RXREG2	Receiver Register (Special Read Mode)	16
078 to 0FF	RES	Reserved	—

Table A-5. MPC801 Internal Memory Map (Continued)

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
MEMORY CONTROLLER			
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
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	RES	Reserved	—
164	MAR	Memory Address Register	32
168	MCR	Memory Command Register	32
16C to 16F	RES	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

Table A-5. MPC801 Internal Memory Map (Continued)

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
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 24F	RES	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	RES	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	RES	Reserved	—
CLOCKS AND RESET			
280	SCCR	System Clock Control Register	32
284	PLPRCR	PLL, Low-Power and Reset Control Register	32
288	RSR	Reset Status Register	32
28C to 2FF	RES	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	RES	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

Table A-5. MPC801 Internal Memory Map (Continued)

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
330 to 33F	RES	Reserved	—
340	PISCRK	Periodic Interrupt Status and Control Register Key	32
344	PITCK	Periodic Interrupt Count Register Key	32
348 to 37F	RES	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	RES	Reserved	—
I^2C CONTROLLER			
180	I2CER	I^2C Event Register	8
184	I2CMR	I^2C Mask Register	8
188	I2CRD	I^2C Receive Data Register	16
18C	I2CTD	I^2C Register	16
190	I2MOD	I^2C Mode Register	8
194	I2ADD	I^2C Address Register	8
198	I2BRG	I^2C BRG Register	8
19C	I2COM	I^2C Command Register	8
1A0 to 1AF	RES	Reserved	—
SERIAL CONTROLLER			
1B0	SECCOM	Serial Controller Command Register	8
1B4 to 1BF	RES	Reserved	—
SERIAL PERIPHERAL INTERFACE			
1C0	SPIER	SPI Event Register	8
1C4	SPIMR	SPI Mask Register	16
1C8	SPIRD	SPI Receive Data Register	32
1CC	SPITD	SPI Transmit Data Register	32
1D0	SPMODE	SPI Mode Register	16
1D5	SPCOM	SPI Command Register	8
1D8 to 1DF	RES	Reserved	—

Table A-5. MPC801 Internal Memory Map (Continued)

INTERNAL ADDRESS	MNEMONIC	NAME	SIZE
PORT B			
1E0	PBODR	Port B Open-Drain Register	16
1E4	PBDAT	Port B Data Register	16
1E8	PBDIR	Port B Data Direction Register	16
1EC	PBPAR	Port B Pin Assignment Register	16
AF0 to 1FF	RES	Reserved	—

A

APPENDIX B
APPLICATIONS

This section describes how to move applications from the MC68360 QUICC environment to the MPC801 PowerQUICC environment. It is assumed that you are familiar with the differences between the 68K-type bus used by the QUICC and the PowerPC™ architecture used by the MPC801. This section is intended to address transferring software from one application to another.

B.1 MPC801 BASIC INITIALIZATION

The values given below are valid for the Motorola MPC801 application development system, but their validity cannot be guaranteed for all other designs. When the MPC801 comes out of power-on or hard reset, the following sequence occurs:

1. The hardware configuration is sampled on the two last rising edges of CLKOUT before $\overline{\text{HRESET}}$ is released.
2. The sampled value can either be from the data bus or an internal default configuration. If, at sampling time, the $\overline{\text{RSTCONF}}$ is asserted, then the configuration is sampled from the data bus. Otherwise, it is sampled from the internal default.
3. The word sampled on the bus informs the MPC801 of the arbitration type that is allowed, what the initial interrupt prefix will be, whether or not the memory controller should be disabled at startup (the general-purpose chip-select machine is disabled, thus disabling $\overline{\text{CS0}}$), what the boot port size should be, and what the base address of the internal memory space is at startup.

Once the reset sequence is completed, the MPC801 should be initialized according to the following steps.

NOTE

The values written to registers in this example apply to a particular application and should not be used without first verifying that they are appropriate for the requirements of the system being initialized.

1. Initialize the core registers.

   MSR and SRR1—0x00001002.

   Enables external interrupts, machine check interrupts and recoverable interrupts. This value does not allow traces or floating-point operation, but enables supervisor mode. The byte ordering is determined in the LE bit of the MSR.

DER—0xffe7400f

Sets the decrementer register to $4x10^8$ iterations. The current values of DER is only suitable if your program is running under the MPCbug and you are not willing to claim any exceptions or handle any exceptions that occur while the program is running. This is due to the fact that all exceptions trap into the MPCbug. In a normal situation, DER must be initialized to zero so that all exceptions go to your program instead of entering debug mode. With this value in the DER, a system hang occurs if there is no connection at the debug port.

2. Initialize the following system interface unit registers.

   IMMR—Set to valid system address

   Sets the internal memory map base address, thus allowing the software to calculate the location of on-chip resources. Notice that setting the IMMR has the same effect as the value sampled in the ISB field on the data bus at system start-up. This value changes based on the base address used in a particular system design and accesses to memory-mapped registers should fit within a particular board's memory map.

   SIUMCR—OR the existing value with 0x00032640

   This three-states SPKROUT, enables the $\overline{\text{GPL}}$ pin functionality, enables $\overline{\text{IRQ6}}$, prevents the debugger from accessing address lines 8:15, and enables parity.

   SYPCR—0xffffff88

   Enables the bus monitor, causes SWT to stop when freeze is asserted, and clears the SWT timer enable bit. The SWT is set to the maximum period which is 64K clocks.

   TBSCR—0x00c2

   Freezes the decrementer and timebase counter when freeze is asserted and also allows the timebase to generate an interrupt when the REFA (REFB) bit is asserted.

   PISCR—0x0082

   Clears the periodic interrupt status and disables PIT interrupts when freeze is asserted.

B.1.1 Programming the UPM

The following tables are examples of how to program the UPM for a given type of DRAM. All example values assume 70ns of DRAM. Each table is a specific type of access and the starting address of each RAM entry is where the UPM goes for a given type of access. The current UPM values are only optimized for 70ns of DRAM when the MPC801 is running at a 50MHz clock. Running at a lower frequency degrades the memory access and causes the system performance to scale incorrectly when it uses these exact values.

Table B-1. Read Single Beat–RSSA

RAM ADDRESS	UPM WORD
0x00	0x0fffec24
0x01	0x0fffec04
0x02	0x0cffec04
0x03	0x00ffec04
0x04	0x00ffec00
0x05	0x37ffec47

Table B-2. Write Single Beat–WSSA

RAM ADDRESS	UPM WORD
0x18	0x0fafcc24
0x19	0x0fafcc04
0x1A	0x08afcc04
0x1B	0x00afcc00
0x1C	0x37ffcc47

Table B-3. Read Burst Cycle–RBSA

RAM ADDRESS	UPM WORD
0x08	0x0fffcc24
0x09	0x0fffcc04
0x0A	0x08ffcc04
0x0B	0x00ffcc04
0x0C	0x00ffcc08
0x0D	0x0cffcc44
0x0E	0x00ffec0c
0x0F	0x03ffec00
0x10	0x00ffec44
0x11	0x00ffcc08
0x12	0x0cffcc44
0x13	0x00ffec04
0x14	0x00ffec00
0x15	0x3fffec47

Table B-4. Write Burst Cycle–WBSA

RAM ADDRESS	UPM WORD
0x20	0x0fafcc24
0x21	0x0fafcc04
0x22	0x08afcc00
0x23	0x07afcc4c
0x24	0x08afcc00
0x25	0x07afcc4c
0x26	0x08afcc00
0x27	0x07afcc4c
0x28	0x08afcc00
0x29	0x37afcc47

Table B-5. Refresh Cycle–RBSA

RAM ADDRESS	UPM WORD
0x30	0xe0ffcc84
0x31	0x00ffcc04
0x32	0x00ffcc04
0x33	0x0fffcc04
0x34	0x7fffcc04
0x35	0xffffcc86
0x36	0xffffcc05

Table B-6. Exception Cycle–ECSA

RAM ADDRESS	UPM WORD
0x3C	0x33ffcc07

3. Initialize CS and the memory controllers.

   OR0/OR1/OR2—OR0=0xffe00954

   OR1 = 0xffff8110—Compares all bits in BR1 to the access address, uses normal $\overline{CS}$ behavior, defines address pins in the BR, and allows page mode and 10 clock wait state with normal timing.

   OR2 = 0xffc00800—Compares 10 most-significant bits in BR2 to the access address, uses normal $\overline{CS}$ behavior that is asserted at the same time as $\overline{AS}$, has zero wait states, and allows page mode.

   BR0/BR1/BR2—BR0=0xffc00800

   This section of memory is left invalid.

   BR1 = 0x21000001—This section of memory starts at address 0x210000, and GPCSM is valid;

   BR2 = 0x00000081—This section of memory starts at 0x000000, and UPM1 is valid.

   MPTPR 0x0800

   Scales the system frequency by eight before the periodic (refresh) timer is calculated.

   MAMR—0x80a21114

   Enables a refresh cycle every 40µs, has 10 column address lines, programs a single cycle precharge, and allows a single beat per word for all burst accesses.

4. Initialize the memory management unit (MMU).

   For a complete description of the initialization of the memory management unit, refer to the memory management unit and cache application note.

5. Initialize the cache.

   For a complete description of the procedure for initializing the cache, refer to the memory management unit and cache application note.

B.1.2 MPC801 MMU/Cache Example

B.1.2.1 BASIC MMU AND CACHE CONCEPT. This concept is a step-by-step example of how to initialize the memory management unit (MMU) and cache. It provides configuration and background information on the memory management unit and cache from a general conceptual viewpoint.

B.1.2.2 GENERAL CONCEPT. A basic review of cache and MMU concepts is necessary for the best productivity. The MPC801 incorporates on-chip cache memory that consists of two fully independent caches for improving overall performance. One cache is implemented as a core instruction cache and is used to store instruction prefetch accesses from main (system) memory. The second cache is implemented as a core data cache and stores data prefetched from the data cache. The data cache can also be used as a temporary location to store data, thus saving time by updating only the cache and not memory (writethrough mode).

Performance throughput is improved when instruction or data words required by a program are available in the on-chip cache and the time required to access them from external memory is eliminated. In the MPC801, having on-chip multiple bus masters result in reduced external bus activity by the core, which in turn increases overall performance by increasing the availability of the bus for use by other bus masters (SDMA) without degrading the performance of the core. Notice that throughout the rest of this section that the term problem mode refers to what is known as user mode in 68K terms.

B.1.2.2.1 Instruction Cache. An instruction cache is a unidirectional cache. At the beginning of a cycle the instruction is read by the core from main memory and then stored in cache for the next access by the core. However, the core can never directly alter the content of an instruction cache. Since an instruction cache can only be accessed by the core, you must assure that other bus masters never alter the contents of memory that is defined as cacheable. This can be done by using the IMMU. During normal operation, the following sequence of operations must be performed before enabling the instruction cache.

1. The cache must be unlocked in all locations by writing 101 (bin) to the CMD field of the IC_CST register.
2. All cache locations must be invalidated by writing 101 (bin) to the CMD field of the IC_CST register. This reset sequence guarantees that the instruction cache has no garbage data marked as valid before it is turned on.
3. Define cacheable and noncacheable regions of main memory by appropriately initializing the memory management unit before enabling the instruction cache.

Only after performing this sequence should you turn the instruction cache on by writing 001 (bin) to the CMD field of the IC_CST register.

B.1.2.2.2 Data Cache. Data cache is a bidirectional cache. The core can read as well as write directly into the content area of the data cache. In addition to this, the MPC801 allows both writethrough and writeback operation. During normal operation, the following sequence of tasks must be performed before enabling the data cache.

1. The cache must be unlocked in all locations by writing 101 (bin) to the CMD field of the DC_CST register.
2. All cache locations must be invalidated by writing 101 (bin) to the CMD field of the DC_CST register. This reset sequence guarantees that the data cache has no valid garbage data when it is turned on.
3. Define cacheable and noncacheable regions and select write-back or write-through mode for the cacheable region of main memory by initializing the memory management unit. This must be performed before enabling the data cache.

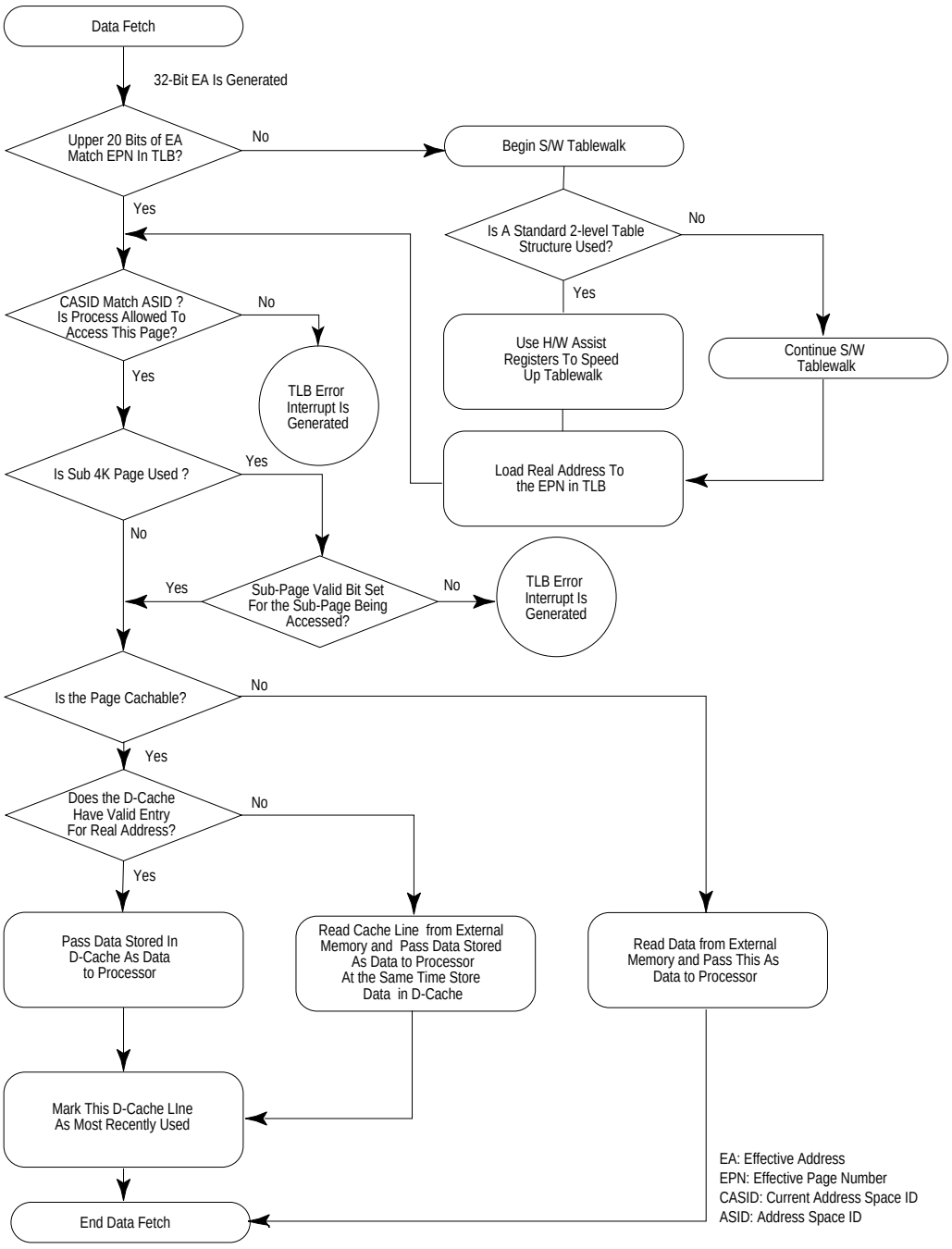
After following this sequence, the instruction cache can be turned on by writing 001 (bin) to the CMD field of the DC_CST register. Notice that since the data cache can only be accessed by the core, no other bus master is allowed to alter the contents of the cache. Cacheable memory locations should not be expected to contain correct data at a particular time in writeback mode because the bus arbitration is transparent to the programmer. In this situation, the SDMA of the MPC801 would be able to execute a read from a memory location that may not yet be updated with new data from the cache.

In another example, if the SDMA writes to a cacheable location, the data in main memory is now more updated than the valid entry available in the data cache. The next core access from this location results in the core obtaining old data from the cache and not the updated data from the main memory. This occurs because the cache has no knowledge that there is newer data in main memory. Cacheable, noncachable, writethrough, and copyback regions can be set up in memory by setting up the DMMU.

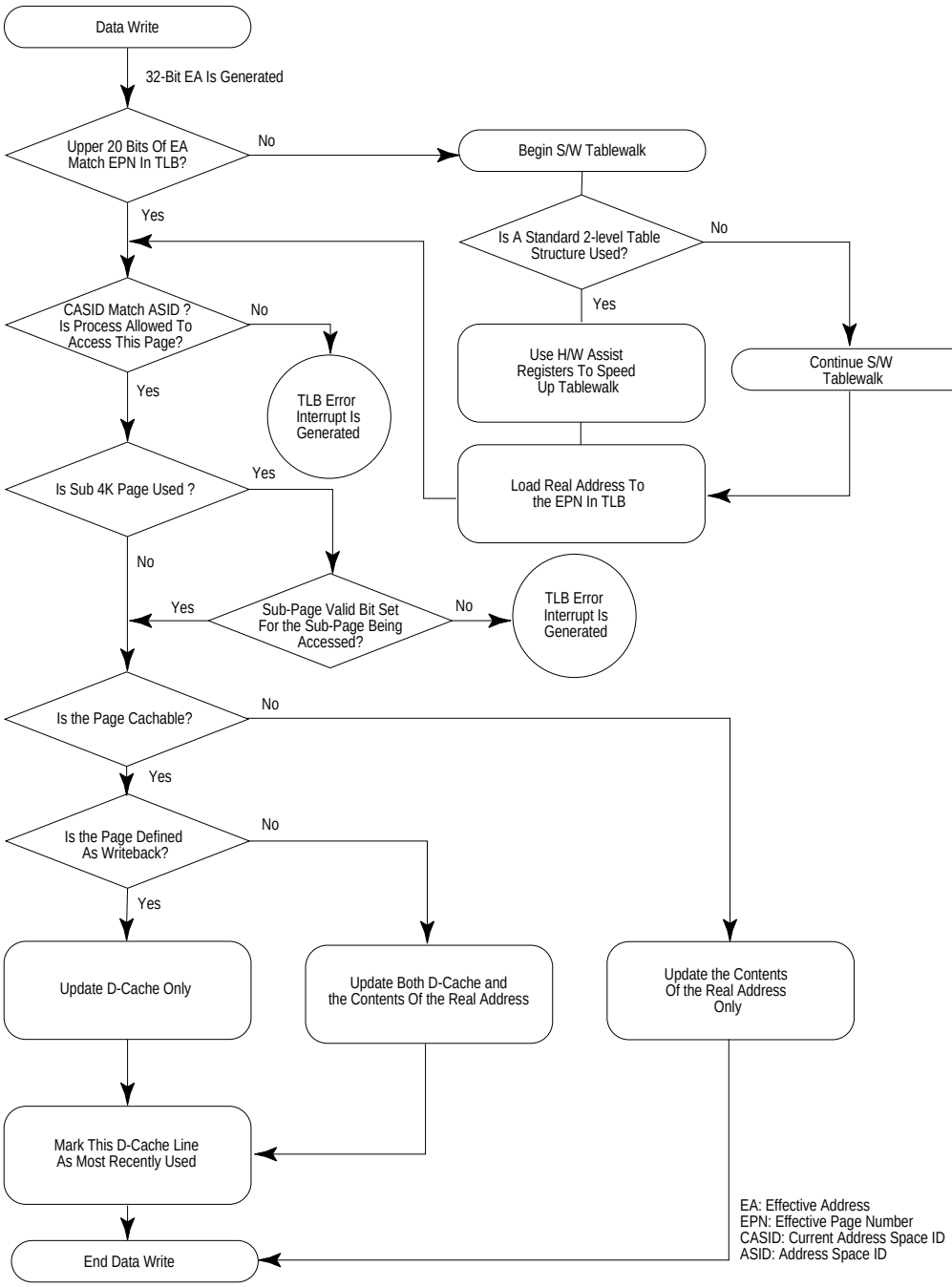
B.1.3 Memory Management Unit

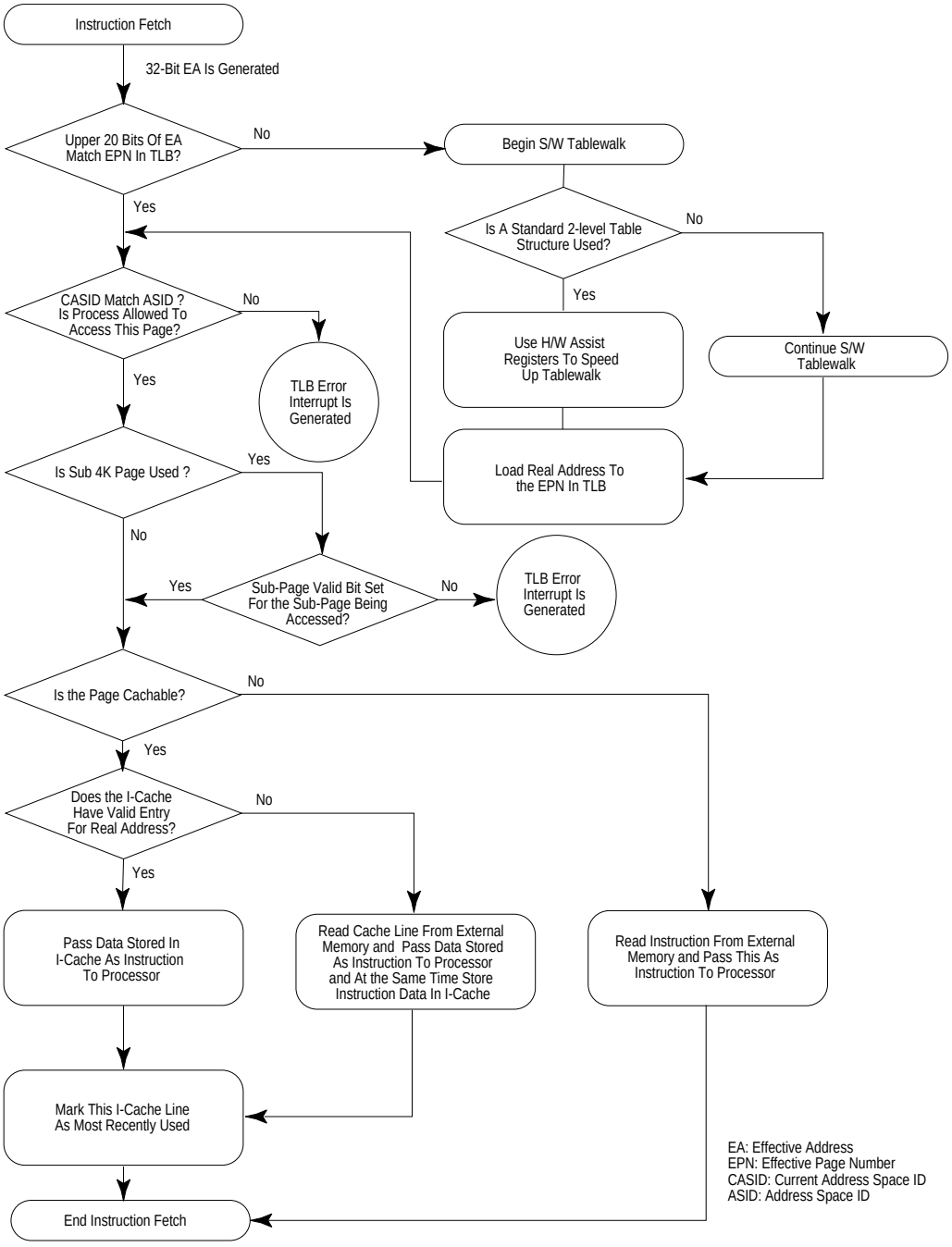
This section describes the basic concepts and background information concerning the MPC801 memory management unit. The MPC801 supports a demand paged virtual memory environment. The term *demand* refers to the fact that software programs request memory through effective addresses and the term *paged* refers to the fact that memory is divided into blocks of the same size page frame. Each page frame is then divided into a known page size—8M, 512K, 16K, 4K, or 1K. The operating system assigns pages to page frames as required.

The principal memory management unit function is to translate an effective address to a real address using translation tables stored in memory. When the memory management unit receives an effective address from the load unit, it searches in each entry of its TLB for the current page to get the corresponding real address. If the translation is in the TLB, the memory management unit provides the real address to the cache controller, which determines if the instruction or data being accessed is cached. When the translation is not in the TLB, the memory management unit searches the translation tables in the memory for the translation information. This is called a tablewalk.


Data Fetch
32-Bit EA Is Generated
Upper 20 Bits of EA Match EPN In TLB?
No
Begin S/W Tablewalk
Yes
Is A Standard 2-level Table Structure Used?
No
Yes
CASID Match ASID ? Is Process Allowed To Access This Page?
No
Use H/W Assist Registers To Speed Up Tablewalk
Continue S/W Tablewalk
Yes
TLB Error Interrupt Is Generated
Is Sub 4K Page Used ?
Yes
Load Real Address To the EPN in TLB
No
Yes
Sub-Page Valid Bit Set For the Sub-Page Being Accessed?
No
TLB Error Interrupt Is Generated
Is the Page Cachable?
No
Yes
Does the D-Cache Have Valid Entry For Real Address?
No
Yes
Pass Data Stored In D-Cache As Data to Processor
Read Cache Line from External Memory and Pass Data Stored As Data to Processor At the Same Time Store Data in D-Cache
Read Data from External Memory and Pass This As Data to Processor
Mark This D-Cache LIne As Most Recently Used
End Data Fetch
EA: Effective Address
EPN: Effective Page Number
CASID: Current Address Space ID
ASID: Address Space ID

B


Data Write
32-Bit EA Is Generated
Upper 20 Bits Of EA Match EPN In TLB?
No
Begin S/W Tablewalk
Yes
Is A Standard 2-level Table Structure Used?
No
Yes
CASID Match ASID ? Is Process Allowed To Access This Page?
No
Use H/W Assist Registers To Speed Up Tablewalk
Continue S/W Tablewalk
Yes
TLB Error Interrupt Is Generated
Is Sub 4K Page Used ?
Yes
Load Real Address To the EPN In TLB
No
Yes
Sub-Page Valid Bit Set For the Sub-Page Being Accessed?
No
TLB Error Interrupt Is Generated
Is the Page Cachable?
No
Yes
Is the Page Defined As Writeback?
No
Yes
Update D-Cache Only
Update Both D-Cache and the Contents Of the Real Address
Update the Contents Of the Real Address Only
Mark This D-Cache Line As Most Recently Used
End Data Write
EA: Effective Address
EPN: Effective Page Number
CASID: Current Address Space ID
ASID: Address Space ID


Instruction Fetch
32-Bit EA Is Generated
Upper 20 Bits Of EA Match EPN In TLB?
No
Begin S/W Tablewalk
Yes
Is A Standard 2-level Table Structure Used?
No
Yes
CASID Match ASID ? Is Process Allowed To Access This Page?
No
Use H/W Assist Registers To Speed Up Tablewalk
Continue S/W Tablewalk
Yes
TLB Error Interrupt Is Generated
Is Sub 4K Page Used ?
Yes
Load Real Address To the EPN In TLB
No
Yes
Sub-Page Valid Bit Set For the Sub-Page Being Accessed?
No
TLB Error Interrupt Is Generated
Is the Page Cachable?
No
Yes
Does the I-Cache Have Valid Entry For Real Address?
No
Yes
Pass Data Stored In I-Cache As Instruction To Processor
Read Cache Line From External Memory and Pass Data Stored As Instruction To Processor and At the Same Time Store Instruction Data In I-Cache
Read Instruction From External Memory and Pass This As Instruction To Processor
Mark This I-Cache Line As Most Recently Used
EA: Effective Address
EPN: Effective Page Number
CASID: Current Address Space ID
ASID: Address Space ID
End Instruction Fetch

B

The MPC801 implements a software tablewalk with hardware assistance to perform the calculation of address required for each search. For more information on TLB operation, refer to **Section 11.2.1 Translation Lookaside Buffer Operation** and **Section 11.8.1 Reloading the TLB** for an example of software tablewalk code. The following three figures demonstrate the flow of instruction or data. The memory management unit obtains the real address and then gets the data or instruction through the cache or main memory.

B.1.3.1 MEMORY PROTECTION. The main reason to use the memory management unit in an embedded application is to provide memory protection. There are two levels of protection—level one and level two.

B.1.3.1.1 Level One Protection. Level one provides the following memory protection:

- Access Protection Group—The access protection register contains 16 fields.The contents of this field are used according to the group protection mode. In the PowerPC mode, each field holds the Kp and Ks bits of a corresponding segment register. To be consistent with the *PowerPC Family: The Programming Environment* manual, 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 for the page protection setting. No override, no access override, or free access override modes are supported.
- Guarded Protection—The guarded attribute pertains to out-of-order execution. When a page is designated as guarded, instructions and data cannot be accessed by an out-of-order execution. When a page is designated as unguarded, out-of-order fetches and accesses are allowed.
- Cache Mode—Writethrough or writeback cache mode is selected.

B.1.3.1.2 Level Two Protection. Level two provides the following memory protection:

- Privilege or Problem Access Protection—This allows you to program a page access as privilege only, problem only, or both.
- Read or Write Protection.
- Page Shared or Not Shared—If the page is marked as shared the address space ID must match the M_CASID entry.
- Cache Enable or Disable.

B.1.3.2 MMU EXAMPLE. You should make sure that accesses to memory are error-free before enabling the memory management unit, or cache. Bits 26 and 27 of the MSR enable/disable the memory management unit.

The following example shows how to set up the memory management unit registers and multiple table descriptors. In this example it is assumed that read/write memory is located at address $40000 for the memory translation tables or table descriptors. This example does not include address translation, so the effective address is identical to the real address. In this example, the memory management unit is used to implement various cache modes, privilege protection, and read/write protection.

Table B-7. Physical Memory Map Example

ADDRESS RANGE	ACCESSED DEVICE	PORT WIDTH
00000000 - 003FFFFF	FLASH PROM Bank 1	32
00400000 - 007FFFFF	FLASH PROM Bank 2	32
00800000 - 00BFFFFF	DRAM 4MByte (4 Meg X 32bit)	32
09000000 - 09003FFF	MPC Internal MAP	32
09100000 - 09100003	Board Control & Status Register (BCSR)	32
10000000 - 17FFFFFF	PCMCIA channel	16

NOTE: 1. Configured at hard reset via the hard reset configuration word to FF000000, changed during a special register, and written/read using **mtspr**/**mftspr** commands.

2. The BCSR is available throughout the minimal block size of the MPC801's CS region, which is 64K. (091000000–0910FFFF).

The following table shows the tasks that can be performed using the memory management unit.

Table B-8. MMU Register

EFFECTIVE ADDRESS= REAL ADDRESS	PAGE SIZE/ NO. OF PAGE	CACHE/ GUARD	CACHE MODE	ACCESS PRIVILEGE	READ/ WRITE	SHARED PAGE	NOTE
00000000 - 007FFFFF	8M/1	On/No	Writethrough	Privilege	Read-only	Shared	Used for monitor program and translation table
00800000 - 008FFFFF	512K/2	On/No	Copyback	Privilege	Read/Write	Shared	Used for stack and monitor scratch pad.
00900000 - 0094FFFF	512K/1	Off/Yes	—	Privilege	Read/Write	Shared	Used for data buffers
00950000 - 00BFFFFF	512K/5	On/No	Copyback	Privilege/ Problem	Read/Write	Shared	Used for problem program and data space
09000000 - 09003FFF	16K/1	Off/Yes	—	Privilege	Read/Write	Shared	Used to access Power QUICC internal RAM and registers
09100000 - 09103FFF	16K/1	Off/Yes	—	Privilege	Read/Write	Shared	Used for access to board configuration register.
10000000 - 17FFFFFF	8M/16	Off/Yes	—	Privilege	Read/Write	Shared	

NOTE: The above cache and protection mode is selected to show as many of the variations as possible. There is no reason why some pages are selected as copyback or some writethrough other than for showing how this selection can be made.

B.1.4 M_TWB MMU TABLEWALK BASE REGISTER

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	L1TB															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	L1TB				L1INDX										RESERVED	

L1TB—Tablewalk Level One Base Value

These bits determine the starting location of a level one translation table. For this example, these bits are set to $00400 so that the level one table begins at address $00400000 (point to top of second Flash location).

L1INDX—Level One Table Index

These bits returns either EPN[0:9] or EPN[2:11], depending on the setting of TWAM bits of the MD_CTR. Since these bits are read-only, a write is ignored if it is set to all zeros.

B.1.5 MI_CTR Instruction MMU Control Register

B

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	GPM	PPM	CIDEF	RES	RSV4I	RES	PPCS	RESERVED								
RESET	0	0	0	0	0	0	1	0								
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED			ITLB_INDX					RESERVED							

GPM—Group Protection Mode

This bit must be set at 0 to select PowerPC protection mode.

0 = PowerPC mode.
1 = Domain manager mode.

PPM—Page Protection Mode

This bit must be set at 0 for page resolution of protection.

0 = Page resolution of protection.
1 = 1Kresolution of protection for 4K pages.

CIDEF—Cache Inhibit Default

This bit must be set at zero to disable the cache when the memory management unit is disabled.

0 = Disable instruction cache when IMMU is disabled.
1 = Enable instruction cache when IMMU id disabled.

RSV4I—Reserve Four Instruction TLB Entries

0 = ITLB_INDX decremented modulo 32.
1 = ITLB_INDX decremented modulo 28.

PPCS—Privilege/Problem State Compare mode

This bit must be set at 1 to compare privileged accesses on tablewalk.

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

ITLB_INDX—Instruction TLB Index

This bit is a pointer to the instruction TLB entry that should be loaded. This bit is automatically decremented at every instruction TLB update. Set to zero since it is loaded at the time of a tablewalk.

B.1.6 Mx_AP Instruction/Data Access Protection Register

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	AP1		AP2		AP3		AP4		AP5		AP6		AP7		AP8	
RESET	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	AP9		AP10		AP11		AP12		AP13		AP14		AP15		AP16	
RESET	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

APx—Access Protection x

Domain Manager Mode:
00 = No access.
01 = Client (access permission defined by page protection bits).
10 = Reserved.
11 = Manager (free access).

PowerPC Mode:
00 = All access are treated as privileged.
01 = Access permission defined by page protection bits.
10 = Problem and privilege interpretation swapped.
10 = All accesses are treated as problem.

B.1.7 MD_CTR Data MMU Control Register

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	GPM	PPM	CIDEF	WTDEF	RSV4D	TWAM	PPCS	RESERVED								
RESET	0	0	0	0	0	1	1	0								
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED			DTLB_INDX					RESERVED							
RESET	0			0					0							

GPM—Group Protection Mode

This bit must be set at 0 to select PowerPC protection mode.

0 = PowerPC mode.
1 = Domain manager mode.

PPM—Page Protection Mode

This bit must be set at 0 for page resolution of protection.

0 = Page resolution of protection.
1 = 1K resolution of protection for 4K pages.

CIDEF—Cache Inhibit Default

This bit must be set at 0 to disable cache when memory management unit is disabled.

0 = Disable data cache when IMMU is disabled.
1 = Enable data cache when IMMU id disabled.

WTDEF—Writethrough default

This bit is set to zero and is “don’t care” since CIDEF is set to zero.

0 = Writethrough mode when cache is enabled by CIDEF.
1 = Copyback mode when cache is enabled by CIDEF.

RSV4I—Reserve 4 Instruction TLB Entries

0 = ITLB_INDX decremented modulo 32.
1 = ITLB_INDX decremented modulo 28.

PPCS—Privilege/Problem State Compare Mode

This bit must be set at 1 to compare access privilege up on tablewalk.

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

TWAM—Tablewalk Assist Mode

This bit is set at 1 to support 4K page hardware assist tablewalk

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

DTLB_INDX—Data TLB Index

This bit is a pointer to the data TLB entry that should be loaded and is automatically decremented at every instruction TLB update. It is set to zero since it will be loaded when a tablewalk occurs.

B.1.8 Level One Descriptor Format Register

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	L2BA															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	L2BA				RESERVED			APG				G	PS		WT	

L2BA—Level 2 Table Base Address

These bits set the base address for the second translation table. Bits 18–19 are only used when the TWAM bits of the MD_CTR register are set to 1. Normally, it should be set to zero.

APG—Access Protection Group

This field selects one of 16 available protection groups form the MI/MD_AP register.

G—Guarded Storage Attribute

0 = Unguarded storage.
1 = Guarded storage.

PS—Page Size Level One

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

WT—Writethrough Attribute For Entry

0 = Copyback region.
1 = Writethrough region.

V—Segment Valid

0 = Segment is invalid.
1 = Segment is valid.

The level one table starts from Address $400000 from the setting in the M_TWB register. Address $400000 contains the level one descriptor 0 (L1D0) and the setting for L1D0 is as follows:

L2BA = $00401 Point level two table to $00401000.
APG = $0 Point APG to be for bits 0–1.
G = 0 Unguarded storage.
PS = 11 (bin) 8M page.
WT = 1 (bin) Writethrough. Should not matter since this is read-only memory.
V = 1 (bin) Segment is valid.

NOTE

The above information is duplicated on address $400004 since it is an 8M page.

Address $400008 contains L1D2 whose setting is as follows:

L2BA = $00402 Point to level two table to $402000.
APG = $0 Point APG to be for bits 0–1.
G = 0 Unguarded storage.
PS = 01 512K page.
WT = 0 Copyback.
V = 1.

Address $40000C to $40008F contains L1D3 through L1D35. The least-significant bit of each long word should be cleared to make L1D3 through L1D35 invalid.

Address $400090 contains L1D36 whose setting is as follows:

L2BA = $00403 Point to level two table to $403000.
APG = $0 Point APG for bits 0–1.
G = 1 Guarded.
PS = 00 Small page size.
WT = 1 Writethrough. Disabled on level 2.
V = 1 Valid.

Address $400094 to $4000FF contains L1D37 through L1D63. Notice that the least-significant bit of each long word should be cleared to make L1D37 through L1D63 invalid.

Address $400100 to $40017F contains L1D64 through L1D95. The contents of L1D64 through L1D95 for the APG, G, PS, WT and V bits are constant and appear as follows:

APG = $0 Point APG for bits 0–1.
G = 1 Guarded.
PS = 11 8M page size.
WT = 1 Writethrough. Disabled on level 2.
V = 1 Valid.

L2BA for L1D64 and L1D65 = $00404. Point Level two table to $404000.

L2BA for L1D66 and L1D67 = $00405. Point Level two table to $405000.

L2BA for L1D68 and L1D69 = $00406. Point Level two table to $406000.

L2BA for L1D70 and L1D71 = $00407. Point Level two table to $407000.

L2BA for L1D72 and L1D73 = $00408. Point Level two table to $408000.

L2BA for L1D74 and L1D75 = $00409. Point Level two table to $409000.

L2BA for L1D76 and L1D77 = $0040A. Point Level two table to $40A000.

L2BA for L1D78 and L1D79 = $0040B. Point Level two table to $40B000.

L2BA for L1D80 and L1D81 = $0040C. Point Level two table to $40C000.

L2BA for L1D82 and L1D83 = $0040D. Point Level two table to $40D000.

L2BA for L1D84 and L1D85 = $0040E. Point Level two table to $40E000.

L2BA for L1D86 and L1D87 = $0040F. Point Level two table to $40F000.

L2BA for L1D88 and L1D89 = $00410. Point Level two table to $410000.

L2BA for L1D90 and L1D91 = $00411. Point Level two table to $411000.

L2BA for L1D92 and L1D93 = $00412. Point Level two table to $412000.

L2BA for L1D94 and L1D95 = $00413. Point Level two table to $413000.

Address $400180 to $400FFF contain L1D96 through L1D1023. You should clear the LSB of each long word to make L1D96 through L1D1023 invalid.

B.1.9 Level Two Descriptor 1K Resolution Format Register

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

RPN—Real Page Number

This field contains the real page number used in address translation. All 20 bits are used when the page size in the level one descriptor is set to 1K or 4K. The upper 18 bits are used when the page size in level one descriptor is set to 16K. The upper 13 bits are used when the page size in level one descriptor is set to 512K and the upper 9 bits are used when the page size in level one descriptor is set to 8M.

PP1—Page Protection For First 1K Page

Instruction Access:
00 = No access.
01 = Executable from privilege mode only.
10 = Executable from both privilege and problem mode.
11 = Executable from both privilege and problem mode.

Data Access:
00 = No access.
01 = Read/write from privilege mode. No access from problem mode.
10 = Read/write from privilege mode. Read-only from problem mode.
11 = Read/write from both privilege and problem mode.

PP2—Page Protection For Second 1K Page

Instruction Access:
00 = No access.
01 = Executable from privilege mode only.
10 = Executable from both privilege and problem mode.
11 = Executable from both privilege and problem mode.

Data Access:
00 = No access.
01 = Read/write from privilege mode. No access from problem mode.
10 = Read/write from privilege mode. Read-only from problem mode.
11 = Read/write from both privilege and problem mode.

PP3—Page Protection For Third 1K Page

Instruction Access:
00 = No access.
01 = Executable from privilege mode only.
10 = Executable from both privilege and problem mode.
11 = Executable from both privilege and problem mode.

Data Access:
00 = No access.
01 = Read/write from privilege mode. No access from problem mode.
10 = Read/write from privilege mode. Read-only from problem mode.
11 = Read/write from both privilege and problem mode.

PP4—Page Protection For Fourth 1K Page

Instruction Access:
00 = No access.
01 = Executable from privilege mode only.
10 = Executable from both privilege and problem mode.
11 = Executable from both privilege and problem mode.

Data Access:
00 = No access.
01 = Read/write from privilege mode. No access from problem mode.
10 = Read/write from privilege mode. Read-only from problem mode.
11 = Read/write from both privilege and problem mode.

SPS—Small Page Size

0 = 1K or 4K page.
1 = 16K, 512K, or 8M page.

SH—Shared Page

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

CI—Cache Inhibit

0 = Cache is enabled for this entry.
1 = Cache is disabled for this entry.

V—Valid

0 = This page is valid.
1 = This page is invalid.

B.1.10 Level Two Descriptor 4K Resolution Format Register

BITS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
RESET	0									1	0					
BITS	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PP		PPM	C	PPAPV				SPS	SH	CI	V
RESET	0				0		0	0	0				0	0	0	0

RPN—Real Page Number

This field contains the real page number used in address translation. All 20 bits are used when the page size in the level one descriptor is set to 1K or 4K. The upper 18 bits are used when the page size in level one descriptor is set to 16K, the upper 13 bits are used when it is set to 512K, and the upper 9 bits are used it is set to 8M.

PP—Page Protection

Instruction access is as follows:

Extended encoding:
00 = No access.
01 = Executable from privilege mode only.
10 = Reserved.
11 = Reserved.

PowerPC encoding:
00 = Executable from privilege mode only.
01 = Executable from both privilege and problem mode.
10 = Executable from both privilege and problem mode.
11 = Executable from both privilege and problem mode.

Data access is as follows:

Extended encoding:
00 = No access.
01 = Read-only from privilege mode. No access from problem mode.
10 = Reserved.
11 = Reserved.

PowerPC encoding:
00 = Read-only from privilege mode. No access from problem mode.
01 = Read/write from privilege mode. Read-only from problem mode.
10 = Read/write from both privilege and problem mode.
11 = Read-only from both privilege and problem mode.

PPM—Page Protection Mode

0 = PP is set to PowerPC encoding.
1 = PP is set to extended encoding.

C—Change Bit For Entry

0 = No change has been performed (write-protected).
1 = Changed region. Writes are allowed.

PPAPV— Page Protection Access Privilege Or Valid

If the PPCS bit of the Mx_CTR register is 0, the settings are as follows. However, if the page size is larger than 4K, then all 4 bits should have the same value

0 = Subpage is invalid.
1 = Subpage is valid.

If the PPCS bit of the Mx_CTR register is 1 the settings are as follows. Other bit combinations are reserved.

1000 = Hit only for privilege access.
0100 = Hit only for problem access.
1100 = Hit for both privilege and problem access.

SPS—Small Page Size

0 = 1K or 4K page.
1 = 16K, 512K, or 8M page.

SH—Shared Page

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

CI—Cache Inhibit

0 = Cache is enabled for this entry.
1 = Cache is disabled for this entry.

V—Valid Bit

0 = This page is valid.
1 = This page is invalid.

Address $401000 to $401FFF contains the level two descriptor (L2D) for L1D0 to L1D1. Since the pages are set to 8 M, all 1,024 entries of L2D will be constant as follows:

RPN = $0.
PP[20-21]= 00 Instruction executable only by privilege access.
Data access read/write only by privilege access.
PP[22] = 0 PowerPC encoding.
PP[23] = 0 Not changed (write-protect).
PP[24-27] = 1,000 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 0 Cache is enabled.
V = 1 Page is valid.

Address $402000 to 402FFF contain L2D for L1D2. The settings are as follows.

Address range $402000 to 4021FF contains the first 512Kpage and is L2D0 through L2D127. The settings are as follows:

RPN = $00800.
PP[20-21]= 00 Instruction executable only by privilege access.
Data access read/write only by privilege access.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no write-protect).
PP[24-27] = 1,000 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 0 Cache is enabled.
V = 1 Page is valid.

Address 402200 to 4023FF contains the second 512Kpage and is L2D128 through L2D255. The settings are as follows:

RPN = $00880.
PP[20-21]= 00 Instruction executable only by privilege access.
Data access read/write only by privilege access.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no write-protect).
PP[24-27] = 1,000 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 0 Cache is enabled.
V = 1 Page is valid.

Address 402400 to 4025FF contains the third 512K page and is L2D256 through L2D383. The settings are as follows:

RPN = $00900.
PP[20-21]= 00 Instruction executable only by privilege access.
Data access read/write only by privilege access.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no write-protect).
PP[24-27] = 1,000 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 1 Cache is disabled.
V = 1 Page is valid.

Address 402600 to 4027FF contains the fourth 512K page and is L2D384 through L2D511. The settings areas follows:

RPN = $00980.
PP[20-21]= 10 Instruction executable on both privilege and problem mode.
Data access read/write from both privilege and problem mode.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no write-protect).
PP[24-27] = 1,100 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 0 Cache is enabled.
V = 1 Page is valid.

Address 402800 to 4029FF contains the fifth 512K page and is L2D512 through L2D639. The settings are as follows:

RPN = $00A00.
PP[20-21]= 10 Instruction executable on both privilege and problem mode.
Data access read/write from both privilege and problem mode.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no-write protect).
PP[24-27] = 1,100 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 0 Cache is enabled.
V = 1 Page is valid.

Address 402A00 to 402BFF contains the sixth 512K page and is L2D640 through L2D767. The setting is as follows:

RPN = $00A80.
PP[20-21]= 10 Instruction executable on both privilege and problem mode.
Data access read/write from both privilege and problem mode.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no write-protect).
PP[24-27] = 1,100 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 0 Cache is enabled.
V = 1 Page is valid.

Address 402C00 to 402DFF contains the seventh 512K page and is L2D768 through L2D895. The settings are as follows:

RPN = $00B00.
PP[20-21]= 10 Instruction executable on both privilege and problem mode.
Data access read/write from both privilege and problem mode.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no write-protect).
PP[24-27] = 1,100 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 0 Cache is enabled.
V = 1 Page is valid.

Address 402E00 to 402FFF contains the eighth or last 512K page and is L2D896 through L2D1023. The settings are as follows:

RPN = $00B80.
PP[20-21]= 10 Instruction executable on both privilege and problem mode.
Data access read/write from both privilege and problem mode.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no write-protect).
PP[24-27] = 1,100 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 0 Cache is enabled.
V = 1 Page is valid.

Address 403000 to 403FFFF contains the L2D for L1D36. Since only two 16K pages (0 and 256) will be used, other pages that are ‘ON’ should be set so they are invalid. This can be accomplished by clearing the least-significant bit of a long word for address 403004 to 4033FC and 403404 to 403FFC.

Address 403000 contains L2D0 for L1D36. The settings are as follows:

RPN = $09000.
PP[20-21]= 00 Instruction executable on privilege mode.
Data access read/write from privilege mode.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no write-protect).
PP[24-27] = 1,100 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 1 Cache is disabled.
V = 1 Page is valid.

Address 403400 contains L2D100 for L1D36. The settings are as follows:

RPN = $09100.
PP[20-21]= 00 Instruction executable on privilege access.
Data access read/write on privilege access.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Changed (no write-protect).
PP[24-27] = 1,100 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 1 Cache is disabled.
V = 1 Page is valid.

Address $404000 to $413FFF contains a level two descriptor (L2D) for L1D64 to L1D95. Since the protection is the same throughout the 800M range, the PP[20-27], SPS, SH, and V bits are constant. The settings for those bits are as follows:

PP[20-21]= 00 Instruction executable only by privilege mode.
Data access read/write only by privilege mode.
PP[22] = 0 PowerPC encoding.
PP[23] = 1 Not changed (no write-protect).
PP[24-27] = 1,000 hit only for privilege access.
$\overline{SPS}$ = 1 16K.
SH = 1 Disable ASID CMP.
CI = 1 Cache is disabled.
V = 1 Page is valid.

For the RPN bits with 8M pages, the value changes every 1K entry:

For L1D40 and L1D41, RPN = $10000.
For L1D42 and L1D43, RPN = $10800.
For L1D44 and L1D45, RPN = $11000.
For L1D46 and L1D47, RPN = $11800.
For L1D48 and L1D49, RPN = $12000.
For L1D4A and L1D4B, RPN = $12800.
For L1D4C and L1D4D, RPN = $13000.
For L1D4E and L1D4F, RPN = $13800.
For L1D50 and L1D51, RPN = $14000.
For L1D52 and L1D53, RPN = $14800.
For L1D54 and L1D55, RPN = $15000.
For L1D56 and L1D57, RPN = $15800.
For L1D58 and L1D59, RPN = $16000.
For L1D5A and L1D5B, RPN = $16800.
For L1D5C and L1D5D, RPN = $17000.
For L1D5E and L1D5F, RPN = $17800.

B.2 CONFIGURING THE MPC801 MEMORY CONTROLLER

A fundamental design goal of the PowerQUICC was to have an easy interface to other system components. To achieve this goal, the system interface unit incorporates a flexible memory controller. Each of the eight memory banks can be configured to support various memory types (Flash EPROM, SRAM, DRAM, EDO DRAM, or synchronous DRAM). The assertion and negation of the memory control signals ($\overline{CS}$, $\overline{WE}$, $\overline{OE}$, $\overline{BS}$, and general-purpose lines) are defined by the software up to a quarter of a system clock resolution, so it is likely the MPC801 can interface to any type of memory. The PowerQUICC facilitates the different types of memory by means of the three machines included in the memory controller:

- The general-purpose chip-select machine (GPCM)
- The user-programmable machine A (UPMA)
- The user-programmable machine B (UPMB)

Each of the eight chip-selects can be controlled by any of the three machines. The GPCM handles standard accesses to SRAM, EPROM, FEPROM, and simple peripherals. Port sizes of 8-, 16-, and 32-bit are supported with individual byte write enable ($\overline{WE}$) and output enable ($\overline{OE}$) control. Therefore, all simple memory interfaces are supported gluelessly with the GPCM.

The user-rogrammable machines (UPMs) offer great flexibility in the definition of memory cycles. Each UPM can control the address multiplexing required for DRAM devices, the timing of four byte enables (BE0:3), and the timing of six general-purpose lines ($\overline{GPL0:5}$). The UPM is completely controlled by the software and it runs a specific pattern for a programmable number of clock cycles.

For each system clock a separate 32-bit word defines the behavior of the address multiplexing, address incrementing, and control signal assertion and negation to a quarter of a system clock resolution. It also defines the transfer acknowledge assertion. Different behavior can be defined for different types of access (read, write, burst read, and burst write).

B.2.1 General Configuration

After power-up, certain registers need initialization, assuming that initial system configuration is complete. The features of each memory bank are programmed through three registers—option register (OR), base register (BR), and machine mode register (MMR). There is an individual OR and BR for each of the eight memory banks.

B.2.2 SRAM Configuration

This application uses a 512K SRAM bank arranged in a 32-bit port. It is implemented using four Motorola MC6226A 128K × 8 CMOS fast-static RAMs. These are available with access times of 20 to 45ns. The 45ns are sufficient devices that interact with the MPC801 using the 2-clock fast termination cycle. The $\overline{CS3}$* signal is used to select the SRAM memory bank. Individual byte strobes are provide via the $\overline{WE}$*[3:0] signals. The $\overline{OE}$* signal is used to control the output enable of the SRAM.

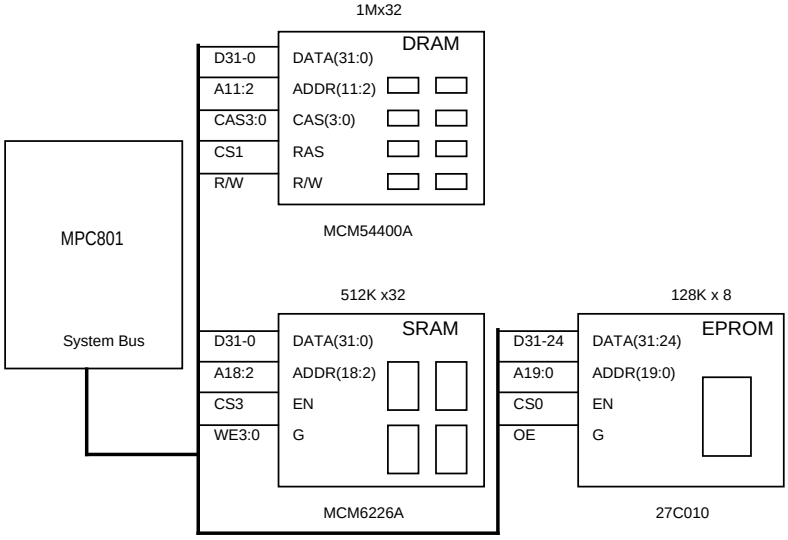


Figure B-1. General System Configuration

To select the memory bank as an SRAM bank, the MS[0:1] bits in the base register should be set at 00. This selects the memory bank to be controlled by the GPCM. When the bank is selected as a GPCM bank, the $\overline{BI}$ bit in the option register should be set.

Defining the memory map of the SRAM bank is achieved by setting several options. The SRAM port size can be defined as 32-bit by setting the PS[0:1] bits in the base register to 00. The base location of the memory bank is selected by setting the BA[0:16] bits in the base register. This allows the block to be defined on 32K boundaries. For example, if a base location of 20000000H is required, these bits would be set as 0010 0000 0000 0000 0B.

The size of the SRAM bank is defined by the address mask bits (AM[0:16]) in the option register. This enables blocks sizes between 32K and 4G to be selected. For example, if a block size of 1M, 80000H, these bits should be set to 0000 0000 0000 1111 1B. PARE in the base register is used to enable parity checking. Since parity it is not required in this design, this bit is cleared. Other functions, such as write protect (WP), and address type operation (AT[0:2]/ATM[0:2]) can be defined as needed.

Table B-9. Option Register

BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
AM0	AM1	AM2	AM3	AM4	AM5	AM6	AM7

BIT 8	BIT 9	BIT 10	BIT 11	BIT 12	BIT 13	BIT 14	BIT 15
AM8	AM9	AM10	AM11	AM12	AM13	AM14	AM15

BIT 16	BIT 17	BIT 18	BIT 19	BIT 20	BIT 21	BIT 22	BIT 23
AM16	ATM0	ATM1	ATM2	CSNT/SAM	ACS0	ACS1	BI

BIT 24	BIT 25	BIT 26	BIT 27	BIT 28	BIT 29	BIT 30	BIT 31
SCY0	SCY1	SCY2	SCY3	SETA	TRLX	RES	RES

Table B-10. Base Register

BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
BA0	BA1	BA2	BA3	BA4	BA5	BA6	BA7

BIT 8	BIT 9	BIT 10	BIT 11	BIT 12	BIT 13	BIT 14	BIT 15
BA8	BA9	BA10	BA11	BA12	BA13	BA14	BA15

BIT 16	BIT 17	BIT 18	BIT 19	BIT 20	BIT 21	BIT 22	BIT 23
BA16	AT0	AT1	AT2	PS0	PS1	PARE	WP

BIT 24	BIT 25	BIT 26	BIT 27	BIT 28	BIT 29	BIT 30	BIT 31
MS0	MS1	RES	RES	RES	RES	RES	V

The memory controller can alter the cycle length and timing characteristics by configuring the option and base registers parameters. To determine actual values, a timing analysis of the interface is required. Figure B-2 illustrates a fast termination read cycle. This shows an access with no compromise in the timings. Notice that the memory controller asserts the $\overline{CS}$ from the rising edge of the SO and the PowerQUICC does not sample the data for two full clock cycles. This means that the access times required by SRAMs are greatly reduced in comparison to standard interfaces.

If a system contains slow peripherals with long data hold times, contention could occur on the data bus between back-to-back cycles. By setting the ACS[0:1] or CSNT bits in the option register, the $\overline{CS}$ signal can be delayed by a quarter or half a clock. Similarly, if the CSNT bit is set, the $\overline{CS}$ and $\overline{WE}$ signals are negated a quarter of a clock earlier. However, in this example it is assumed this is not the case and the ACS[0:1] bits are set to 00 and CSNT is cleared. In an MPC801 design there are three critical timings for a fast termination read cycle—SRAM enable access time (tACS), address access time (tAA), and output enable time(tOE). The equation below provides a calculation of tACS:

tACS =2 tcyc – t18 – t22

Where,

tcyc = Period of processor clock = 40ns for a 25MHz clock
t22 = CLKOUT high to $\overline{CS}$ = 0-20ns
t18 = Data-in to CLKOUT high (data setup) = 6ns

Therefore,

tACS = 80ns – 6ns – 20ns
tACS = 54ns

Now the address access time (tAA) can also be calculated in a similar manner:

tAA = 2 tcyc – t18 – t8

Where, t8 = CLKOUT high to address valid 19ns.

Therefore,

tAA = 80ns – 6ns – 19ns
tAA = 55ns

Finally, the output enable time (tOE) is calculated as tOE = tcyc – t18 – t25.

Where,

t25 = CLKOUT high to $\overline{OE}$ asserted 11ns
tOE = 40ns – 6ns – 11ns
tOE = 23ns

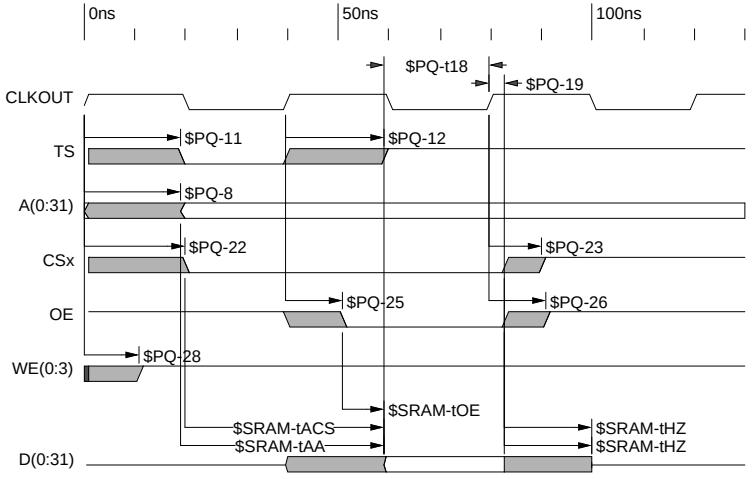


Figure B-2. SRAM Read Timing Analysis

B

For most industry standard devices (such as the MCM6226A, tAA,and tACS) are similar. Therefore, a memory with an access time of at least 54ns is required for fast termination cycles. This is by no means a fast memory speed and it easily meets the slowest access time for the MCM6226A (45ns). The output enable time required is 23ns, which is well within the parameter of even slow static devices, and 15ns for a 45ns MCM6226A.

Figure B-3 illustrates the critical timing (tDW) data valid to end of write for a fast termination write cycle.

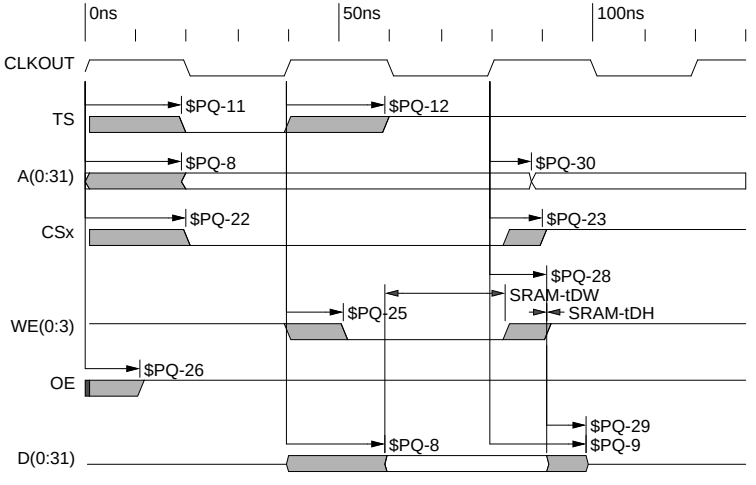


Figure B-3. SRAM Write Timing Analysis

Now in this system, tDW = tcyc – t8. Where, t8 = CLKOUT high to data valid 19ns. Therefore, tDW = 2ns. For most standard SRAMs, tDW is approximately half the access time. For example, for the MC6226A at 45ns, this is defined as 20ns, thus allowing for a 1ns margin. Notice that this is not taking into account the clock skew of the system. If it is, then a faster part might be required.

The attributes of the SRAM memory access cycles are determined and the timing characteristics of the SRAM can be configured. There are several relevant bits to aid in programming. The option register's SCY[0:3] bits initialize the number of wait states required. For a fast termination cycle, this is programmed to 0H, which defines a 2-cycle access when using the SRAM. TRLX, CSNT, and ACS[0:1] bits in the option register all relax the timing parameters of the access, which is useful for slow peripherals that require additional address setup time. However, none of these options are required when using FSRAM.

To determine the how the $\overline{TA}$ signal is generated internally, the SETA bit in the option register should be reset. The final bit to be set is the valid (V) bit of the base register. The $\overline{CS}$ signal is not asserted until this bit is set. This results in the following register settings:

BR = 20000001.
OR = 000F8100.

B.2.3 EPROM Configuration

This application uses a single 1Mb 27C010 as a boot EPROM and it is arranged as an 8-bit port. Initially, this is selected by the CONFIG pins at reset. The EPROM is enabled by the $\overline{CS0}$ line and its output enable is controlled by $\overline{OE}$, as illustrated in Figure B-4. After reset, $\overline{CS0}$ acts as the global chip-select for the whole system so it must be reconfigured by the initialization code.

To select the memory bank as an EPROM bank, the MS[0:1] bits in the base register should be set at 00 so the memory bank will be controlled by the GPCM. When the bank is selected as a GPCM bank, the $\overline{BI}$ bit in the option register should be set. The memory map of the EPROM bank is defined by setting several options. The EPROM port size is made 8-bit by setting the PS[0:1] bits in the base register to 01.

To select the base location of the memory bank the BA[0:16] bits in the base register must be set. If a base location of 00000000H is required, these bits should be set to 00000H. The size of the EPROM bank is defined by the address mask AM[0:16] bits in the option register. If it is a block size of 128K, 20000H, these bits should be set to 00038H 0000 0000 0000 0001 1B. For an EPROM bank, it is recommended that the WP bit in the base register be set so that any write access results in a no match to this memory bank. When the hardware bus monitor is enabled, it causes the $\overline{TEA}$. signal to assert.

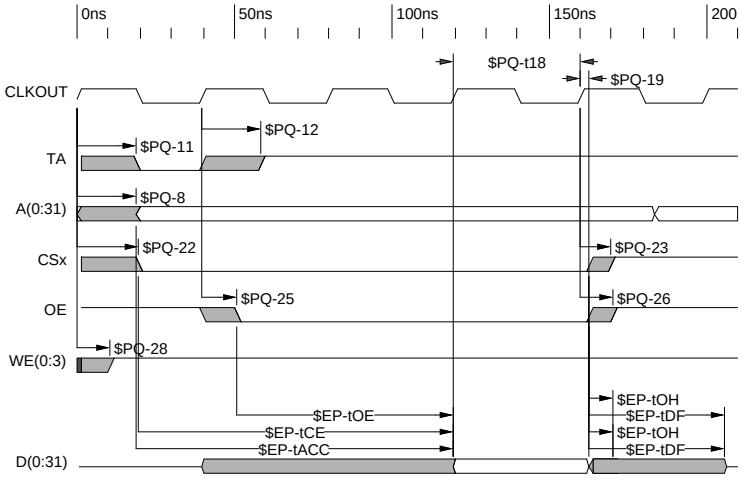


Figure B-4. EPROM 1 Wait State Read

B

The critical timing analysis for the 27c010 is calculated as follows. Figure B-4 illustrates an EPROM read cycle. In an MC68360 design there are three critical timings for the read cycle. They are the EPROM chip enable time, tCE, the address access time, tACC and tOE, and the output enable time. The equation below shows the calculation of tCE:

tCE =2 tcyc + twait- t18 - t22

Where:

twait = The number of wait states required, 40ns for each wait state
tcyc = Period of processor clock = 40ns for a 25MHz clock
t22 = CLKOUT high to $\overline{CS}$ = 0-20ns
t18 = Data-in to CLKOUT high (data setup) = 6ns

Therefore, for two wait states:

tCE = 80ns + 80ns – 6ns – 20ns
tCE= 134ns

The address access time (tACC) can also be calculated in a similar manner as follows:

tACC = 2 tcyc + twait – t18 – t8

Where,

t8 = CLKOUT high to address valid 19ns

Therefore for 2 wait states:

tACC = 80ns + 80ns – 6ns – 19ns
tACC = 135ns

Finally, the output enable time tOE is calculated:

tOE = tcyc + twait – t18 – t25

Where,

t25 = CLKOUT high to $\overline{OE}$ asserted 11ns
tOE = 40ns + 80ns – 6ns – 11ns
tOE = 103ns

For most EPROMs, tAA is the same value as tCE (the critical value). For a five cycle read, accessing a 120ns EPROM is required to meet the critical timings. It is also worth verifying that the $\overline{CS}$ to data HI-Z (tDF) is fast enough to avoid corrupting the next processor cycle. For a 120ns part is 35ns.

Therefore,

Data Driven = tDF + t23

Where,

t23 is CLKOUT to $\overline{CS}$ negated 3-10ns

So,

Max Time Data Drive = 10ns + 25ns = 45ns

For an SRAM write cycle, data can be driven after:

tcyc + t8 = 40ns + 10ns

This allows a grace period of approximately 50–45ns = 5ns between the time the EPROM is three-stated and the SRAM write cycle begins.

B.2.4 DRAM Configuration

The DRAM in this application is an 8M bank arranged in a 32-bit port. The parts chosen are Motorola MCM54400A 1M × 4 CMOS dynamic RAMs that are ideal for interfacing to the memory controller of the MPC801. The MPC801 provides all the control signals for a glueless interface. Row and column address strobes are supplied by the $\overline{CS}$ and $\overline{BS}$ bits. The RD/$\overline{WR}$ bit drives the read/write enable signal on the DRAM. Finally, the MPC801 provides row and column addresses controlled by internal multiplexors. A DRAM single in-line memory module (SIMM) could also be used.

DRAM is interfaced to the MPC801 using the UPM for memory control signal assertion and negation down to a quarter of a system clock resolution. This is achieved by generating a version of the system clock phase shifted by 90• (GCLK1). A second clock, GCLK1, is in phase with the system clock and a 32-bit word defines the behavior of the memory cycle for each clock, as illustrated in Figure B-10. Notice that the first four bits define the state of the $\overline{CS}$ bit for each quarter clock phase and, likewise, the next four bits define the state of the $\overline{BS}$[0:3] bits. Figure B-11 shows how these relate to CLKOUT. Asserting and negating the $\overline{CS}$ and $\overline{BS}$[0:3] bits can be controlled down to a 10ns resolution at 25MHz or a 6ns resolution at 40MHz.

Table B-11. UPM Word Structure

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	RES	RES

BIT 24	BIT 25	BIT 26	BIT 27	BIT 28	BIT 29	BIT 30	BIT 31
LOOP	EXEN	AMX0	AMX1	NA	UTA	TODT	LAST

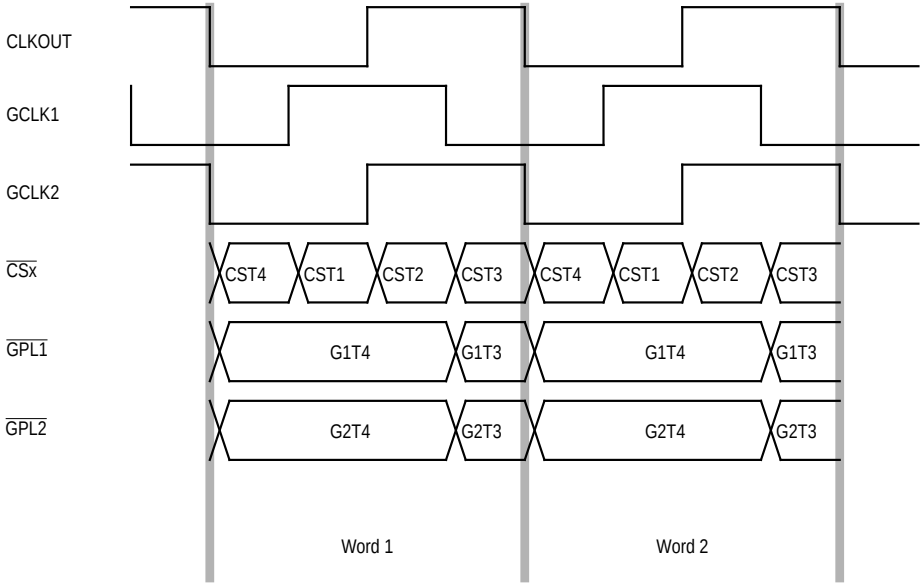


Figure B-5. UPM Signal Timings

The behavior of the six general-purpose lines can be controlled to a smaller degree, but they can still be asserted at two different points per clock. These are at the falling edge of GCLK2 and GCLK1. The operation of these bits are defined by bits 8–21 of the UPM RAM word. The LOOP bit is used to repeat a pattern for a given number of cycles and to implement wait states into a cycle or repeat a set pattern for a burst cycle.

The EXEN bit is used to provide a clean exit from a cycle when an exception occurs and the AMX bits are used to define the address multiplexing. Additionally, the NA bit is used to increment the address when a burst cycle is implemented and the UTA bit is used to define to sampling point for the $\overline{TA}$ signal. The TODT bit is used to define the required time for a $\overline{RAS}$ precharge. Finally, the LAST bit defines a UPM RAM word as the last in a particular cycle.

Figure B-6 illustrates the configuration of the UPM RAM in the MPC801. It consists of a block of 64 32-bit entries. Notice that there are six different entry points to the UPM RAM—burst read, burst write, single read, single write, the periodic timer (used for refresh), and exceptions. The system interface unit decodes the correct cycle type and jumps to the appropriate point in the UPM table. For example, for a burst read it jumps to location 08H, then decodes the UPM word at that location, and asserts the external signals accordingly. On each system clock it jumps to the next UPM entry until the LAST bit is set and the decode is terminated.

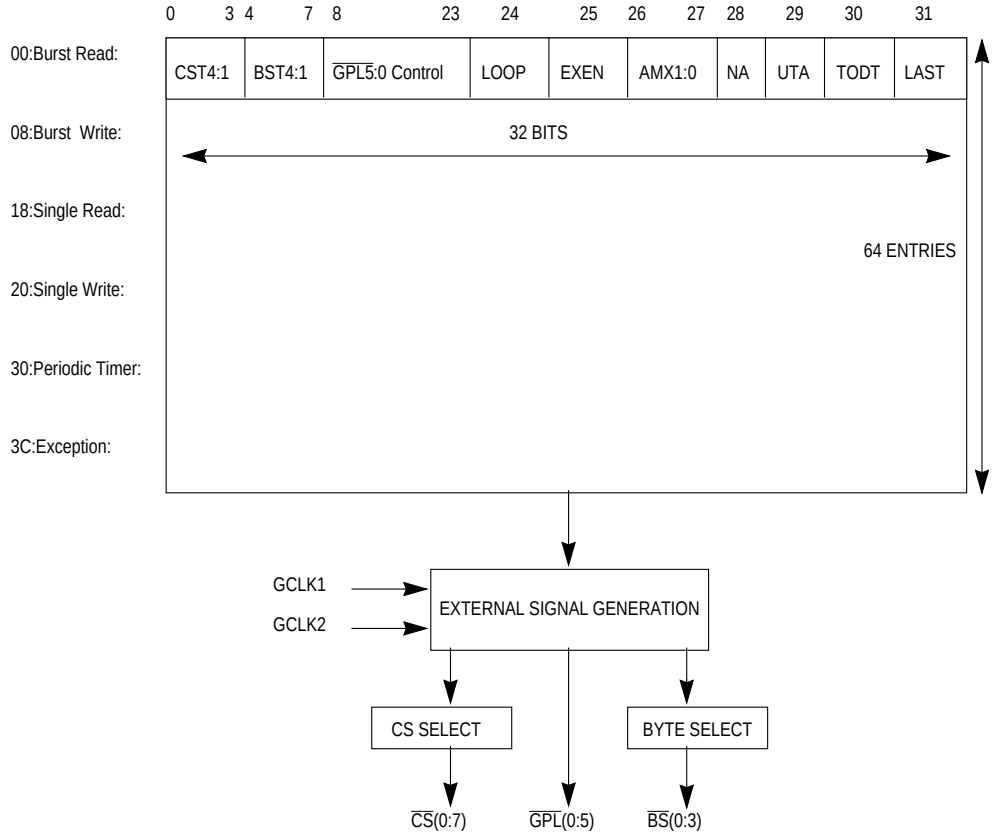


Figure B-6. UPM RAM Configuration

Because the UPM's is flexible, there is no single correct configuration for a particular memory. One example configuration for the DRAM described above would be to select the memory bank as a DRAM bank controlled by the UPMA and set the MS[0:1] bits in the base register at 10. The memory map of the DRAM bank is defined by setting several options similar to the SRAM. The DRAM port size can be defined as 32-bit by setting the PS[0:1] bits in the base register to 00.

The base location of the memory bank is selected by setting the BA[0:16] bits in the base register, which allows the block to be defined on 32K boundaries. For example, if a base location of 10000000H is required, these bits would be set to 1000 0000 0000 0000 0B. The size of the DRAM bank is defined by the AM[0:16] bits in the option register, which enables 32K and 4G block sizes to be selected. For example, if a block is size 8M (8000000H), these bits should be set to 0000 0000 0111 1111 1B.

The SAM bit in the option register is used to define the initial address multiplexing for the DRAM and should be set to 1. This actual address multiplexing is defined by the AMA[0:2] bits in the MAMR. For a 32-bit port using devices with 10 row addresses and 10 column addresses, these should be set to 010. As with DRAM, parity is not required in this design, therefore the PARE bit is cleared in the base register. Again, other functions, such as write protect (WP) and address type operation AT[0:2]/ATM[0:2] can be defined by you.

The DRAM refresh controller is supported by the PTA[0:7] bits in the MAMR register and is enabled by the PTAE bit. Motorola's MC54400A DRAM requires that each bit be refreshed every 16 ms by cycling through the 1,024 row addresses in sequence within the specified refresh time. Therefore, a refresh cycle is required every 15.6µs. The PTA[0:7] bits in the MAMR and the PTP[0:7] bits in the MPTPR register determine this value using the following equation:

Refresh period = (PTA)/(system clk/PTP[0:7])

Where:

Refresh period = 15.6
System clk = 25MHz
PTP[0:7] = 32 (defined in the MPTPR)

Solving for RFCNT:

RFCNT = 15.6µs $\times$ 25MHz/32
RFCNT = 12.1875

Since PTA must be an integer, it is rounded down to 24 decimal, thus allowing a refresh period of 15.36µs. PTA[0:7] is then programmed with the value 0C Hex. To achieve a 3-cycle access to the DRAM, 60ns parts must be used. Figure B-7 illustrates the timing of this interface. There are many timing constraints for the DRAMs, but five major timing constraints determine the suitability of a device:

- tAA—Access time from column address
- tCAC—Access time from column address strobe
- tRAC—Access time from row address strobe
- tRC—Random read or write cycle time
- tRP—$\overline{\text{RAS}}$ precharge time

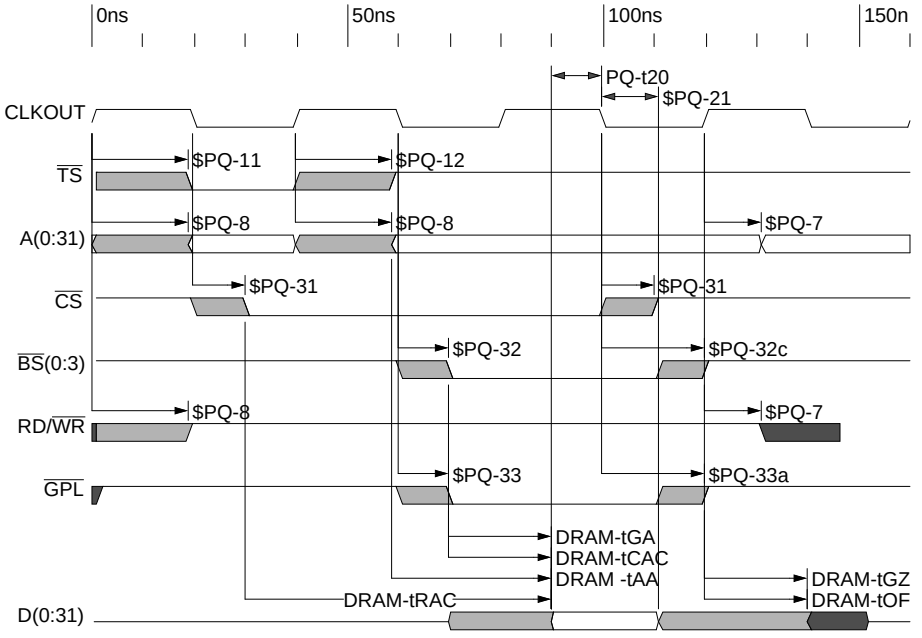


Figure B-7. DRAM 3-Cycle Read

As illustrated in Figure B-11, equations can be generated to calculate these access times:

tAA = 1.5tcyc – t8 – t20

Where,

tcyc = Period of processor clock = 40ns for a 25MHz clock
t8 = CLKOUT to Address Valid = 10–19ns
t20 = Data-in to CLKOUT falling edge (data setup) = 4ns

Therefore,

tAA = 60ns – 20ns – 4ns
tAA = 36ns

Similarly, expressions can be developed for tRAC and tCAC as follows:

tRAC = 2tcyc – t31 – t20
tCAC = tcyc – t32 – t20

Where,

t31 = CLKOUT falling edge to $\overline{CS}$ asserted = 0–10ns
t32 = CLKOUT falling edge to $\overline{CS}$ asserted = 0–10ns

Therefore,

tRAC = 80ns – 10ns – 4ns
tRAC= 66ns

And,

tCAC = 40ns – 10ns – 4ns
tCAC = 26ns

tRAC is known as the device access time and the nearest standard DRAM available with this access time is a 60ns device. So for a 3-cycle access, a 60ns DRAM is required.

To achieve this timing, the data output from the DRAM should be sampled with the falling edge of the CLKOUT and the GPL_A4DIS (19) bit in the MAMR should be defined as 1 (the default value). Finally, to ensure tRC (cycle time) and tRP $\overline{RAS}$ (precharge time), the DSA[0:1] bits in the MAMR should be set. If the disable timer is not enabled, the precharge time given to the strobe is at least:

$$tRC = tcyc + t31_{min} – t31_{max} = 40ns + 0 – 10ns = 30ns$$

To ensure the required tRC on the DRAM (40ns), one clock cycle of the disable timer must be added. The DSA bits in the MAMR should be 00, and a DRAM read access must be programmed as follows.

First Clock:

$\overline{CS}$ is asserted on the falling edge of GCLK2 so: CST[4:1] = 0000 B
$\overline{BS}$ are not asserted so BST[4:1] = 1111 B
$\overline{GPL1}$ (used for $\overline{OE}$) is not asserted so all GPL bits are 1
LOOP = 0 – Since this pattern is not repeated
EXEN = 0 – No exception is generated after this clock
AMX(0:1) = 10 – So the address is multiplexed
NA = 0 – No address increment
UTA = 1 – No $\overline{TA}$
TODT = 0 – No precharge time yet
LAST = 0 – Not the last entry

Giving the value: 0000 1111 1111 1111 1111 1111 0010 0100 B

Second Clock:

$\overline{CS}$ is still asserted so: CST[4:1] = 0000 B
$\overline{BS}$ are not asserted so BST[4:1] = 0000 B
$\overline{GPL1}$ (used for $\overline{OE}$) is asserted so all G1T4 and G1T3 are 0, all other bits are 1
LOOP = 0 – Since this pattern is not repeated
EXEN = 0 – No exception is generated after this clock
AMX[0:1] = 00 – So the address is not multiplexed
NA = 0 – No address increment
DLT3 = 1 – Sample data on the falling edge of CLKOUT
UTA = 0 – $\overline{TA}$ asserted
TODT = 0 – No precharge time yet
LAST = 0 – Not the last entry

Giving the value: 0000 0000 1111 0011 1111 1111 0000 0000 B

Third Clock:

$\overline{CS}$ is negated so: CST(4:1) = 1111 B
$\overline{BS}$ are negated so BST(4:1) = 0001 B
$\overline{GPL1}$ (used for $\overline{OE}$) is negated so G1T4 = 1and G1T3 are 1, all other bits are 1
LOOP = 0 – Since this pattern is not repeated
EXEN = 1 – Exceptions can be generated after this clock
AMX(0:1) = 00 – So the address is not multiplexed
NA = 0 – No address increment
UTA = 1 – $\overline{TA}$ negated
TODT = 1 – Precharge time as per the DSA in MAMR
LAST = 1 – The last entry

Giving the final value as: 1111 0001 1111 1111 1111 1111 0100 0111 B

These values should be placed into the UPM RAM at 18 Hex (the starting point for a single read cycle).

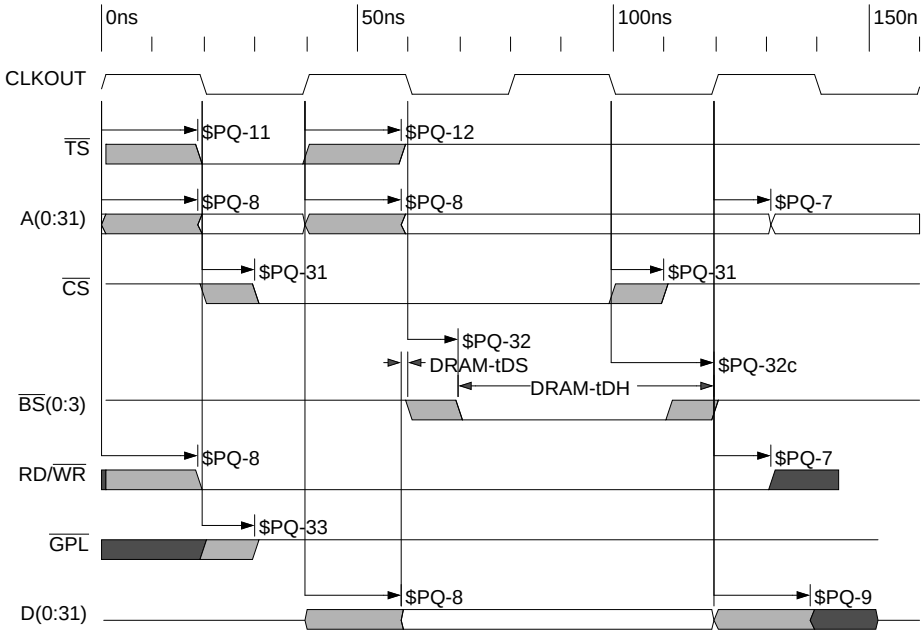


Figure B-8. DRAM 3-Cycle Write

The DRAM data setup (tDS) and data hold (tDH) timings are easily met by t8 and t9 timings for the MPC801. Notice that a full timing analysis should be carried out before implementing a design. The UPM programming for the write cycle is as follows:

0000 1111 1111 1111 1111 1111 0010 0100
0000 0000 1111 1111 1111 1111 0000 0000
1111 0001 1111 1111 1111 1111 0100 0111

The MPC801 also supports burst accesses to memory. Figure B-9 and Figure B-10 illustrate the timings for burst accesses for read and write. Notice that they only show a single burst access for clarity. Normally, the MPC801 will have a four beat burst in this case 3,2,2,2. To use the loop option in the UPM machine, the RLFA and WLFA bits in the MAMR should be set to 0011.

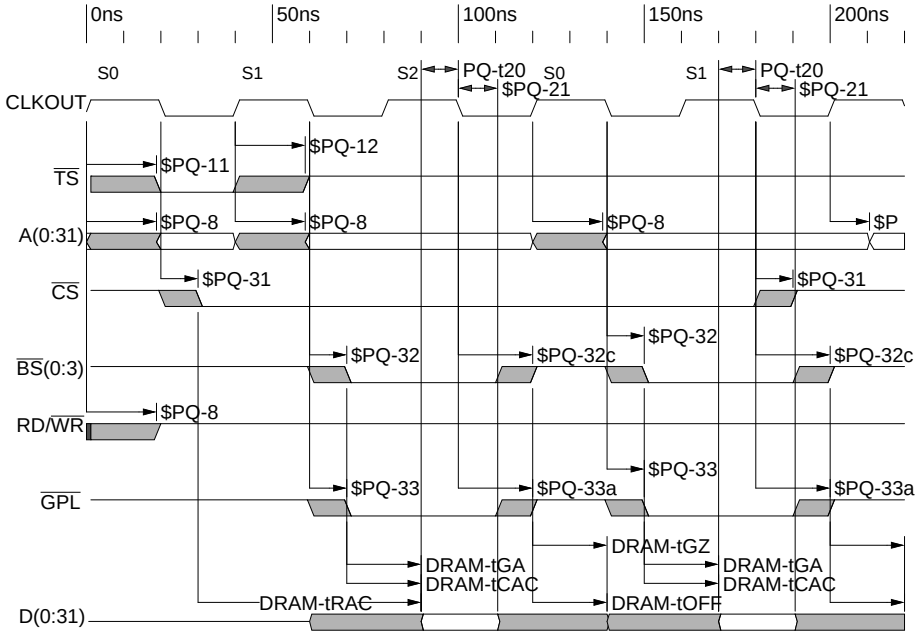


Figure B-9. Burst Read Access

UPM Entries:

0000 1111 1111 1111 1111 1111 0010 0100 B
0000 0000 1111 0011 1111 1111 1000 0000 B
0000 0001 1111 0011 1111 1111 1100 1100 B
0000 0000 1111 0011 1111 1111 0000 0000 B
1111 0001 1111 1111 1111 1111 0100 0111 B

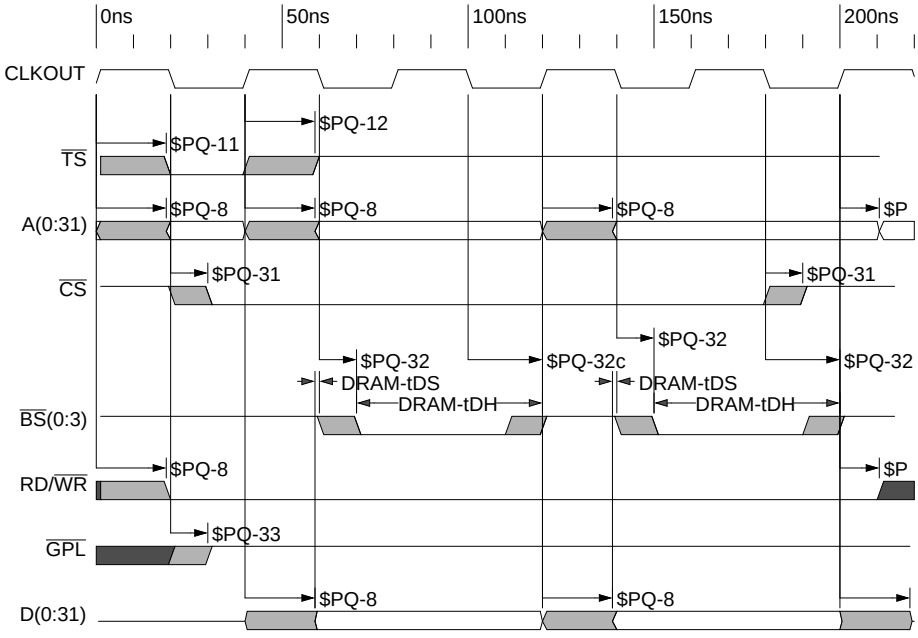


Figure B-10. Write Burst Access

UPM Entries:

0000 1111 1111 1111 1111 1111 0010 0100 B
0000 0000 1111 1111 1111 1111 1000 0000 B
0000 0001 1111 1111 1111 1111 1100 1100 B
0000 0000 1111 1111 1111 1111 0000 0000 B
1111 0001 1111 1111 1111 1111 0100 0111 B

B.3 PORTING TO THE POWERQUICC

As the next member in the family of integrated data communications controllers, the PowerQUICC retains a great amount of similarity between it and previous members of the data communications family (MC68302 IMP and the MC68360 QUICC). The PowerQUICC is a natural migration from the MC68360, providing greater serial power and core performance in a highly integrated package. It is important to remember that there are still many differences between the original QUICC and the PowerQUICC. However, it is not always safe to assume that a function in the PowerQUICC will work like the same function in the original QUICC.

B.4 USING THE POWERPC CORE

The core of the PowerQUICC is now based on the PowerPC architecture rather than the MC68000. The processor architecture of the PowerPC is very different from that of the MC68000. It is recommended that a designer enroll in a course about PowerPC architecture before starting a PowerQUICC design.

B.5 BIT LABELING

It is important to note that the bit labeling of the PowerPC programming environment is the opposite of the MC68000's programming environment. The actual arrangement of the bits is the same, but the high bit is referred to as Bit 0 and the low bit as Bit 31.

B.5.1 Code Portability

The PowerPC core is not binary compatible with the CPU32+ core on the MC68360. Therefore, any code written will have to be ported to the new processor. If code was written in a high-level language such as 'C', a cross complier can provide the simplest code migration path. Any MC68000 assembly code will have to be rewritten in C or PowerPC assembly language.

B.6 CACHE

The PowerQUICC has both an address and data cache. This is different from the CPU32+ core of the MC68360, which had no cache at all. The cache allows the processor to run code much faster, but can create pitfalls for both overall system performance and debugging capability.

B.6.1 Cache Performance Impact

A cache can improve core performance dramatically if it is used effectively. However, turning on the cache does not guarantee higher performance. The degree to which a cache will help is directly related to the memory access pattern of the program running on the core. Actually, the performance of a core can sometimes decrease. A document of this size and breadth cannot effectively address the ways that a program can be optimized to achieve better cache performance.

Some documented methods that can yield a significant improvement in access speed include locking cache blocks, making areas of memory uncacheable, optimizing program flow, and optimizing data access patterns. These techniques are complex and their effectiveness can be altered by a simple program recompilation. Caches can also complicate a benchmarking process. A benchmark, such as Dhrystone, will fit entirely in a cache and can give misleading indications of processor and application performance. Because of this, it is recommended that segments of code are run on an evaluation board to determine the performance that can be expected from a specific application.

B.6.2 Data Coherency

Changes made in the cache are not made to memory when the cache is in writeback mode. This creates data coherency problems if another bus master (an SDMA, IDMA, or external processor) accesses a memory location that has a more recently updated value in the cache. This can also result in data loss or program execution problems. Thus, writethrough mode should be used for any memory area that can be accessed by another bus master.

B.6.3 Debugging

The CPU32+ of the MC68360 fetches its instructions from the system bus every time a particular instruction is executed. By monitoring the external bus, it is easy to determine which instructions are executing at any given time. Also, areas of memory can be monitored to determine when reads and writes occur.

When a cache is placed between the core and the system bus, many of the instructions that are being executed should be out of the sight of the system, thus resulting in better performance. Ideally, the core fetches instructions from cache rather than external memory, which makes the fetches invisible, thus complicating the debug process. The same restriction applies to data reads and, in some cases, writes. This restriction is familiar to those who have worked with a MC68040 or MC68060. However, the PowerQUICC has some additional functions in its background debug mode that gives it more debugging visibility than the MC68040 has.

B.7 MEMORY MANAGEMENT UNIT

Another function of the PowerPC core is that the memory management unit allows the use of virtual memory and special caching options. The memory management unit on the PowerQUICC cannot be disabled and using it can be complex.

B.8 REAL-TIME OPERATING SYSTEMS

A real-time operating system (RTOS) can make using the memory management unit on the PowerQUICC simple. Most commercial RTOS implementations are designed to handle memory management unit and interrupt schemes, which allow the software designer to concentrate on more application-specific code than the subtleties of the memory management unit and interrupts. When porting code from the MC68360 QUICC, keep the memory management unit, cache, and interrupt handling as special cases.

B

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X