MC88410UM/AD

MC88410

SECONDARY CACHE CONTROLLER USER'S MANUAL

MOTOROLA

MOTOROLA

MC88410
Secondary Cache Controller
User's Manual

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Section 3
Signal Description

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Section 5
System Bus Interface

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Concluded)

LIST OF TABLES

LIST OF TABLES (Concluded)

SECTION 1
OVERVIEW

The MC88410 is a highly integrated secondary cache controller that reduces both memory latency and system bus use while extending multiprocessing capabilities to achieve a higher level of system performance. The MC88410 secondary cache controller together with the MCM62110 fast static RAM provide a complete secondary cache solution for both single processor and multiprocessor environments. The MC88410 provides tag, control, and buffering for 1/4-Mbyte, 1/2-Mbyte, and 1-Mbyte secondary cache configurations, all in a single-chip cache controller. When used with the MC88110 RISC microprocessor and MCM62110 secondary cache RAM array, the MC88410 requires no external programming, provides bus arbitration for the MC88110, maintains cache coherency, and eliminates the need for external logic between the microprocessor, the secondary cache, and the system bus.

The MC88410 and MCM62110 array are designed to provide low latency memory accesses. Initial secondary cache accesses incur only one wait state to the processor while subsequent transfers in a burst incur zero wait states. In addition, data streaming to the processor reduces the penalty on secondary cache misses.

The MC88410 extends multiprocessing capability by significantly reducing system bus bandwidth consumption. The increased bus availability, along with the hardware enforced cache coherency protocol of the MC88410, enables the implementation of dual-bus systems and scalable shared-bus multiprocessing systems. In addition, the MC88410 extends system flexibility by providing a choice of the secondary cache line sizes, order of burst transfers, and system bus clock frequencies.

1.1 MC88410 FEATURE LIST

The major features of the MC88410 secondary cache controller are as follows:

- Improved system performance
  - —One wait state on initial access and zero wait state on burst subsequent accesses
  - —Reduced secondary cache miss penalty by streaming data to the processor
  - —Decreased system bus bandwidth used by the processor
  - —Secondary cache write-back or write-through policy as specified by the processor.
- Integrated secondary cache control functions
  - —Tag, control, and buffering for up to 1 Mbyte of secondary cache RAM
  - —No processor/cache glue logic for the MC88110 and MCM62110 array
  - —Processor bus arbitration for a single MC88110/MC88410 combination.

- Complete hardware support for multiprocessor applications
  - —Vertical cache coherency between the MC88410 and the MC88110
  - —Lateral cache coherency between multiple MC88110s and MC88410 pairs
  - —Bus pipelining to allow efficient interleaving of system resources.
- System Configuration Flexibility
  - —Support for 1/4-Mbyte or 1/2-Mbyte cache (with 32K x 9 SRAMs), or 1-Mbyte cache (with 128K x 9 SRAMs)
  - —Double secondary cache size (two MCM62110 arrays) using one (Figure 1-2) or two (Figure 1-3) system buses with a single MC88110 and two MC88410s
  - —Secondary cache line size of 32 or 64 bytes
  - —Zero-word-first or critical-word-first burst order
  - —Full-speed or half-speed system bus
  - —Both MC88110 bus and MC88410 system bus easily accommodated with single memory design.
- Access for system level diagnostics
  - —Write/read/compare testing of tags and cache RAM array
  - —IEEE 1149.1 JTAG boundary scan.

1.2 MC88410 BENEFITS IN SINGLE PROCESSOR SYSTEMS

The MC88410 can be used in single MC88110 systems to significantly enhance performance as well as support diverse system configurations. The MC88410 provides a high degree of integration through its single-chip tag, control, and buffering for 1/4-Mbyte, 1/2-Mbyte, or 1-Mbyte secondary caches. The arbiter of the MC88410 can be enabled to provide arbitration at the processor interface. No external logic is required between a single MC88410 and a single MC88110. The basic MC88110/MC88410 system consists of a single MC88110 processor, the MC88410 secondary cache controller, and the MCM62110 secondary cache RAM array as shown in Figure 1-1.

The MC88410 and MCM62110 array are optimized to provide low latency memory access to the MC88110 RISC microprocessor. On the initial access, one wait state is incurred between a valid address and the first data cycle. Subsequent burst accesses do not incur any wait states between data transfers. A processor line fill, for example, takes six cycles: one address cycle, one wait cycle, and four cycles to transfer the data. The large secondary cache provided with the MC88410 and MCM62110 array improves the cache hit rate. In addition, streaming data from the system bus to the processor reduces the secondary cache miss penalty.

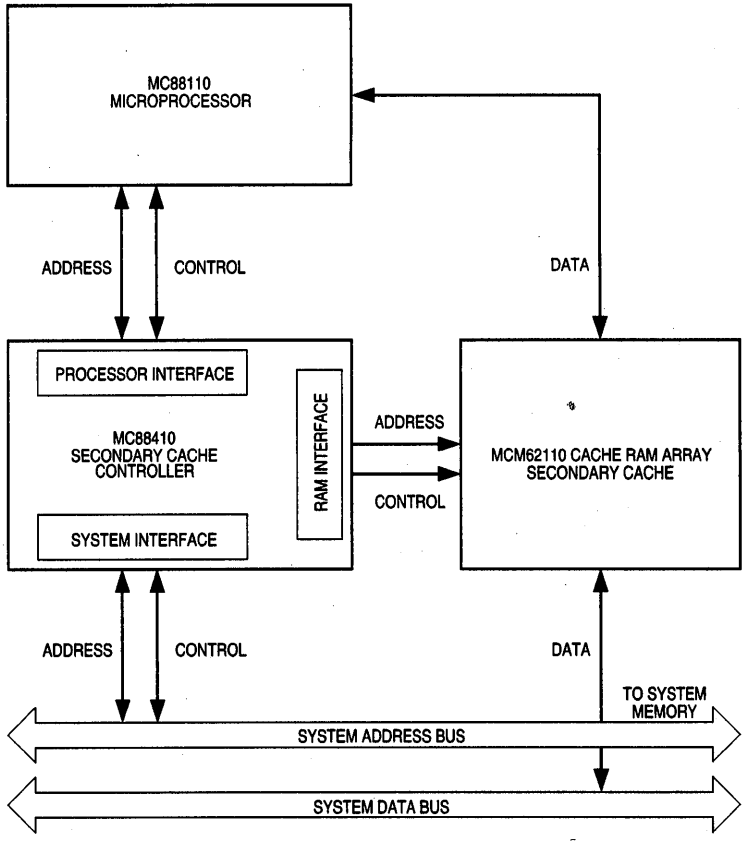


Figure 1-1. MC88110/MC88410 System Configuration

The MC88410 enhances the MC88110 system flexibility by providing a choice of the system bus clock frequencies, order of burst transfers, and secondary cache line sizes. Half-speed mode allows the MC88410, MC88110, and secondary cache to operate at full speed while the system bus operates at half the processor clock frequency. The MC88410 also supports either zero-word-first or critical-word-first order for burst transactions. The MC88410 cache tags can be configured for either a 32-byte or 64-byte secondary cache line length.

The MC88410 reduced system bus bandwidth consumption is important in single processor systems with other peripherals sharing the bus. A system can also use two MC88410s and one MC88110 without degrading cache access time. This configuration doubles the size of the secondary cache (MCM62110 array) as shown in Figure 1-2. The MC88410 chip select signal can be used to divide the address space in half. In addition, two MC88410s allow the processor to access two different system buses as shown in Figure 1-3. For example, one interface can be the local system bus while the other can interface to a back-plane bus. Configurations using two MC88410s must add an external arbiter at the processor interface.

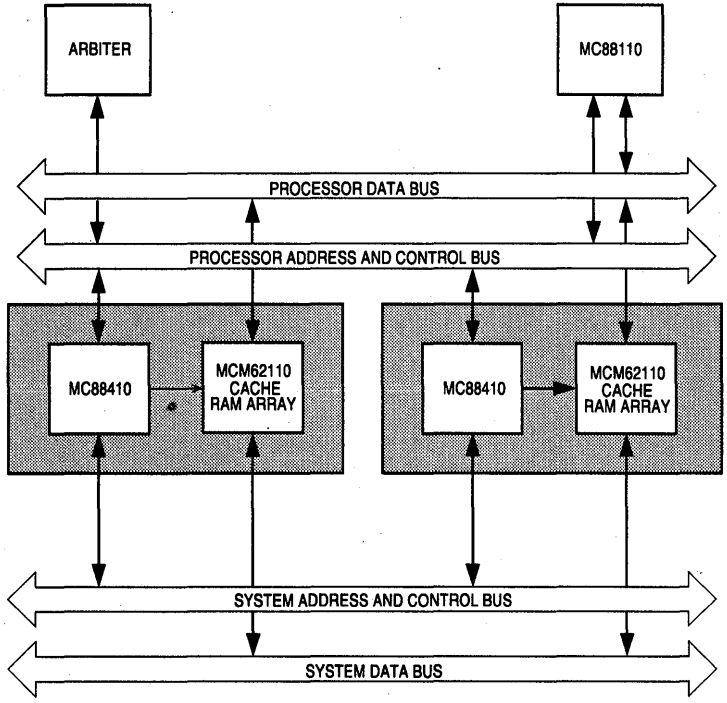


Figure 1-2. System with Two MC88410s

1.3 MC88410 BENEFITS IN MULTIPROCESSOR SYSTEMS

The benefits of using the MC88410 in single processor implementations also apply to multiprocessor systems. In addition, the MC88410 improves the MC88110 multiprocessing capability by significantly reducing system bus bandwidth consumption. The increased available bandwidth, along with the MC88410 hardware enforced cache coherency protocol, enables the implementation of scalable shared-bus multiprocessing systems. Processor nodes can be connected to provide a variety of system configurations. Figure 1-4 illustrates a multiprocessing scheme in which two processors, each with a different level of MC88410 support, share a system interface. The MC88410 extends the MC88110 bus protocol for pipelined bus systems while maintaining cache coherency.

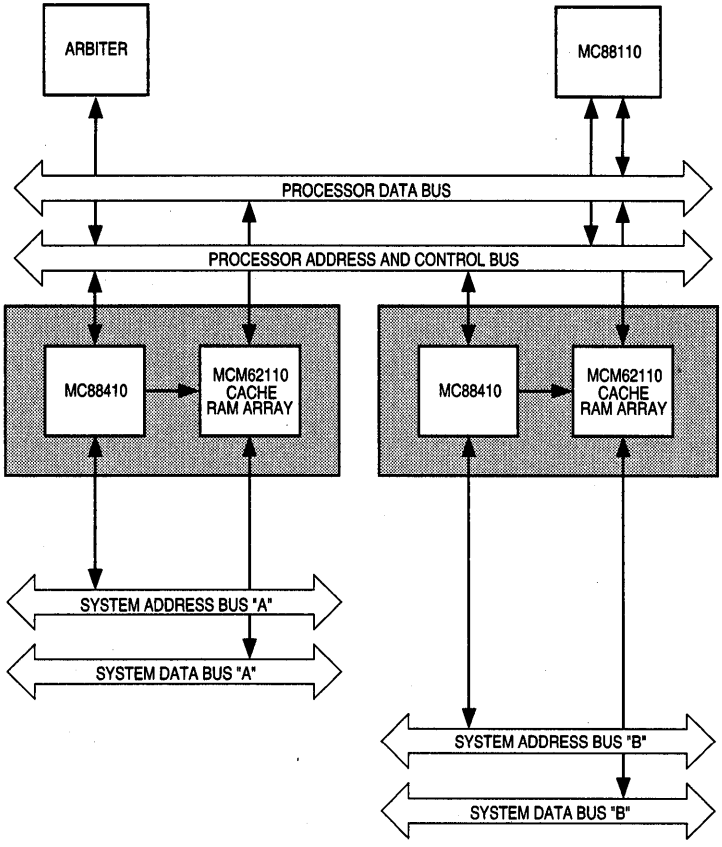


Figure 1-3. Dual-Bus System

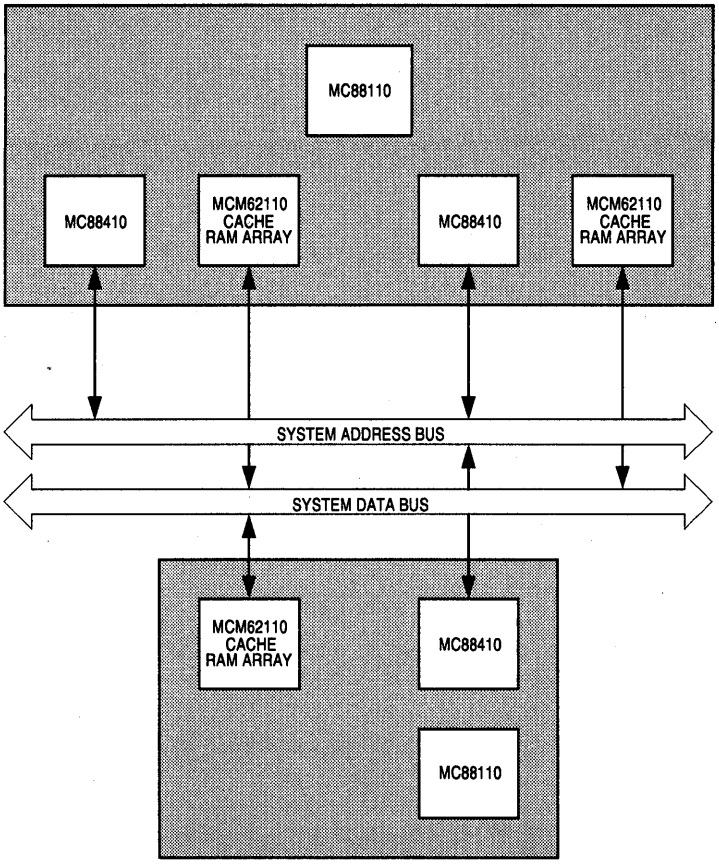


Figure 1-4. Multiprocessor Implementation

1.4 CACHE COHERENCY

Cache coherency is an important consideration in multicache, multiprocessor systems. To attain coherent caches in a system, every processor must have access to the most current data and be notified when its cached copy of data is no longer valid. The MC88110/MC88410 node maintains cache coherency by hardware enforced bus snooping. The MC88110/MC88410 uses a write invalidate with intervention protocol to ensure that only one cache in the system has a modified copy of a given cache line at all times. The protocol allows other caches to have local copies which are all consistent. When an MC88110 writes data to a memory location shared by other caches, the other cache controllers are notified that their copy of the cache line containing that data is stale and must be invalidated. The MC88110/MC88410 snoops bus transactions by monitoring externally initiated bus transactions and comparing all addresses to the internal data cache tags. A "snoop hit" occurs when the cache tag for a valid entry matches the address on the bus.

The MC88410 supports two levels of hardware based cache coherency: vertical and lateral. Vertical coherency refers to data coherency between the primary (MC88110) cache and the secondary (MC88410-controlled) cache. Lateral coherency refers to coherency between the secondary cache and other caches on the system interface.

To maintain vertical coherency, the MC88410 uses a processor tag (PTAG) to dynamically monitor the state of the primary cache. The MC88410 improves processor bus bandwidth by filtering snoop transactions on the system bus so only primary cache "snoop hits" are passed to the processor.

The main tag (MTAG) maintains the status of the secondary cache. The MC88410 uses the MTAG to enforce coherency between its secondary cache, other caches, and main memory. The MTAG and PTAG can be simultaneously accessed by the processor or the system.

1.5 MC88110/MC88410 SYSTEM OVERVIEW

The following paragraphs describe the general operation of the basic MC88110/MC88410 system and the function of each component. The MC88110 RISC microprocessor contains separate on-chip 8-Kbyte instruction and data caches and memory management units (MMU). The MC88110 supports both single-beat and burst data transactions with a choice of memory update policies. The secondary cache using the MCM62110 fast static RAM (FSRAM) provides dual data ports, on-chip parity checking, and output latches, and supports data streaming. By filtering system bus traffic for the microprocessor and managing secondary cache accesses and coherency, the MC88410 significantly improves overall system performance. The MC88410 incorporates three independent interfaces and integrated cache tags to reduce part count and improve performance while allowing flexibility in system definition. The MC88410 is designed specifically to support the MC88110.

1.5.1 MC88110 Microprocessor

The MC88110 is the second implementation of the M88000 family of RISC microprocessors. The MC88110 is a Symmetric Superscalar™ design capable of issuing and retiring two instructions per clock cycle without any special alignment, order, or type restrictions on the instruction stream. Instructions are issued to multiple execution units, execute in parallel, and can complete out of order, with the processor automatically keeping results in the correct program sequence. In a single-chip implementation, the MC88110 integrates the central processing unit, floating-point unit, graphics processing unit, virtual memory address translation, instruction cache, and data cache.

Ten independent execution units communicate with a general register file and an extended register file through multiple, 80-bit, internal buses. Each of the register files has sufficient bandwidth to supply four operands and receive two results per clock cycle. Each of the pipelined execution units, including those that execute floating-point and data movement instructions, can accept a new instruction and retire a previous instruction on

---

Symmetric Superscalar is a trademark of Motorola, Inc.

every clock cycle. The high data and instruction throughput requires low memory latency to maintain peak performance. The addition of the MC88410 and secondary cache RAM array reduces memory latency to provide a high level of data throughput and system performance.

1.5.1.1 MC88110 Instruction and Data Cache

The MC88110 includes separate on-chip data and instruction caches. Each cache provides 8 Kbytes of 2-way set-associative, physically addressed memory. Cache management facilities provide both the instruction and data caches with a cache freezing capability.

The instruction unit attempts to fetch two instructions each clock cycle from the instruction cache. If there is an instruction cache miss, or if the instruction cache is disabled, the instruction cache requests that the bus interface unit (BIU) run an external bus transaction to fetch the needed instructions. The data cache and data unit may request that the BIU run an external bus transaction as a result of a load, store, or exchange instruction, or for cache coherency reasons.

The instruction cache uses physical address tags, so the instruction cache does not need to be flushed on a context switch. The MC88110 instruction cache is configured as 128 sets with two lines per set. Each line contains eight 32-bit words, an address tag, and a valid bit. Instruction cache coherency must be maintained by software and is supported by a fast invalidation capability.

The MC88410 contains hardware to ensure coherency between the MC88110 data cache and the secondary cache. An understanding of the operation of the MC88110 data cache is useful in understanding the operation of the secondary cache.

The MC88110 data cache is also configured as 128 sets with two lines per set. Each line contains eight 32-bit words, an address tag, and three status bits used to enforce cache coherency. When a data cache access begins, the data unit provides the data cache and MMU with the logical address of the desired information. The data MMU translates the logical address to the physical address and provides the cache with information about the type of cache access being performed. The MC88110 data cache provides three software-selectable memory update policies as well as hardware to support cache coherency.

1.5.1.2 MC88110 Memory Update Policy

The MC88110 provides hardware support for three memory update policies: write-back, write-through, and cache-inhibit. The memory update policy used for the secondary cache is determined by the MC88110 MMUs. Each page or block of memory is specified to be in one of these modes within the corresponding page or block descriptor in the data memory management unit. The MC88110 also has a store-through option for the store instruction that allows individual accesses to be performed in the write-through mode, even if the corresponding page or block is designated as operating in the write-back mode.

In the write-back mode, memory is not updated each time a corresponding cache line is modified. In the write-through mode, write operations update memory every time a write occurs. When the access is cache-inhibited, data is never copied into the data cache

of the MC88110 or the secondary cache. Instead, read and write operations access main memory directly.

1.5.1.3 MC88110 Bus Overview

The MC88110 bus interface includes the address bus, data bus, and control and information signals. The address of the instruction or data needed by the processor is driven on the address bus. Similarly, the requested instruction or data is transferred to the processor on the data bus. The bus interface control and information signals include the transfer attribute, arbitration, transfer control, snoop control, processor status, and interrupt signals.

There are two types of bus transactions that transfer data to the secondary cache or system bus: single-beat transactions and burst transactions. During single-beat transactions, a byte, half word, word, or double word is transferred between the processor and the secondary cache or system bus. During burst transactions, eight words are transferred in 4 double-word transfers.

The MC88410 supports all of the MC88110 bus transactions. Two transaction types are not cacheable in the MC88110 but are cacheable in the secondary cache. These transactions are the locked transactions and MMU hardware table searches.

1.5.1.4 MCM62110 FSRAM Secondary Cache

The MC88410 supports a direct-mapped and physically-addressed secondary cache that contains both instructions and data. The secondary cache can be implemented with MCM62110 dual-bus FSRAMs. The MCM62110 array incorporates a 32K x 9-bit static core with two 9-bit I/O ports (8 data, 1 parity). Each I/O port has input registers and output latches. In the simplest configuration, eight MCM62110 arrays are configured in parallel to form a 64-bit (plus parity) data bus. This single bank design eliminates the need for traditional bank switching on burst accesses and reduces the number of devices for typical configurations. The MC88110 data bus and the system data bus connect directly to the MCM62110 dual-bus cache RAM array.

The MCM62110 array has a streaming feature that allows data to be passed, through the RAM, between the processor and system ports in either direction. This streaming is accomplished by latching data in from one port and asynchronously enabling the outputs on the other port. It is also possible to write to the RAM while streaming data through it.

The MC88410 directly drives and controls the MCM62110 secondary cache RAM array. No external logic is needed between the MCM62110 array and the processor bus, system bus, or MC88410.

1.5.2 MC88410 Secondary Cache Controller

The MC88410 controls the secondary cache, enforces cache coherency, and arbitrates processor bus transactions. The MC88410 is not a programmable device and only reacts to input signals from either the processor or the system bus interface.

1.5.2.1 MC88410 Functional Overview

The MC88410 acts upon signals from the processor and system interfaces and controls the secondary cache through the RAM interface. Cache line size, order of burst transfers, system clock speed, and secondary cache size are set by configuration signals at reset. The basic functional blocks in the MC88410 are shown in Figure 1-5. The MC88410 contains three functional units: tag unit, decode unit, and execution unit. The tag unit contains the main tag and processor tag and maintains the status of the secondary cache and primary (MC88110) data cache. The decode unit contains separate decoders for the processor and system address buses. The decoders evaluate each request from the processor or system bus and its associated lookup status from the tags and decide what actions need to occur to satisfy the request. The MC88410 also contains a flush mechanism to flush or invalidate pages of the secondary cache or the entire secondary cache. The execution unit drives transactions to the processor and system buses and controls the secondary cache.

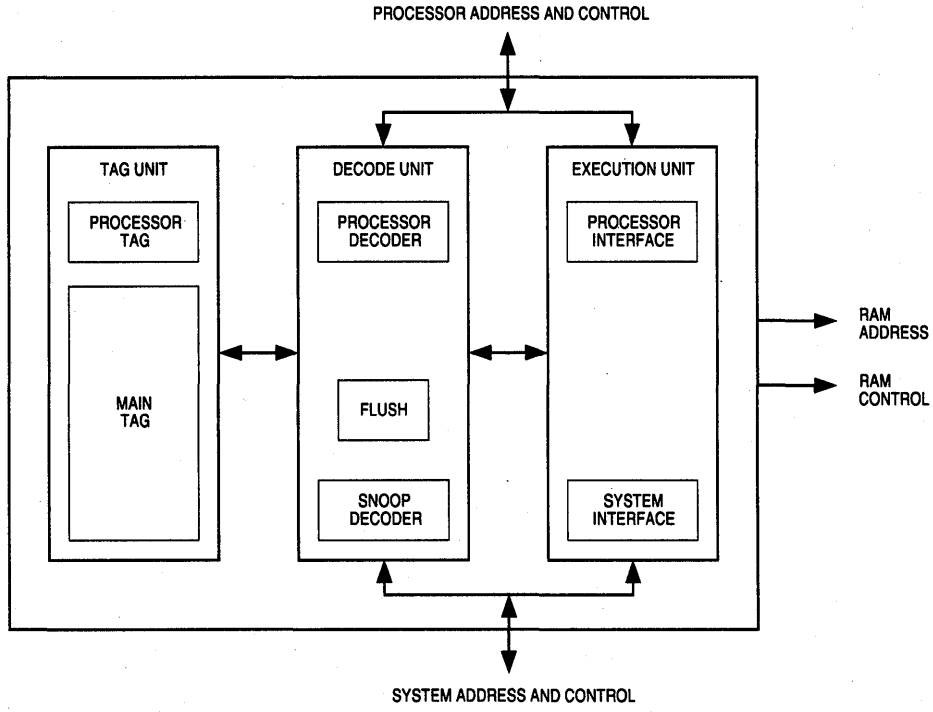


Figure 1-5. MC88410 Functional Block Diagram

A transaction begins when a valid address is latched and looked up in the main tag and processor tag and then the status is sent to the decode unit. The decode unit uses the tag status and the attribute signals from the bus to decode the appropriate transaction. The execution unit arbitrates for the processor and system buses, controls the secondary cache, and executes the transactions. For example, a cache hit from a snoop request is decoded

by the snoop decoder and causes the system interface to arbitrate for the system bus and execute the appropriate transaction. The efficient arbitration of internal resources allows the MC88410 to always respond to a system snoop request in two clock cycles.

1.5.2.2 MC88410 Bus Overview

Figure 1-1 shows the MC88410 bus interfaces. The MC88410 processor interface connects the MC88410 to the processor address bus and control signals. The system interface connects the MC88410 to the system address bus and control signals. The MC88410 has a RAM interface that drives the address and control signals directly to the MCM62110 secondary cache RAM array. The MC88410 has a triple-interface architecture to provide concurrent access to the system bus, processor bus, and RAM cache.

When the microprocessor begins a transaction on the processor bus, the MC88410 decodes the tag address and transaction attributes, performs the appropriate transaction, and updates the cache tags. If necessary, the MC88410 arbitrates for control of the processor or system bus. The MC88410 constantly "snoops" the address of system bus transactions to ensure data coherency between the primary cache, secondary cache, and main memory. If a transaction is required to maintain coherency, the MC88410 arbitrates for the processor and system buses and executes the appropriate transaction. During a data transaction, the MC88410 drives the address to the secondary cache and controls the data output of the cache. Snoop transactions that only hit in the secondary cache are handled by the MC88410 and an invalidation broadcast is not issued to the processor.

1.5.2.3 MC88410 Cache Tags

The processor tag is checked for the contents of the primary (MC88110) data cache. The processor tag is a copy of the 128-set, two-way set-associative MC88110 data cache address tag with an inclusion bit instead of the three status bits. The inclusion bit in the processor tag is set if the primary data cache has a valid copy of the data as shown in Figure 1-6.

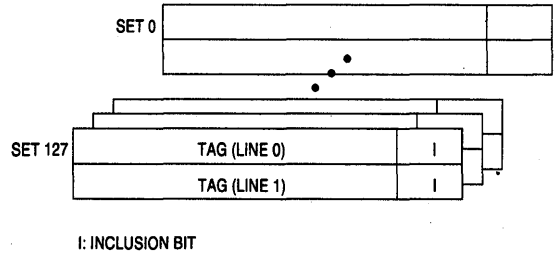


Figure 1-6. Processor Tag Organization

The main tag provides up to 16K entries of secondary cache tags for secondary cache lines of either 32 bytes or 64 bytes. Figure 1-7 shows the organization of the secondary cache for a 32-byte line size. The main tag provides hit or miss status for all processor and system transactions. The main tag pointer contains the upper 14 to 12 bits of the address, depending on the cache organization determined at reset. The main tag also includes

three status bits used to enforce cache coherency: the shared, modified, and valid bits. The MC88410 implements an inclusion policy which ensures that all PTAG entries are also MTAG entries. Figure 1-8 shows the organization of the main tag.

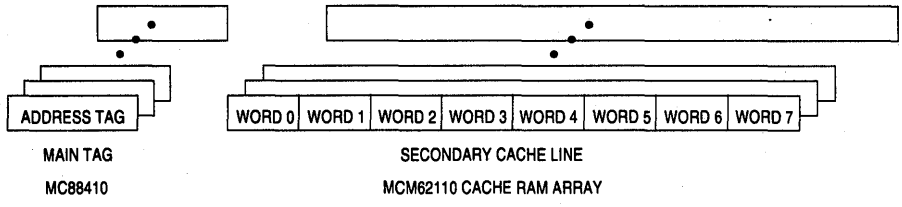


Figure 1-7. Secondary Cache Organization

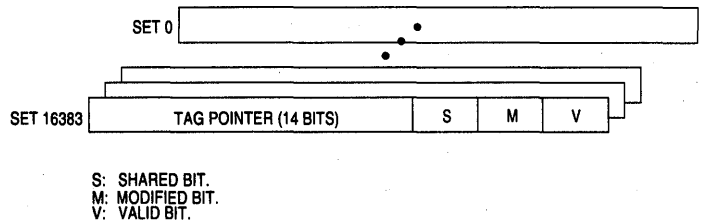


Figure 1-8. Main Tag Organization

1.5.2.4 MC88410 Address Decode

The MC88410 interprets an incoming 32-bit address differently depending on cache size and cache line size. In all cases the address consists of four fields: tag, tag index, word offset, and byte offset. The tag index points to the main tag entry used for comparison with the address. The word offset then selects a double word in the selected cache line. Finally, the byte offset indexes into each double word (eight bytes) to select a specific byte.

For example, Figure 1-9 conceptually illustrates the address decoding of a processor or system bus address for a 256-Kbyte cache with a 32-byte line size. For this configuration, bits 31 to 18 of the address are compared to the value stored in the MTAG to determine hit or miss status. Bits 17 to 5 of the address are decoded into one of the 8-Kbyte MTAG entries and are also passed to the secondary cache for its address decode. Bits 4 and 3 identify the requested double word within the selected secondary cache line. Finally, address bits 2 to 0 are decoded to select the byte. To select the PTAG entry for comparison, bits 11 to 5 are always decoded into one of the 128 sets of PTAG entries. An address is decoded and looked up in both the PTAG and MTAG simultaneously.

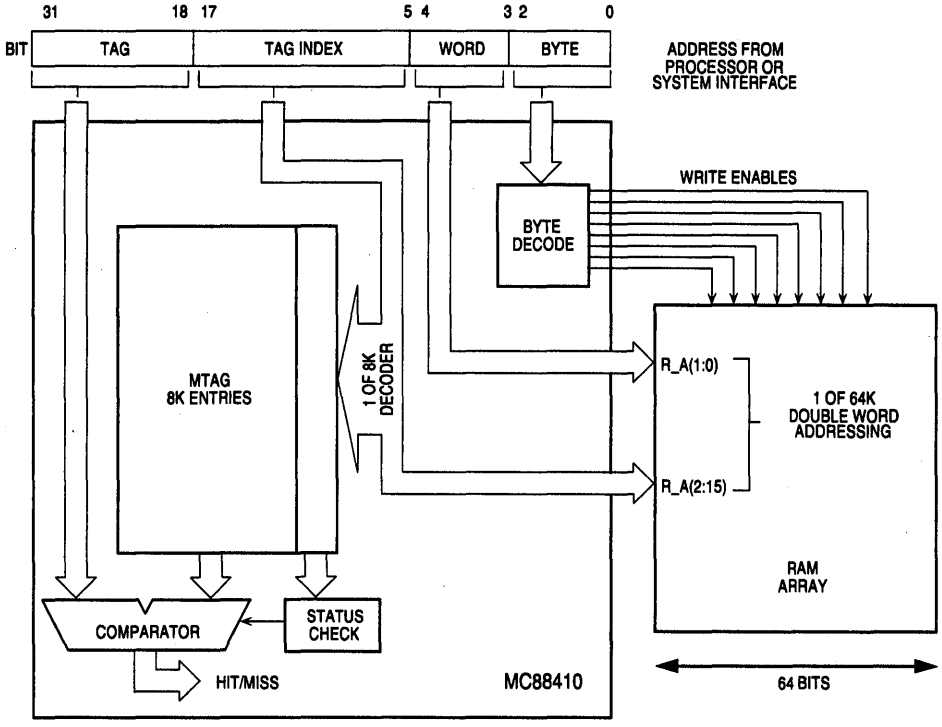


Figure 1-9. Address Decode: 256-Kbyte Cache with 32-Byte Line Size

SECTION 2
SECONDARY CACHE OPERATION

The MC88410 supports a direct mapped and physically addressed secondary cache that contains both instructions and data. The secondary cache can be configured to support 1/4 Mbyte or 1 Mbyte of cache RAM with line sizes of 32 or 64 bytes each. Note that support for a 1-Mbyte cache is dependent on migration of the MCM62110 to the 128K x 9 density. The secondary cache can also be configured to support either zero-word-first or critical-word-first burst ordering on the system bus. The data streaming protocol allows data to be forwarded to the primary cache as it is being written to the secondary cache.

This section describes the cache organization, possible line states, memory update policies, processor cache accesses, flush and invalidate operation, cache coherency, and bus snooping examples for the MC88410. Refer to **Section 4 Processor Bus Interface** and **Section 5 System Bus Interface** for detailed timing information.

NOTE

The MC88410/MCM62110 secondary cache contains both data and instruction data types. The term **data** is used in this manual to refer to both data and instructions unless specifically noted otherwise.

2.1 CACHE ORGANIZATION

The MC88410 may be configured to support a direct mapped cache in one of three configurations: 1/4-Mbyte cache with 32-byte line size, 1/4-Mbyte cache with 64-byte line size, and 1-Mbyte cache with 64-byte line size. Note that the line size applies only to the system bus interface. The line size for the processor interface is fixed at the line size for the MC88110, which is 32 bytes. The cache size and the cache line size are configured at reset. For more information about configuring the MC88410, see **Section 5 System Bus Interface.**

The MC88410 has two sets of cache tags: the main tag (MTAG) and the processor tag (PTAG). The MTAG is used to determine whether there is a secondary cache hit, and contains the three status bits and the high order bits of the address of each line of the secondary cache. The MTAG directly maps up to 16 Kbytes of secondary cache tags for secondary cache lines of either 32 or 64 bytes. The MTAG organization varies with the organization of the secondary cache, which is configured at reset.

The PTAG is used to determine whether a cache line is included in the primary (MC88110) data cache and it is similar to the 128-set, two-way set-associative MC88110

data cache address tag with an inclusion bit instead of the three status bits. The inclusion bit in the PTAG is set if the primary data cache has a valid copy of the data (see Figure 2-1). For each entry in the PTAG with the inclusion bit set, there is a corresponding entry in the MTAG. The status bits of the corresponding entry in the MTAG match the status bits of the line in the primary data cache.

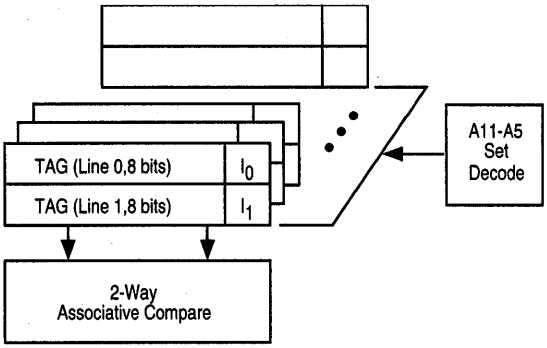


Figure 2-1. PTAG Organization

The MC88410 interprets an incoming 32-bit address differently depending on cache size and cache line size. In all cases the address consists of four fields: the address tag, the tag index, the word offset, and the byte offset. The tag index points to the main tag entry used for comparison with the address tag. If the upper bits of the address match the address tag, then it is a cache hit. The word offset then selects a double word in the selected cache line. Finally, the byte offset indexes into each double word (eight bytes) to select a specific byte.

The following paragraphs describe the address decoding, the MTAG organization, and cache organization for each of the possible configurations.

2.1.1 1/4-Mbyte Configurations

For the 1/4-Mbyte configurations, the external cache of the MC88410 is built from a single bank of eight MCM62110 fast static RAM (FSRAM) devices. No external logic is needed between the MC88410 and the MCM62110 array. Likewise, no external logic is needed between the MCM62110 array and the two data buses (processor and system). The arrangement of the MCM62110 array for the 1/4-Mbyte configurations is shown in Figure 2-2.

For the 1/4-Mbyte cache size, the lower 18 bits of the address are needed to index into the cache, and the upper 14 bits are used for the address tag. The division of the address space between the tag index and the word offset depends on the line size. The following paragraphs describe the address decoding and the MTAG organization for the 32-byte and 64-byte line size configurations with the 1/4-Mbyte secondary cache size.

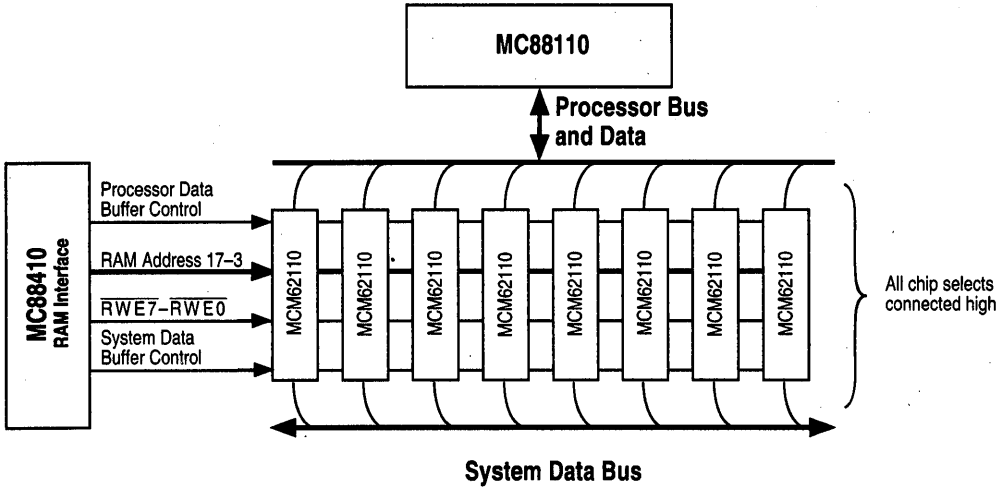


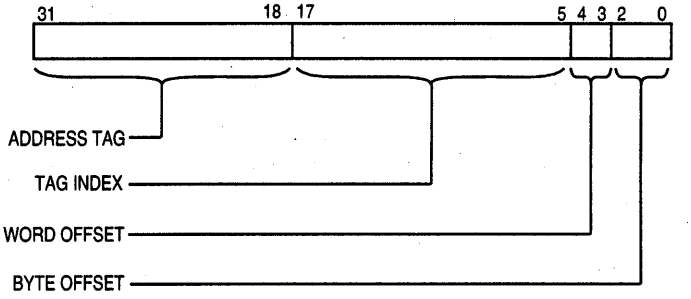
Figure 2-2. 1/4–Mbyte Hardware Configuration

2.1.1.1 1/4 Mbyte with 32-Byte Line Size Configuration

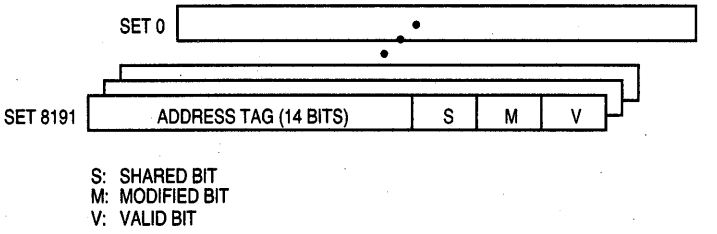
When the secondary cache is configured to be 1/4-Mbyte in size with a 32-byte line size, the address is decoded and the MTAG is configured as shown in Figure 2-3. In this case, bits 17–5 of the address are used to select one of 8K MTAG entries. The 14-bit address tag of this MTAG entry is then compared to bits 31–18 of the address to determine if there is a secondary cache hit. If a secondary cache access is necessary, then bits 17–0 of the address are used by the MC88410 to access the appropriate data in the MCM62110 array.

2.1.1.2 1/4 Mbyte with 64-Byte Line Size Configuration

When the secondary cache is configured to be 1/4-Mbyte in size with a 64-byte line size, the address is decoded and the MTAG is configured as shown in Figure 2-4. In this case, bits 17–6 of the address are used to select one of 4K MTAG entries. The 14-bit address tag of this MTAG entry is then compared to bits 31–18 of the address to determine if there is a secondary cache hit. If a secondary cache access is necessary, bits 17–0 of the address are used by the MC88410 to access the appropriate data in the MCM62110 array.



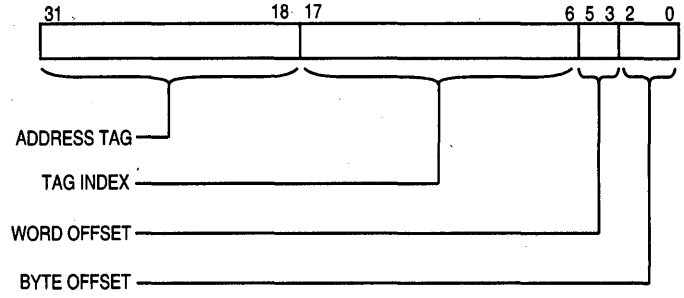
ADDRESS DECODE



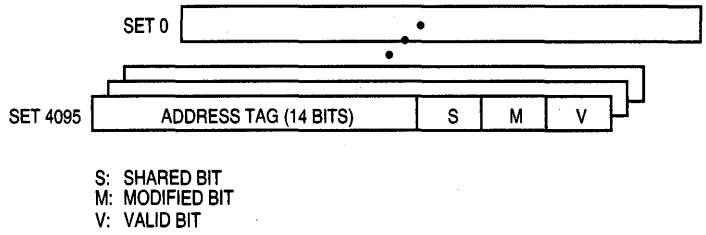
S: SHARED BIT
M: MODIFIED BIT
V: VALID BIT

MTAG ORGANIZATION

Figure 2-3. 1/4–Mbyte Cache and 32–Byte Line



ADDRESS DECODE



S: SHARED BIT
M: MODIFIED BIT
V: VALID BIT

MTAG ORGANIZATION

Figure 2-4. 1/4–Mbyte Cache and 64–Byte Line

2.1.2 1 Mbyte with 64-Byte Line Size Configuration

A single MC88410 is able to control secondary cache sizes of up to 1 Mbyte; however, this is dependent on the future migration of the MCM62110 to the 1-Mbit density. Figure 2-5 shows the 1-Mbyte configuration using 1-Mbit SRAMs.

When the secondary cache is configured to be 1 Mbyte in size, the line size must be specified as 64 bytes. The lower 20 bits of the address are needed to index into the cache, and the upper 12 bits are used for the address tag. In this case, the address is decoded and the MTAG is configured as shown in Figure 2-6. Bits 19–6 of the address are used to select one of 16K MTAG entries. The 12-bit address tag of this MTAG entry is then compared to bits 31–20 of the address to determine if there is a secondary cache hit. If a secondary cache access is necessary, then bits 19–0 of the address are used by the MC88410 to access the appropriate data in the 1-Mbit cache array.

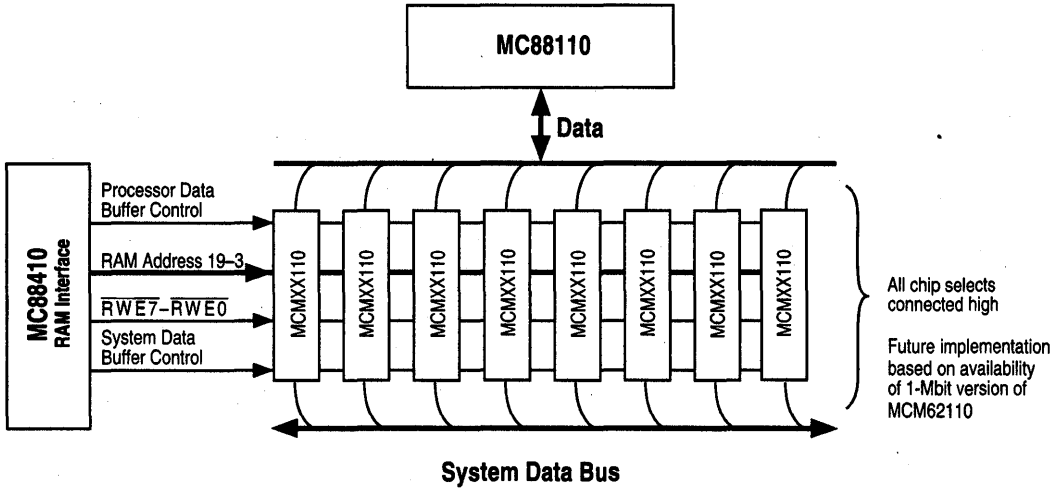
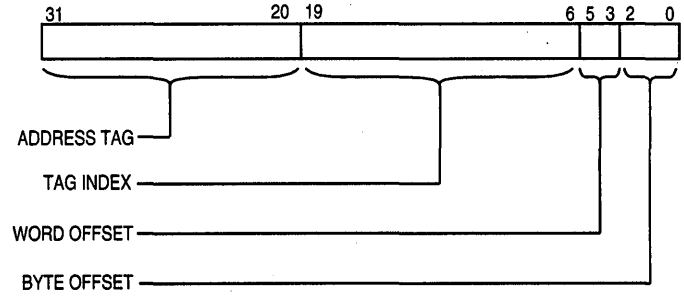
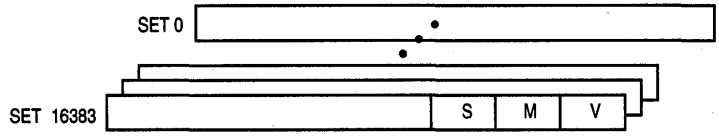


Figure 2-5. Future 1–Mbyte SRAM Cache



ADDRESS DECODE



S: SHARED BIT
M: MODIFIED BIT
V: VALID BIT

MTAG ORGANIZATION

Figure 2-6. 1/2–Mbyte Cache and 64–Byte Line

2.2 SECONDARY CACHE LINE STATES

When the MC88110 initiates an access, the actions taken by the MC88410 depend on whether the access is cacheable. If the access is cacheable, the actions taken by the secondary cache depend on the state of the cache line.

Each cache line can be in one of four states at any one time (shared, shared-unmodified, exclusive-modified, and exclusive-unmodified). These states reflect the status of the line with respect to memory and whether the processor node (MC88410/MC88110 or MC88110) has exclusive ownership of the cached data. The state of each cache line is indicated by the three-state bits in that line: the first bit indicates whether a line is valid or invalid, the second bit indicates whether the line is shared or exclusive to the processor node, and the third bit indicates whether the line is modified or unmodified with respect to memory. Table 2-1 describes the four possible cache states.

Table 2-1. Cache Line States

State	Description
Invalid	The information in this line is no longer valid and should not be used. A line is marked invalid as a result of four conditions: the entire cache or a specific line in the cache is invalidated, the bus snooping logic marks the line as invalid, a bus error occurs during a cache line read access, or a cache hit occurs for a cache-inhibited access.
Shared-unmodified	The data in this line is shared among processors (or processor nodes), so other caches may have a copy of this line. However, this line is unmodified with respect to memory.
Exclusive-modified	Only one processor node (this processor node) has a copy of the data in this line in its cache(s), and the line has been modified with respect to memory. Note that if any word in the line is modified, then the entire line is marked as modified. If the line is marked exclusive-modified in the MTAG, the secondary cache does not necessarily contain the most updated data, it may be contained by the primary cache. Note that cache lines containing instructions are never marked exclusive-modified.
Exclusive-unmodified	Only one processor node (this processor node) has a copy of this line in its internal cache, and the line is unmodified with respect to memory.

NOTE

Throughout this section, the following nomenclature is used: when a cache line is referenced as modified, it is exclusive-modified (no shared-modified state exists within the MC88110 or MC88410 caches). When a cache line is referenced as exclusive, it can be assumed that it is not relevant to that context whether it is exclusive-modified or exclusive-unmodified.

During a data access, the secondary cache line that contains the data being read or written may change state. The state of the cache line after the access depends on the previous state of the line, the type of access, and whether the access resulted in a hit or a miss in the secondary cache.

2.3 MEMORY UPDATE POLICIES

Transactions from the MC88110 follow one of the three memory update policies: write-back, write-through, and cache-inhibited. The MC88410 determines the memory update policy in effect for each transaction from the $\overline{\text{P\_CI}}$ and the $\overline{\text{P\_WT}}$ signals of the processor interface (see Table 2-2).

Table 2-2. Memory Update Policy Encoding

$\overline{\text{P\_CI}}$	$\overline{\text{P\_WT}}$	Memory Update Policy
Asserted	Don't care	Cache-inhibited
Negated	Negated	Write-back
Negated	Asserted	Write-through

In the write-back mode, memory is not updated each time a corresponding cache line is modified. In the write-through mode, write operations update memory every time a write occurs. When the access is cache-inhibited, data is never stored in the primary or secondary cache, but read and write operations access main memory directly. Note that all three modes of operation have specific advantages and disadvantages; therefore, the choice of which mode to use depends on the system environment as well as the application.

The distinction between the write-back and write-through modes affects only single-beat write transactions from the processor. Single-beat write transactions that follow the write-through policy always update memory as well as the secondary cache on cache hits. If there is a cache miss on a write-through access, only memory is updated, and there is no secondary cache line fill. In the write-through mode, memory is always updated during write operations, and global transactions cause other snooping bus masters to invalidate or copy back their cached images of the memory being updated.

Single-beat write transactions that follow the write-back policy do not necessarily cause a system bus transaction to update memory. Instead, memory updates occur only when a modified line is to be replaced due to a cache miss or when another bus master attempts to access a specific address for which the corresponding cache entry has been modified. A single-beat processor write that hits a shared-unmodified line under the write-back policy causes the MC88410 to perform a system invalidate transaction, update the secondary cache, and mark the line exclusive-modified. Single-beat write hits to exclusive lines do not cause a system bus transaction under the write-back policy. Write transactions that miss under the write-back policy cause the MC88410 to allocate a secondary cache line (see **2.7 Secondary Cache Line Allocation**).

If a memory location is designated as cache-inhibited, information from this location is never stored in either the primary or the secondary cache. Cache-inhibited accesses that miss in the MC88410 perform the necessary transaction across the system and processor interfaces with no effect on the secondary cache contents. Cache-inhibited accesses that hit in the secondary cache cause the hit line to be invalidated. If the access hits a modified

line, the MC88410 copies the appropriate data back to main memory before invalidating the secondary cache line.

2.4 CACHE COHERENCY

The MC88410 supports two levels of hardware based coherency protocol: vertical and lateral. Vertical coherency refers to data coherency between the primary cache and the secondary cache. To maintain this type of coherency, the contents of the primary cache must be a subset of the secondary cache. Coherency can only be maintained if one and only one MC88110 resides on the processor interface, but the MC88110 data cache may use either write-back or write-through policies. Lateral coherency refers to coherency between the MC88410 cache and other caches on the system interface (i.e. other MC88110s or MC88110/MC88410 nodes). The secondary cache supports both write-back and write-through policies as directed by the $\overline{\text{P\_WT}}$ signal.

2.4.1 Vertical Coherency

The MC88410 maintains vertical coherency with the use of the PTAG. The PTAG has the same associativity and size as the primary data cache tag and dynamically keeps track of which primary data cache lines are valid (see **2.1 Cache Organization**). Vertical coherency must be taken into consideration for all snoop transactions on the system interface and for all secondary cache line invalidate or copyback transactions. For each of these transactions, the PTAG hit/miss status and the MTAG status for that line determine the actions of the MC88410.

The MC88410 monitors and filters snoops at the system interface so that only snoop hits are passed to the processor. All global addresses snooped on the system interface are passed to both the MTAG and PTAG. The PTAG and MTAG lookups occur independent of and parallel to each other. If the address hits in the PTAG and the snoop is to an exclusive-modified line (as determined by the MTAG status) or if the snoop is marked intent-to-modify, the MC88410 knows that the processor has a valid cached copy of the addressed data that must be flushed and invalidated. The MC88410 causes this invalidation by initiating a primary cache invalidate transaction.

In addition to system snoops, the PTAG also enforces vertical coherency for secondary cache line invalidation due to a flush or replacement copyback transaction. All addresses being copied back from the secondary cache are checked against the PTAG. If there is a PTAG hit, then a primary cache invalidate transaction must precede the secondary cache copyback to ensure that the most recent copy of the data is written to memory.

The MC88110 includes a feature that flushes and invalidates pages or all of the primary data cache. Flushing and invalidating the primary cache without flushing the secondary cache could leave some of the inclusion bits improperly asserted. This may degrade performance but coherency is maintained. Invalidating the secondary cache without flushing the primary cache can cause data incoherency.

There are three types of transactions that may cause a PTAG state transition: processor accesses, system snoops, and MTAG replacements. For each case, the MC88410

determines if there is a PTAG hit or miss. Figure 2-7 shows the state transitions for the PTAG lines. In this diagram, the terms hit and miss refer only to PTAG hit/miss, and do not reflect MTAG status. Also, only data (not instruction) transactions apply. All events on the diagram are labeled with both type (processor, snoop, or MTAG invalidate) and status (PTAG hit or miss).

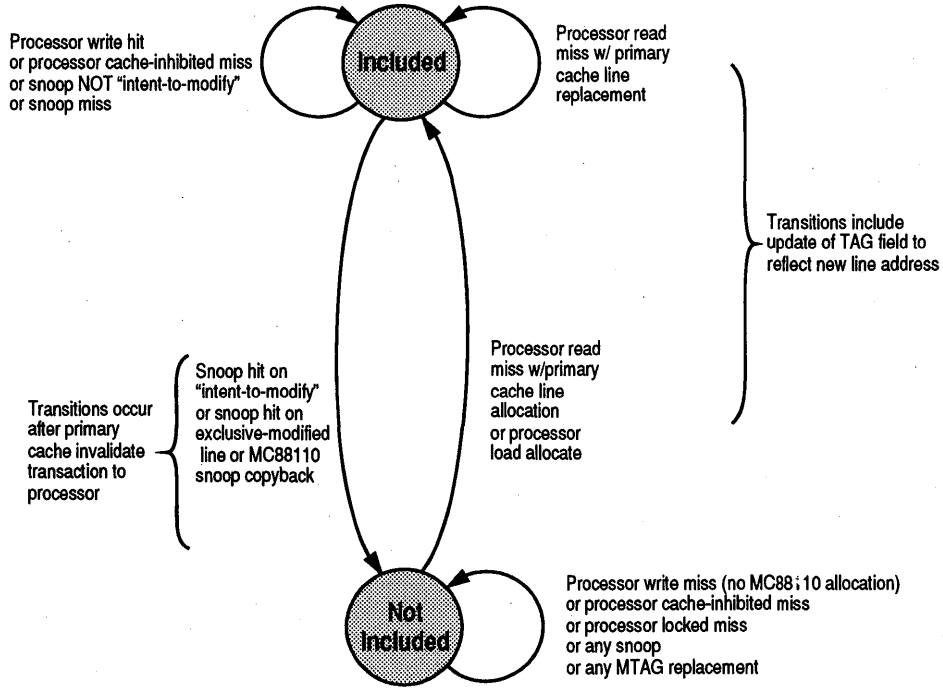


Figure 2-7. PTAG State Transitions

2.4.2 Lateral Coherency

The MC88410 uses the MTAG and system interface snooping to maintain data coherency between secondary caches and main memory according to the lateral coherency state machine. The MC88410 snoops all global traffic on the system address interface. Alternate masters on the system interface use a wait-retry protocol to alert the interface master of potential coherency hazards.

The MC88410 cache state logic is implemented as a four-state design, but also supports a three-state model. The three-state model includes all of the states except the exclusive-unmodified state. When operating in the three-state model, all internal cache state transitions are visible on the external signals of the MC88410. In the four-state model, the transition from the exclusive-unmodified state to the exclusive-modified state for a write hit is not visible on the bus.

The distinction of whether the three- or four-state model is in use is determined by the status of the two shared input signals on the system bus interface ($\overline{\text{SHD}}$ and $\overline{\text{TSHD}}$). Other snooping MC88410s on the bus should drive the $\overline{\text{SHD}}$ signal with their snoop hit status output ($\overline{\text{S\_SSTAT0}}$). Systems with distant snoopers can assert $\overline{\text{TSHD}}$ as late as the first $\overline{\text{S\_TA}}$. For more information about the timing for the $\overline{\text{SHD}}$ and $\overline{\text{TSHD}}$ signals, refer to **Section 5 System Bus Interface**. Systems implementing a three-state cache model simply keep the $\overline{\text{SHD}}$ signal asserted and force all line fills to be marked as shared-unmodified. Note that during line fills for write misses in write-back mode, the $\overline{\text{SHD}}$ signal is ignored (i.e., write miss line fills are always marked as exclusive-modified).

State transition diagrams for the data cache in the four-state model are shown in Figures 2-8 and 2-9 and described in the following paragraphs. Figure 2-8 shows the state transition diagram for the cache operating in write-back mode, and Figure 2-9 shows the state transition diagram for the cache operating in write-through mode. State transitions for the cache in the three-state model are shown in Figure 2-10. All other operations that are not explicitly shown in these diagrams do not affect the cache state.

In the following diagrams, state transitions labeled as "shared" (for example, shared read miss) imply that the $\overline{\text{SHD}}$ or $\overline{\text{TSHD}}$ input signals to the MC88410 are asserted at the appropriate time during the line fill operation. Transitions labeled as "exclusive" imply that the $\overline{\text{SHD}}$ and $\overline{\text{TSHD}}$ input signals are negated during the line fill.

Figure 2-8 shows all state transitions possible for the secondary cache in write-back mode for the four-state model. A line can change state due to a cache miss. Replacing a cache line with a line from main memory is referred to as replacement. For any initial state, an exclusive read miss with replacement changes the line state to exclusive-unmodified, a shared read miss with replacement changes the line state to shared-unmodified, and a write miss with replacement or a read miss with intent-to-modify changes the line state to exclusive-modified. In a multiprocessor system a snoop hit on a read changes the line state of the snooping processor node to shared-unmodified (after a copyback of the data, if modified). A snoop hit on a write- or read-with-intent-to-modify changes line state of the snooping processor to invalid (after a copyback of the data, if modified). When the MC88110 performs a processor invalidate transaction to an unmodified line, the line state

changes to exclusive-modified. If an exclusive-modified line is flushed, the line state changes to unmodified. Finally, if there is a locked read hit to a shared-unmodified line, the line state changes to exclusive-unmodified (after a system invalidate transaction).

Write operations in write-through mode leave the cache state unaffected. Figure 2-9 shows all state transitions possible for the secondary cache when in write-through mode. The exclusive-unmodified state cannot be reached in write-through mode. If a cache line is already in either of the exclusive states when write-through mode is selected, the line does not change state while in write-through mode. This does not cause coherency problems, but if the mode is changed back to write-back, some data may be copied back to memory which is already consistent with the line in the data cache.

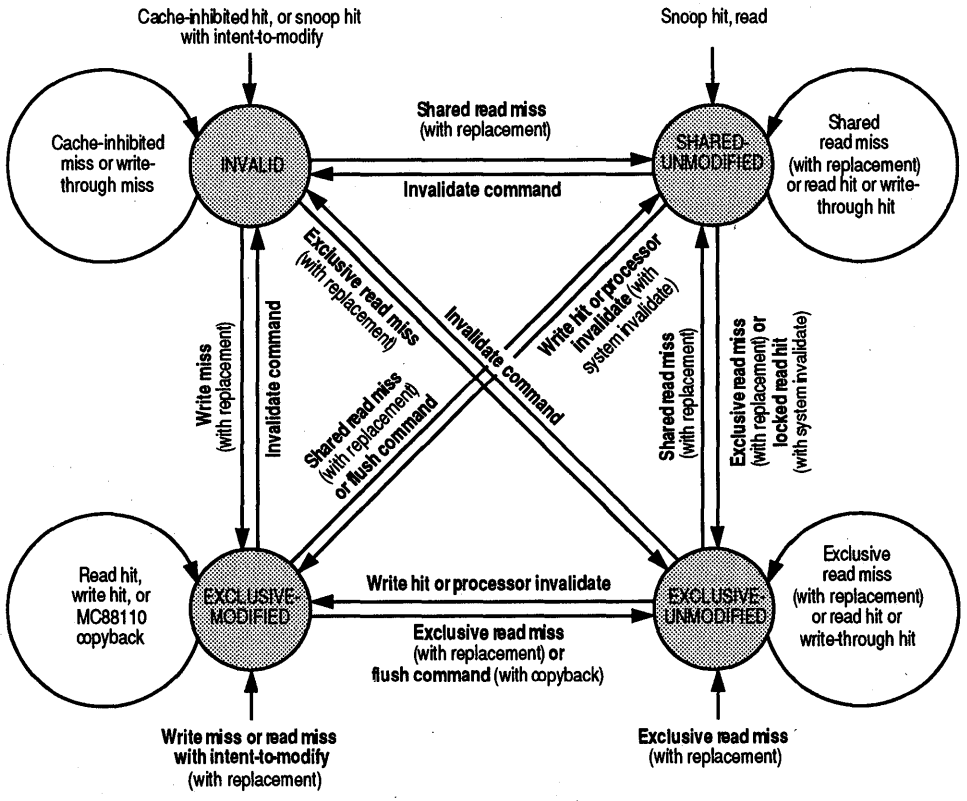


Figure 2-8. Secondary Cache States in Write-Back Mode

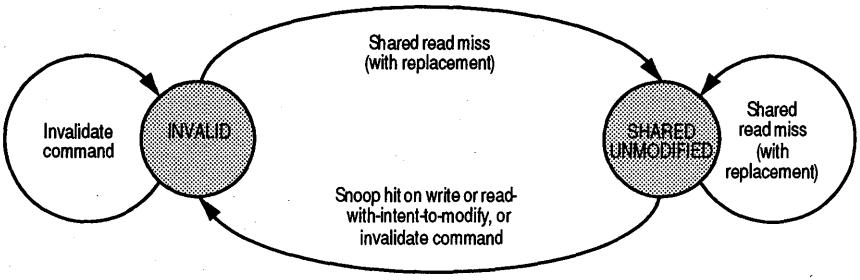


Figure 2-9. Secondary Cache States in Write-Through Mode

Figure 2-10 shows all possible state transitions for the secondary cache in the three-state model. The three-state model does not include the exclusive-unmodified state.

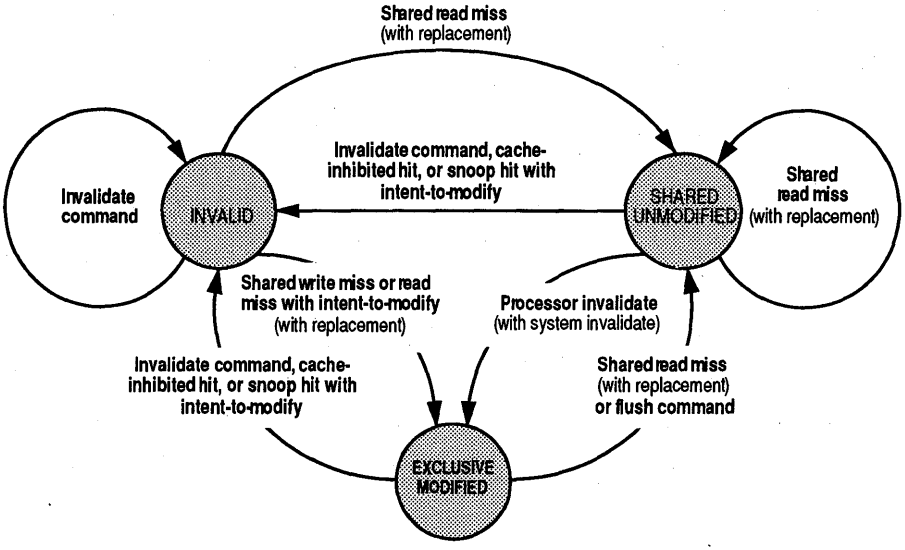


Figure 2-10. Secondary Cache States with Three-State Model

For the three-state model, any read miss line fill that is not intent-to-modify puts the line in the shared-unmodified state. Any line fill that is intent-to-modify puts the line in the exclusive-modified state. When the MC88110 performs a processor invalidate transaction to a shared-unmodified line, the line state changes to exclusive-modified. If a snooping system bus master (MC88410 or MC88110) performs a snoop copyback transaction, then the cache line of the snooping bus master changes state to either shared-unmodified (for a read) or invalid (for a write- or a read-with-intent-to-modify) depending on what caused the snoop copyback.

There are potential benefits to both the three-state and four-state models. The three-state model is useful because all internal state transitions are visible on the system bus. (Note that the primary cache in an MC88110/MC88410 system always uses the three-state model because the $\overline{\text{SHD}}$ input to the MC88110 is grounded.) However, the three-state implementation can cause lower performance than the four-state implementation. In the three-state implementation, the exclusive-unmodified state does not exist; therefore, all data is read in as shared-unmodified. A write hit to shared-unmodified data causes the snooping bus master to perform an system invalidate transaction on the system bus. If the data had been read as exclusive-unmodified (as in the four-state model), then a write hit would simply change the state of the data to be exclusive-modified, and no system bus traffic would occur.

2.5 TRANSACTION OVERVIEW

The transfer of data from the secondary cache or the external memory system to the processor is defined as a processor read transaction. The transfer of data from the processor to the secondary cache and/or to the external memory system is defined as a processor write transaction. The transfer of data on the system bus is defined as a system bus transaction. Invalidate transactions are processor or system bus transactions used to maintain cache coherency. System bus invalidate transactions are not used for data transfer. Note that system bus transactions result from processor transactions, system bus snoop transactions, or secondary cache flushing. The actions of the MC88410 for any transaction depend on whether the access is cacheable and the state of the secondary cache line. Table 2-3 lists the possible single-beat transactions from the MC88110.

Table 2-3. Single-Beat Processor Transaction Types

Transaction	Description
Single-beat read	During single-beat read transactions, the MC88110 reads a byte, half word, word, or double word from the MCM62110 array.
Single-beat write	During single-beat write transactions, the MC88110 writes a byte, half word, word, or double word to the MCM62110 array.
Locked read	A locked transaction uses an indivisible single-beat read/write transaction to exchange the contents of a general register in the processor with that of an addressed memory location. Unlike the MC88110, locked transactions are cacheable in the secondary cache.
Locked write	A locked transaction uses an indivisible single-beat read/write transaction to exchange the contents of a general register with that of an addressed memory location. Unlike the MC88110, locked transactions are cacheable in the secondary cache.
Table search	A table search operation is a series of single-beat transactions performed by the MC88110 when a logical address misses in its address translation caches. Table search transactions are cacheable in the secondary cache.
Write-through	A write-through transaction causes processor store instruction to write through the secondary cache and directly to memory.
Allocate load	The allocate load option is a primary data cache control feature that allows the user to allocate a line in the data cache without filling the entire primary cache line.

Table 2-4 lists the possible burst transactions that are performed by the MC88110.

Table 2-4. Burst Processor Transaction Types

Transaction	Description
Burst Read Transactions	
Read miss line fill	A processor read access that misses in a primary cache causes a processor bus transaction to occur in which an entire line of data is read from the MCM62110 array and written to a primary cache. This operation is called a cache line fill operation. A cache miss occurs when caching is enabled and the instruction/data required by the processor is not resident in the appropriate cache.
Data cache read-with intent-to-modify	A read-with-intent-to-modify transaction is caused by a write access that misses in the primary data cache in write-back mode. A read-with-intent-to-modify transaction operates like a burst read transaction for a primary cache line fill but has the side effect of broadcasting to the MC88410 that the cache line being read will be modified.
Touch load	The "touch load" option is a primary cache control feature that allows data to be loaded into the data cache under user program control.
Burst Write Transactions	
Replacement copyback	When a data cache miss occurs and the corresponding primary cache set has two valid entries, the cache access algorithm selects one of the two lines in the corresponding cache set for replacement. The MC88110 checks the state of the line to be replaced, and if the line is modified, the line is copied back to the MCM62110 array. This operation is called a replacement copyback.
Snoop copyback	When the MC88110 has a cache hit during a primary cache invalidate transaction, it determines if the cache line is modified. If the line is modified, the line must be copied back to the secondary cache before the system bus transaction can complete. This operation is called a snoop copyback.
Flush copyback	The MC88110 has a primary cache control feature that causes either all modified lines or any individual modified line in the primary data cache to be transferred out of the secondary cache, and causes the transferred line(s) to be marked as "unmodified". Each line is transferred by a burst write transaction called a flush copyback.
Flush load	The "flush load" is a primary cache control feature that allows the user to force a modified cache line to be written to the secondary cache.

Table 2-5 lists the possible system bus transactions which may result from processor, snoop, or flush transactions.

Table 2-5. System Bus Transaction Types

Transaction	Description
Single-Beat Transactions	
Cache-inhibited read	Read transactions that are cache-inhibited cause the MC88410 to read a byte, half word, word, or double word from an external device.
Cache-inhibited write	Write transactions that are cache-inhibited cause the MC88410 to write a byte, half word, word, or double word to an external device.
Write-through	A write-through transaction causes processor store instruction to write through the secondary cache directly to memory.
Locked store-load	A locked store-load transaction is always interpreted by the MC88410 to be a cache-inhibited transaction. The store acts as a cache-inhibited write and the load acts as a cache-inhibited read.
Cache-inhibited locked load-store	A locked load-store transaction is cacheable by the secondary cache unless it is a cache-inhibited transaction. The store acts as a cache-inhibited write and the load acts as a cache-inhibited read.
Cacheable locked load-store in write-through	A locked load-store transaction is cacheable by the secondary cache. In the write-through mode, the transaction propagates to the system bus as a read-with-intent-to modify single-beat transaction if it misses in the secondary cache, or as a single-beat write if it hits in the secondary cache.
Burst Read Transactions	
Secondary cache line fill	A processor read access that misses in the secondary cache causes a system bus transaction to occur in which an entire line of data is read from external memory and written to the secondary cache and (if necessary) the processor. This operation is called a cache line fill operation. A cache miss occurs when caching is enabled and the instruction/data required by the processor is not resident in the secondary cache.
Secondary cache read-with-intent-to-modify	A read-with-intent-to-modify transaction is caused by a write access that misses in the primary data cache and the secondary cache in write-back mode. A read-with-intent-to-modify transaction operates like a burst read transaction for a secondary cache line fill but has the side effect of broadcasting to snooping devices that the cache line being read will be modified.
Burst Write Transactions	
Replacement copyback	When a secondary cache miss occurs and the corresponding secondary cache line has a valid entry, the line is replaced. The MC88410 checks the state of the line to be replaced, and if the line is modified, the line is copied back to external memory. This operation is called a replacement copyback.
Snoop copyback	When a snooping MC88410 has a primary or secondary cache hit during a global transaction, the snooping MC88410 determines if the cache line is modified. If the line is modified, the line must be copied back to memory before the device performing the global access can complete its transaction. This operation is called a snoop copyback.
Flush copyback	The MC88410 has a flush feature that causes either all modified lines or a page of modified lines in the secondary cache to be transferred out to external memory. Each line is transferred by a burst write transaction called a flush copyback.

Table 2-6 lists the invalidate transactions that are used by the MC88410 and MC88110 to maintain vertical and lateral coherency.

Table 2-6. Invalidate Transaction Types

Transaction	Description
Processor invalidate	Processor invalidate transactions are single-beat transactions used by the MC88110 to maintain cache coherency. Processor invalidate transactions broadcast to the MC88410 that a shared line in the cache will be modified; thus, the MC88410 must invalidate its cached versions of the memory. The MC88410 treats the processor invalidate as a single-beat write and transfers valid data.
System invalidate	System invalidate transactions are single-beat transactions used by the MC88410 to maintain cache coherency among multiple MC88410s. System invalidate transactions broadcast to snooping devices that a shared line in the primary or secondary cache will be modified; thus, snooping devices must invalidate their cached versions of the memory. No data transfer is required during the system invalidate transaction.
System DMA invalidate	The DMA invalidate is a global burst read transaction that is outside the MC88410 system bus protocol. Because the MC88410 is unable to produce a global burst write, it is assumed that an external device is overwriting memory. If the DMA invalidate hits in the secondary cache, the cache line will be invalidated.
Primary cache invalidate	The primary cache invalidate transactions are single-beat transactions used by the MC88410 to maintain vertical cache coherency between the primary and secondary cache. Primary cache invalidate transactions broadcast to the MC88110 that a shared line in the primary cache will be modified and the MC88110 must invalidate its cached versions of the memory. There is no data transferred during the primary cache invalidate transaction.
Primary cache DMA invalidate	A system DMA invalidate that hits in the PTAG is propagated to the processor as a global burst write. No data is transferred during a processor DMA invalidate.

2.6 BURST ORDERING AND STREAMING

The MC88410 provides burst data transfers across both the processor and system interfaces. Transfers across the processor interface always start with the double word presented by the processor and continue with the subsequent double word(s) in the line. If the first double word is not the first double word in the line, the fill wraps around and fills the double word(s) at the beginning of the line. The MC88410 increments the address internally and sequences the MCM62110 array; the addresses incremented by the processor are not used.

The order of burst addressing on the system interface is programmable at reset. The two options are zero-word-first and critical-word-first. With critical-word-first operation, the burst transfer on the system interface starts with the same double word as the burst transfer on the processor interface and continues with the subsequent double word(s) in the line, wrapping around if necessary. With zero-word-first ordering, the bursts on the system interface always start with double word zero. This ordering applies to both secondary cache line fills and secondary cache snoop copyback transactions. Replacement copyback transactions always start with word zero. The MC88410 provides all addresses (four or eight) for burst operations on the system interface.

The MC88410 uses data streaming to reduce the penalty seen by the processor on secondary cache misses. With data streaming, data is written to the processor bus for the primary cache line fill as it is being written into the secondary cache from memory. Data streaming is straightforward for configurations with 32-byte secondary cache line size and

critical-word-first system ordering. In this case, the data from each of the four transfers is written to both the secondary and primary caches. Note that the data is valid on the processor bus one clock after it is written to or read by the secondary cache. For information on the streaming timing see **Section 5 System Bus Interface**. Upon transfer of the last word to the primary cache, the operation is completed. An example of this is shown in Figure 2-11.

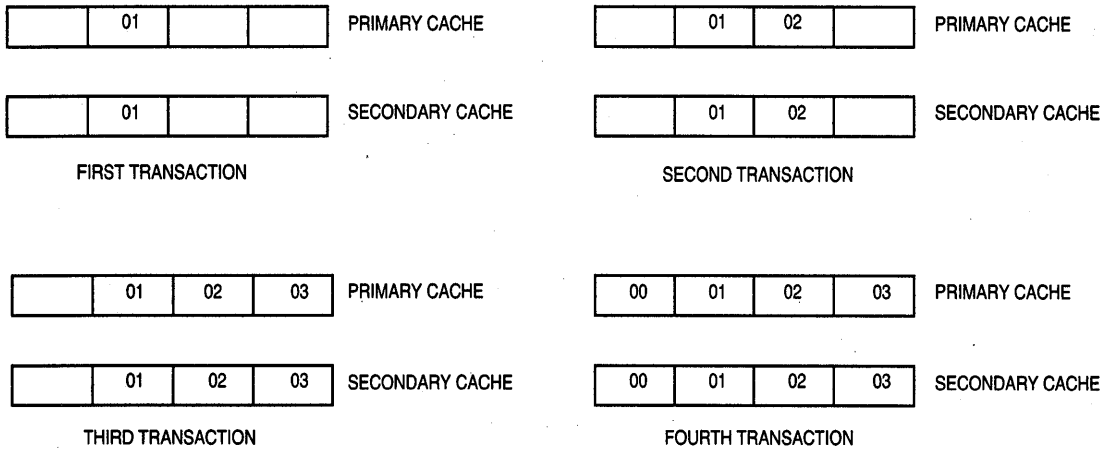


Figure 2-11. Streaming with 32-Byte Secondary Cache Line and Critical-Word-First

Data streaming is more complex when the MC88410 is configured to have 64-byte secondary cache line sizes and/or zero-word-first system ordering. For example, when the system ordering is changed to zero-word-first, the secondary cache line is filled starting at word zero, regardless of which double word contains the critical information. As the secondary cache line fill continues, streaming begins when the critical word from the processor is reached. Upon completion of the secondary cache line fill, the MC88410 wraps around and completes the burst transaction to the processor.

Bus errors detected on the system bus during streaming must be passed to the processor bus. For configurations involving the 64-byte line size, if the critical word is the first word of the line, the entire primary cache line could be streamed to the processor before the secondary fill has completed. However, if a bus error occurs on the secondary cache line fill after the primary cache line fill has completed, then there would be a valid primary line that was never loaded in the secondary cache violating the inclusion policy. To prevent this situation, the MC88410 does not complete the last transfer of the primary cache line fill until the secondary line has completed its line fill.

Figure 2-12 shows an example of when the critical double word is the second one in the line. In the first transfer, the data for the first double word is read into the secondary cache lines. For the three subsequent transfers, the data is read into the secondary cache line and streamed to the processor bus and the primary cache line. Upon successful

completion of the secondary cache line fill, the MC88410 must complete the primary cache line fill by wrapping around and writing the first double word on the processor bus.

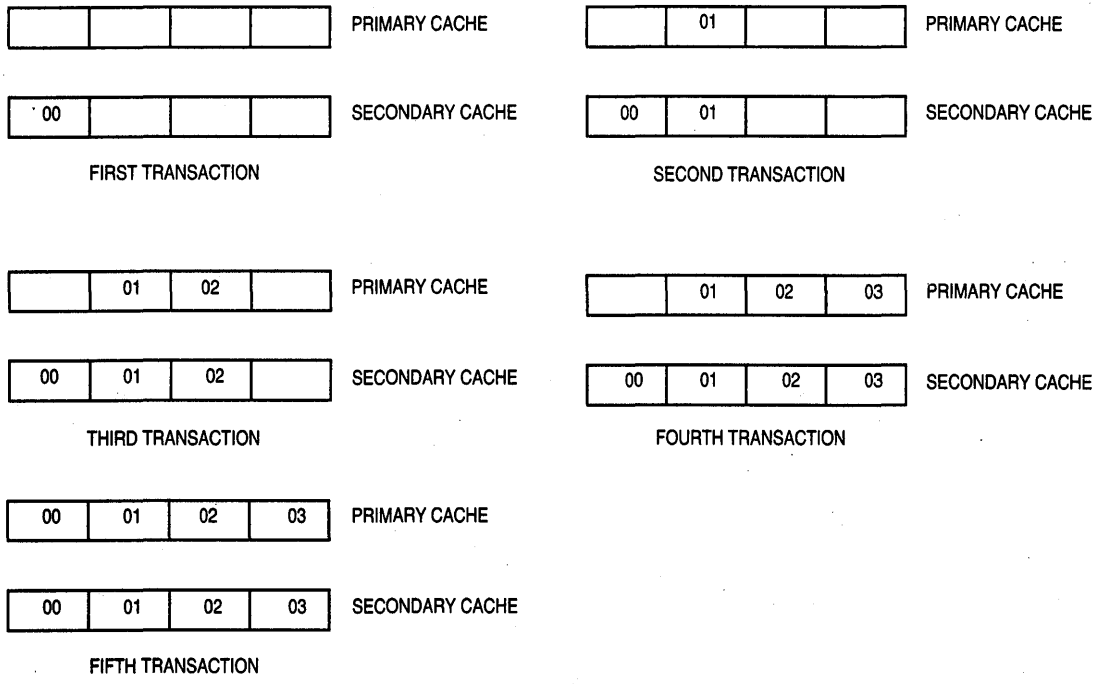


Figure 2-12. Streaming with 32-Byte Secondary Cache Line and Zero-Word-First

Another possible configuration is with 64-byte secondary cache lines and critical-word-first operation on the system bus. In this case, the critical double word is the first transaction on the system bus, and the data is streamed to the processor bus during the transaction. The data continues to be streamed for each of the double words on the first half of the secondary cache line. While the second half of the secondary cache line is being read, the MC88410 inserts wait states on the processor bus. When the secondary cache line fill wraps around, the data streaming continues until the end of the line fill.

Figure 2-13 shows an example of when the critical double-word is the second one in the line. In the first transfer, the critical double word is read into the secondary cache line and streamed to the processor. The data for the subsequent two transfers is also read into the secondary cache and streamed to the processor. For the next four transfers, however, the data is read into the secondary cache while the MC88410 inserts wait states to the processor. When the secondary cache line fill wraps around to complete, data streaming resumes for the last double-word transfer.

Finally, the MC88410 could be operating with 64-byte secondary cache lines and zero-word-first operation on the system bus. In this case, the first double word is the first transaction on the system bus. The secondary cache line fill continues, and streaming begins when the critical word from the processor is reached. During the four double-word transfers for the second half of the secondary cache line, streaming is discontinued and the MC88410 inserts wait states on the processor bus. Upon completion of the secondary cache line fill, the MC88410 wraps around and completes the burst to the processor.

Figure 2-14 shows an example of when the critical double word is the second one in the line. In the first transfer for this case, the data for the first double word is read into the secondary cache lines. For the three subsequent transfers, the data is read into the secondary cache line and streamed to the processor bus and the primary cache line. During the remainder of the secondary cache line fill, the processor bus waits. Upon successful completion of the secondary cache line fill, the MC88410 must complete the primary cache line fill by wrapping around and writing the first double word on the processor bus.

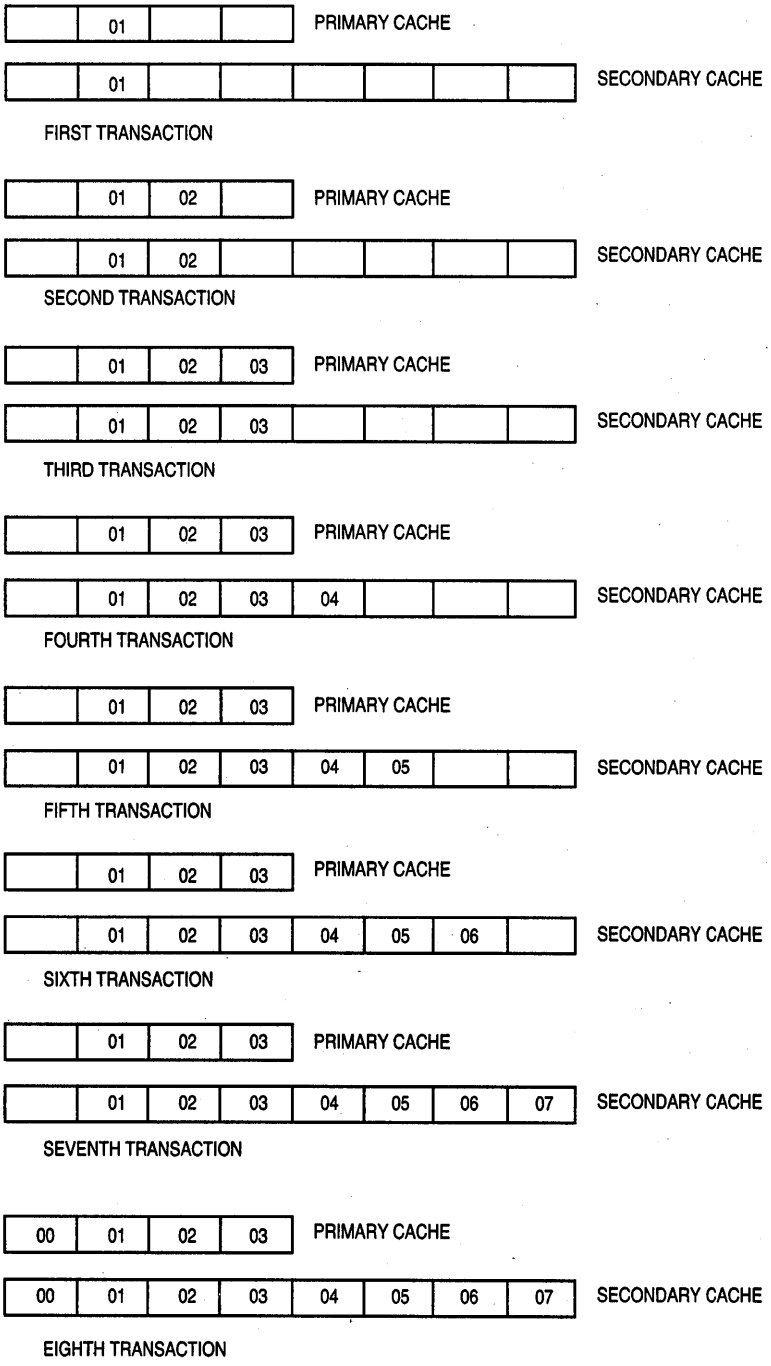


Figure 2-13. Streaming with 64 Byte Secondary Cache Line and Critical-Word-First

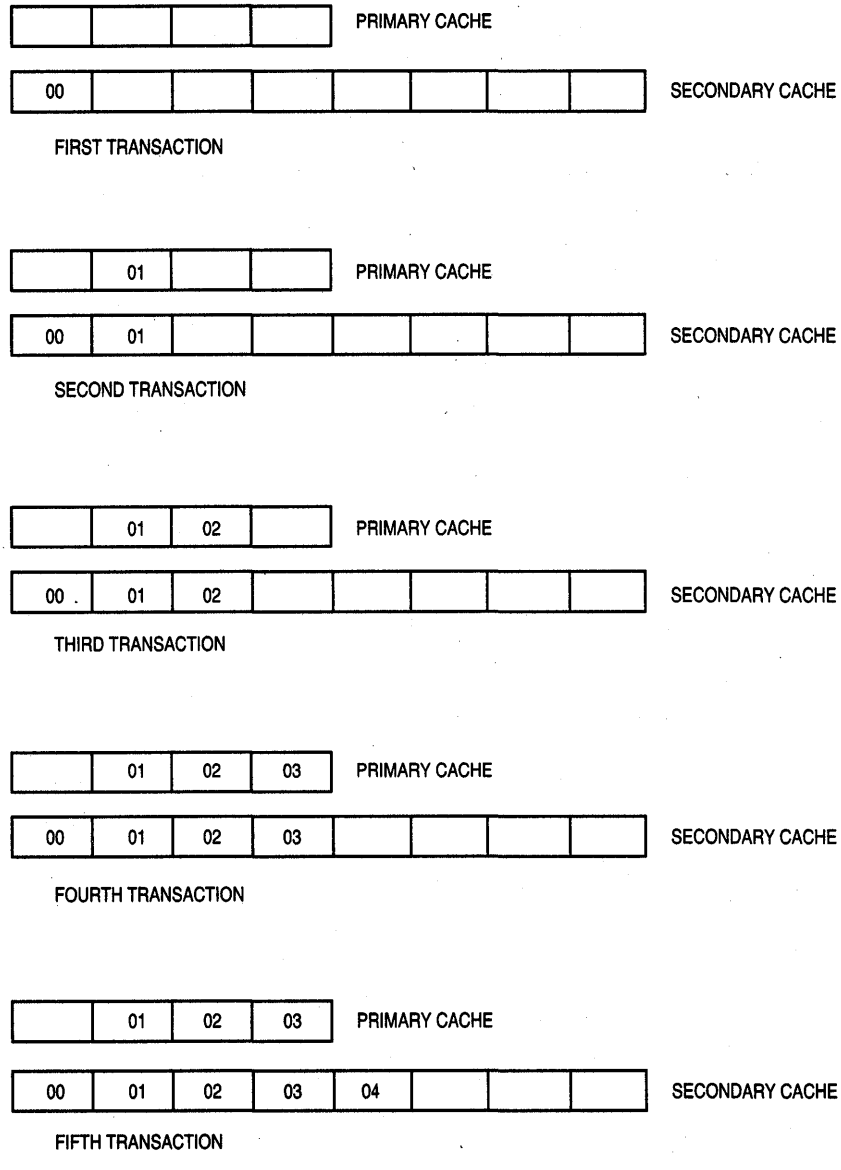


Figure 2-14a. Streaming with 64 Byte Secondary Cache Line and Zero-Word-First

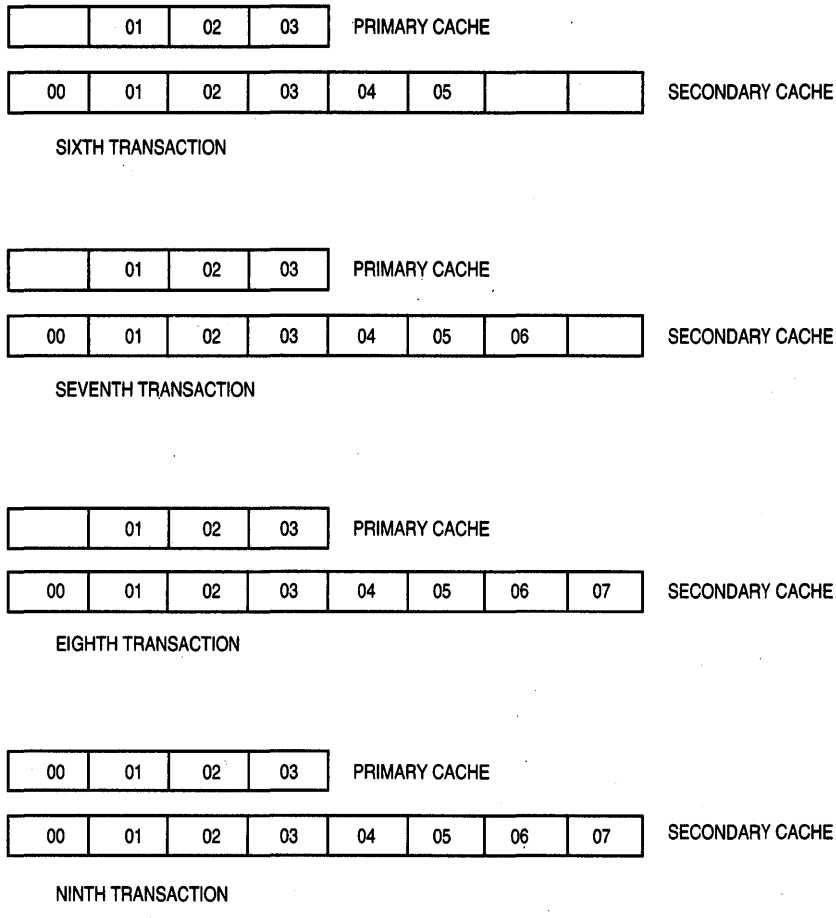


Figure 2-14b. Streaming with 64 Byte Secondary Cache Line and Zero-Word-First

2.7 SECONDARY CACHE LINE ALLOCATION

When the processor initiates a transaction for a cacheable location that misses in the secondary cache, the MC88410 must allocate a line and perform a secondary cache line fill, as shown in Figure 2-15. If the line to be replaced is included in the primary data cache (PTAG hit), the MC88410 retries the read transaction, acquires the processor bus, and initiates a primary cache invalidate transaction to the MC88110. If the primary cache contains modified data, the MC88110 asserts $\overline{\text{P\_ARTRY}}$ and performs a snoop copyback before invalidating its cache line. The snoop copyback causes the MC88410 to update the secondary cache line and negate the inclusion bit in the PTAG. If the primary cache does not contain modified data, then the MC88410 simply negates the inclusion bit. When the MC88110 reinitiates the read transaction, the transaction causes another secondary cache miss; however, during the second read the line to be replaced is a PTAG miss. Note that configurations using a 64-byte line size require two primary cache invalidate transactions.

If the secondary cache line that is to be replaced is not in the primary cache and is marked unmodified, the MC88410 arbitrates for the system bus and performs a secondary cache line fill (using either critical-word-first or zero-word-first order as specified at reset) of either 32 or 64 bytes, streaming to the processor bus as appropriate. If the line to be replaced is exclusive-modified, a secondary cache copyback precedes the line fill. If the processor transaction is a write- or a read-with-intent-to-modify, the system bus line fill transaction is intent-to-modify, and the line is marked exclusive-modified. If the processor transaction is a read-without-intent-to-modify, then the line is marked shared-unmodified if the $\overline{\text{TSHD}}$ or $\overline{\text{SHD}}$ signals were asserted at the appropriate time during the line fill (see **Section 5 System Bus Interface**); otherwise, the line is marked exclusive-unmodified.

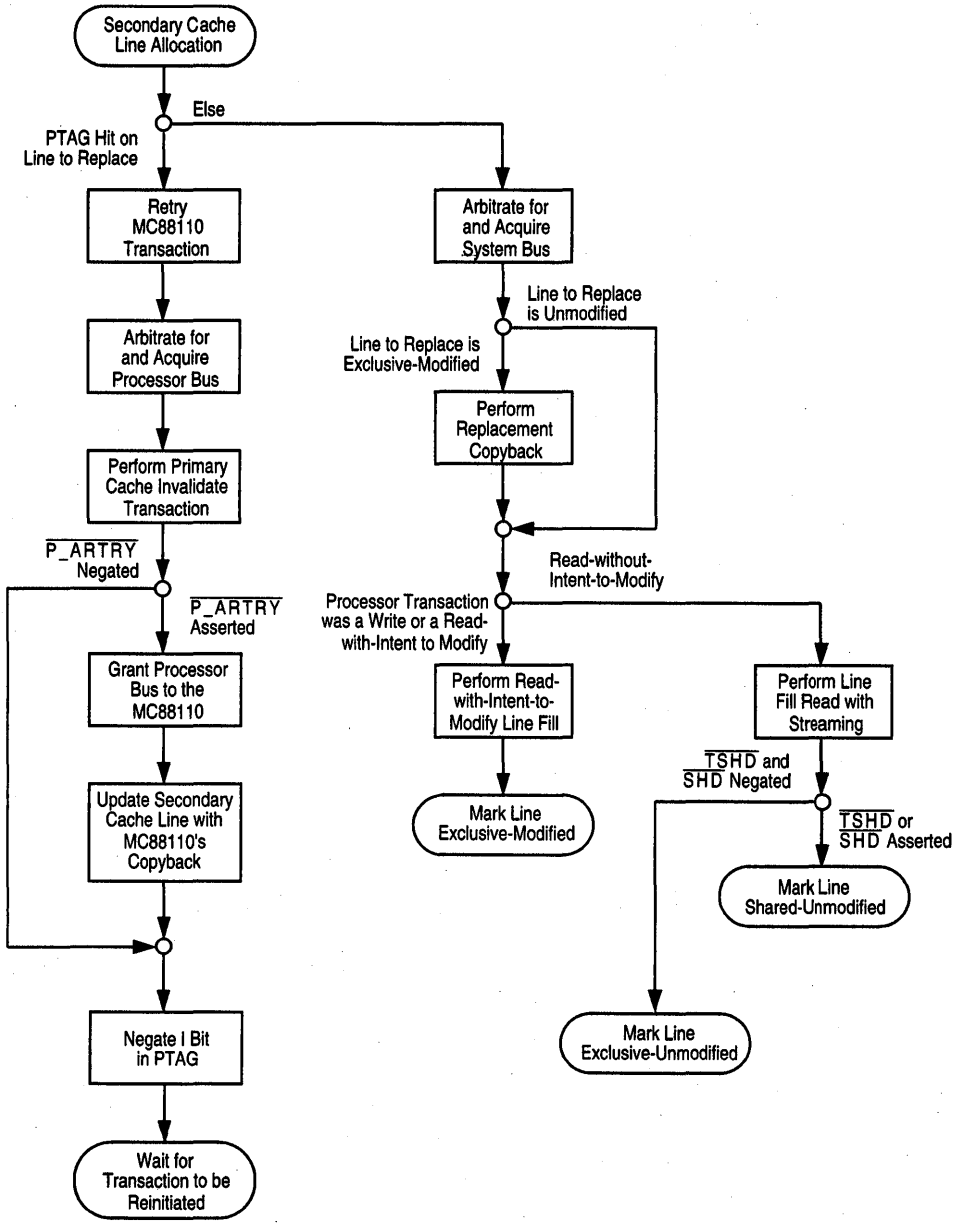


Figure 2-15. Secondary Cache Line Allocation Flow

2.7.1 Processor Read Transactions (Not Locked)

The MC88110 requests data either in a single-beat read or a burst read transaction. If the read transaction is due to a write miss or an allocate load transaction, then the MC88110 intends to modify the data as soon as it is received and the transaction is labeled intent-to-modify. The actions taken by the MC88410 due to a processor read transaction depend on the size of the transaction, whether it is intent-to-modify, whether there is a secondary cache hit, and the state of the line if there is a secondary cache hit. The following paragraphs give a brief description of the types of processor read transactions, followed by a description of the MC88410 actions caused by each transaction type.

2.7.1.1 Single-Beat Read Transaction

Single-beat read transactions include read accesses in write-through mode, table search descriptor fetches, allocate loads, and cache-inhibited reads. Note that although the table-search descriptor fetches are always cache-inhibited in the primary cache, they are cacheable in the secondary cache. The allocate load is a feature provided by the MC88110 that allows allocation of a primary cache line with only one beat of data transfer instead of the usual four. An allocate load that is not cache-inhibited and misses in the secondary cache causes a full secondary cache line to be allocated even though only 64 bits are transferred to the processor. A cache-inhibited allocate load does not cause a secondary cache line to be allocated, and acts as a cache-inhibited read. The allocate load is the only single-beat read (except the locked read, described in **2.7.3 Locked Transactions**) which is marked intent-to-modify.

2.7.1.2 Burst Read Transaction

Burst read transactions include line fills of the primary instruction or data caches. If the request hits in the secondary cache, data is provided to the processor using critical-word-first burst ordering of four 64-bit beats. If the request misses, the MC88410 takes actions to fill a secondary cache line with the requested data and to update the tags accordingly. If the request is a data access, the appropriate line in the PTAG is updated to reflect the newly included line. A system bus burst read that results from a read miss in the secondary cache is called a secondary cache line fill.

Burst read transactions from the processor are qualified as intent-to-modify if they are the result of a write miss in the primary data cache. On a secondary cache miss, this qualification is carried on the system interface for the secondary cache line allocation, allowing the new secondary cache line to be marked exclusive-modified. On a secondary cache hit, a system invalidate transaction and MTAG update to exclusive-modified precedes the processor response if needed. A system bus burst read that results from a processor read-with-intent-to-modify transaction is called a secondary cache read-with-intent-to-modify.

2.7.1.3 Read Transaction Flow

The processor read transaction flow is shown in Figure 2-16. When the MC88410 recognizes a read transaction on the processor bus, it determines if the transaction is cache-inhibited and if there is a secondary cache hit. If it is cache-inhibited, the MC88410 performs a single-beat read on the system interface and transfers the data to the processor

interface. If there is a cache hit on a cache-inhibited transaction, the secondary cache line is marked invalid (after a copyback if the line was modified) preceding the single-beat read on the system interface.

If the transaction is not cache-inhibited and there is a secondary cache hit, then the actions of the MC88410 depend on the state of the cache line and whether the transaction is intent-to-modify. If the transaction is intent-to-modify and the line is shared-unmodified, the MC88410 must successfully perform a system invalidate transaction and mark the line exclusive-modified before completing the processor read transaction. If the transaction is intent-to-modify and the line is marked exclusive-unmodified, the MC88410 must mark the line exclusive-modified before completing the transaction. If the transaction is intent-to-modify and the line is marked exclusive-modified, or if the transaction is not intent-to-modify, the transaction can be completed immediately.

If the transaction is not cache-inhibited and there is a secondary cache miss, then the MC88410 must perform a secondary cache line fill before completing the processor read transaction. If the transaction is single-beat, the critical word is streamed to the processor and its transaction completes. Once the secondary cache line fill is complete, if the read transaction is a burst, the MC88410 completes the burst (if necessary), updates the tag entry in the PTAG, and sets the inclusion bit.

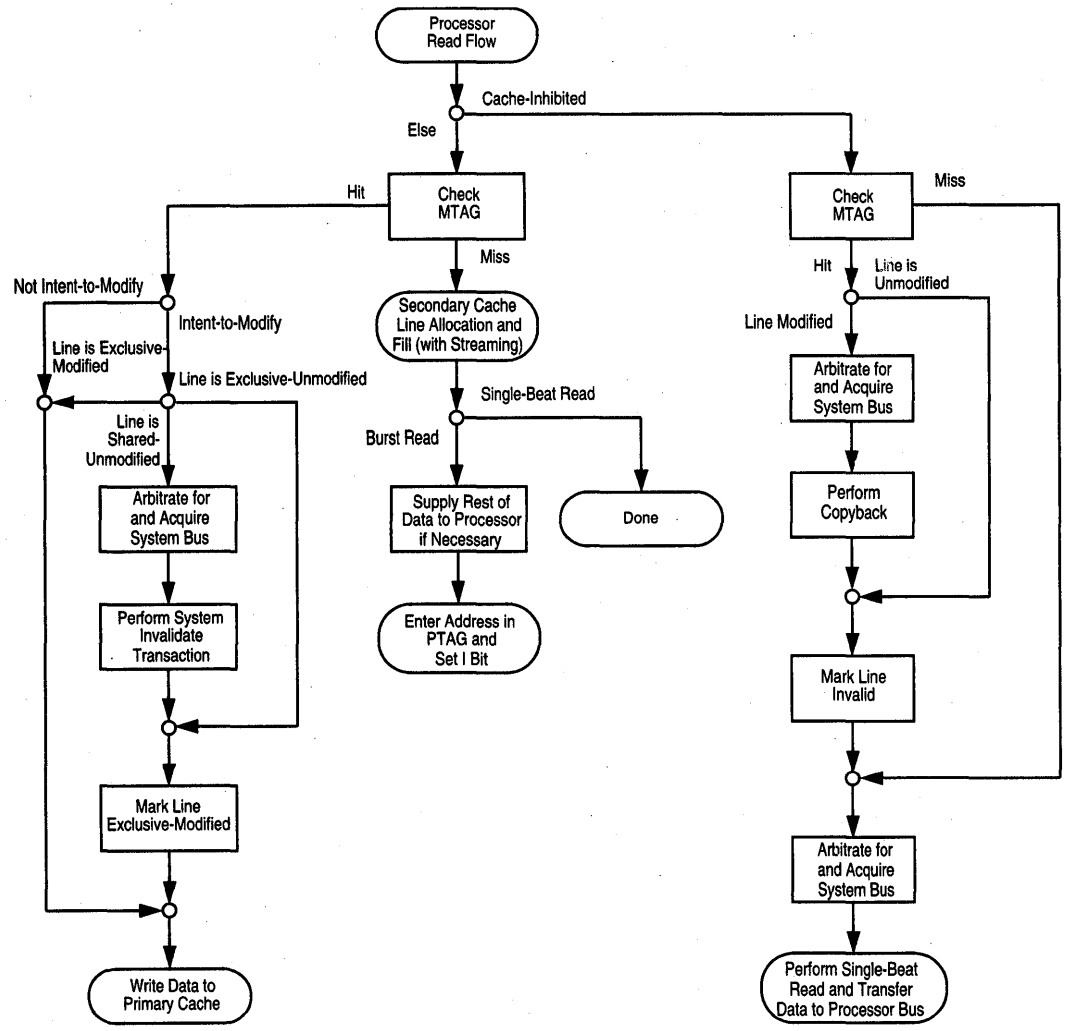


Figure 2-16. Processor Read Transaction Flow

2.7.2 Processor Write Transactions (Not Locked)

The MC88110 writes data either in a single-beat write or a burst write transaction. The actions taken by the MC88410 due to a processor write transaction depend on the size of the transaction, whether there is a secondary cache hit, and the state of the line if there is a secondary cache hit. The following paragraphs give a brief description of the types of processor write transactions, followed by a description of the MC88410 actions caused by each transaction type.

2.7.2.1 Single-Beat Write Transaction

Single-beat write transactions include primary cache write-through accesses, write-back accesses, and cache-inhibited writes. Write-through accesses that hit the secondary cache are written into it and to the system interface. No state change is involved. Write-through accesses that miss the secondary cache do not allocate a secondary line and are written directly to the system interface without affecting the secondary cache contents.

The single-beat write-back transaction occurs for three different reasons. The first is as a result of the MC88110 attempting to write to a shared-unmodified line in the primary cache which follows the write-back policy. In this case, the MC88110 performs a processor invalidate transaction before writing the data to the primary cache line to alert the secondary cache that it intends to modify that line. For subsequent writes to that line, the MC88110 does not perform a bus transaction at all. The processor invalidate is identical to a single-beat write-back transaction, except that the MC88110 $\overline{\text{MC}}$ signal is negated, indicating that it is not necessary to complete the memory transaction, but that snooping processors should invalidate their cached lines. Since the $\overline{\text{MC}}$ signal is not connected to the MC88410, the MC88410 does not distinguish between the processor invalidate and the single-beat write-back transaction. Processor invalidate transactions from the processor always hit in the secondary cache.

The single-beat write-back transaction also occurs when the MC88110 is in the forced write-through mode. This feature forces the primary cache to use a write-through policy regardless of the memory management unit's (MMU) mapping, so the MC88110 performs a single-beat write for all store instructions. Since the MMU's configuration is not affected by this, the memory update policy is still available for use at the secondary cache level. Therefore, for a write to a page marked as write-back in the MMU, the MC88110 performs a single-beat write transaction but specifies that the secondary cache should follow the write-back policy by negating the $\overline{\text{P\_CI}}$ and $\overline{\text{P\_WT}}$ signals.

If a single-beat write-back transaction misses in the secondary cache, the MC88410 performs a burst read-with-intent-to-modify transaction, updates the secondary cache line, and marks the line exclusive-modified. If the transaction hits and the cache line is marked shared-unmodified, the MC88410 performs a system invalidate transaction, updates the secondary cache, and marks the line exclusive-modified. If the transaction hits and the cache line is marked exclusive-unmodified, then the MC88410 simply updates the secondary cache and marks the line exclusive-modified.

Finally, if the transaction is cache-inhibited ($\overline{\text{P\_CI}}$ asserted), the write transaction will not be written into the primary or secondary cache and will be written to main memory as a single-beat transaction.

2.7.2.2 Burst Write Transaction

For processor burst write transactions, the MC88410 updates the secondary cache in critical-word-first order. The data being copied back is written into the secondary cache only. MC88110 snoop copybacks cause the appropriate inclusion bit in the PTAG to be cleared because the MC88110 invalidates its primary cache line after a snoop copyback. The inclusion bit remains set for replacement and flush copyback transactions. For the replacement copyback, the inclusion bit is left set, but the address in the tag is updated when the processor performs its subsequent line fill. For flush copybacks, the inclusion bit is left set because if the copyback is a result of a flush command (not flush and invalidate) the MC88110 does not invalidate the line after the copyback.

2.7.2.3 Write Transaction Flow

The processor write transaction flow is shown in Figure 2-17. When the MC88410 recognizes a write transaction on the processor bus, it determines if the transaction is cache-inhibited. If it is cache-inhibited, the MC88410 performs a single-beat write transaction on the system interface. If there is a cache hit on a cache-inhibited transaction, the secondary cache line is marked invalid (after a copyback if the line was modified) before the single-beat write on the system interface.

If the transaction is a processor burst transaction, the MC88410 updates the secondary cache line. If the transaction is a snoop copyback, the MC88410 assumes that the MC88110 has invalidated its cache line and negates the inclusion bit.

If the transaction is a single-beat transaction with the write-through policy in effect, then the actions of the MC88410 depend on whether there is a cache hit. If there is a secondary cache hit, the data is written both to the secondary cache line and to main memory. If there is a secondary cache miss, the data is simply written to main memory.

If the transaction is a single-beat transaction with the write-back policy in effect, the actions of the MC88410 depend on whether there is a cache hit. If there is a secondary cache hit, the actions of the MC88410 depend on the state of the cache line. If the line is shared-unmodified, the MC88410 must successfully perform a system invalidate transaction before completing the processor transaction. Once the MC88410 has exclusive ownership of the line, the data is written to the secondary cache line and the line is marked exclusive-modified.

If the transaction is a single-beat transaction with the write-back policy in effect and there is a secondary cache miss, the MC88410 must perform a secondary cache line fill before completing the processor write transaction. Once the secondary cache line fill is complete, the line is updated with the data from the processor and the line is marked exclusive-modified.

2.7.3 Locked Transactions

The MC88110 supports an exchange memory (**xmem**) instruction that is a combination of a load and store instruction. The **xmem** instruction normally causes a locked read access followed by a locked write access. However, the **xmem** instruction can also function as a locked write access followed by a locked read access. The **xmem** accesses are cache-inhibited on the primary cache, but may not be treated as cache-inhibited on the secondary cache depending on the order of the transactions and the memory update policy in effect, as shown in Figure 2-18.

If the first locked transaction is a write access or a cache-inhibited read, the MC88410 treats the **xmem** as a pair of cache-inhibited accesses. If there is a cache hit on the first transaction, then the secondary cache line is marked invalid (after a copyback if the line was modified) preceding the locked read or locked write on the system interface. The second half of the locked pair of transactions is guaranteed to be a cache miss, so the transaction is passed from the processor bus to the system bus.

If the first locked transaction is a cacheable read, the MC88410 treats the **xmem** as a pair of cacheable data requests. If the locked read hits the cache, the MC88410 simply marks the line exclusive-modified and provides the data for the processor (preceded by system invalidate cycle if needed). If the locked read misses the cache, it allocates a secondary line (see **2.7 Secondary Cache Line Allocation**). The second half of the locked pair is guaranteed to be a cache hit to an exclusive-modified line, so the secondary cache line is updated with the data from the processor. If the write-through policy is in effect for the locked write, the data is written to memory with a locked single-beat write transaction.

Between the read and write halves on a cacheable **xmem**, the MC88410 maintains a lock collision buffer that retries any snooped address that attempts to access the same line address as the **xmem** (see **Section 5 System Bus Interface** for more information).

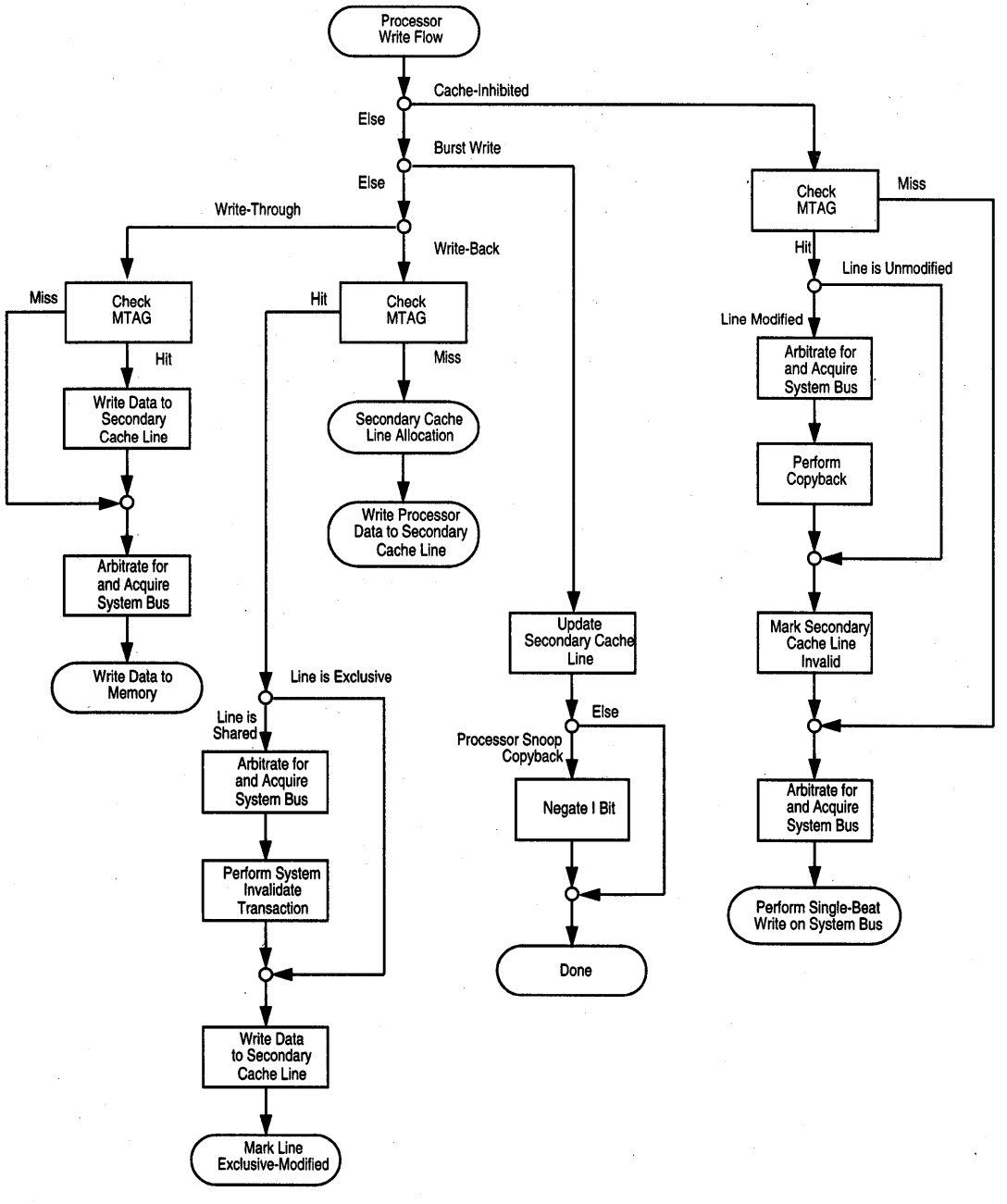


Figure 2-17. Processor Write Transaction Flow

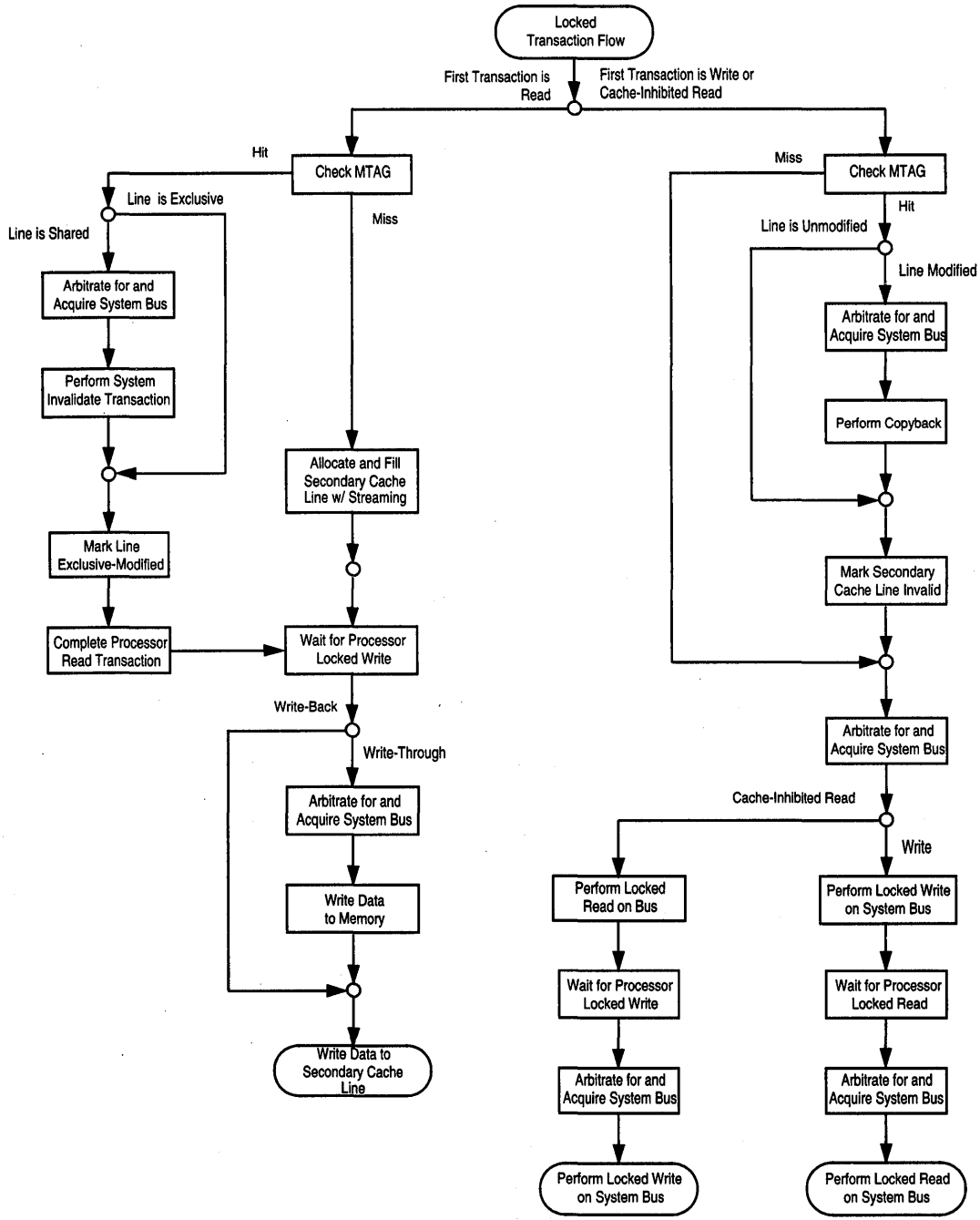


Figure 2-18. Locked Transaction Flow

2.8 CACHE FLUSHING AND INVALIDATION

The MC88410 supports the ability to flush a single page or the entire cache while maintaining both vertical and horizontal cache coherency. It also allows for the incoherent invalidation of the cache. Flush page and flush all are coherent operations that cause the MC88410 to execute copyback transactions and primary cache invalidate transactions as needed. For the invalidate all operation, the MC88410 simply clears all the V bits in the MTAG and all the I bits in the PTAG. During a flush operation, the MC88410 is available for snooping and to process additional MC88110 transactions. However, during an invalidate operation, snooping is disabled and MC88110 transactions are stalled.

The flush page, flush all, and invalidate all operations begin at the lowest address in the selected granularity regardless of the actual address that was used. For example, flush all and invalidate all operations begin at tag location zero and increments up to the maximum tag address. A flush page begins at set zero for the selected page address and steps up to the maximum tag address for that page.

The following paragraphs describe the flush control signals, flush operations, and invalidate operations.

2.8.1 Flush and Invalidate Control

The MC88410 system interface has two flush control input signals: FO and F1. It also includes the $\overline{\text{FBSY}}$ output signal to indicate when a flush or invalidate is in progress. The MC88410 initiates the flush and invalidate operations when it detects the appropriate encoding of flush control signals for at least one clock cycle. For the flush page operation, the MC88410 uses the page specified by the last cache-inhibited write before the flush control signals are detected (see **2.8.2 Flush Page and Flush All Operations** for more information). The encoding for F1 and FO is shown in Table 2-7.

Table 2-7. Flush Control Signal Encoding

F1	F0	Function
0	0	No operation
0	1	Flush page
1	0	Flush all
1	1	Invalidate all

To initiate a flush or invalidate operation, the appropriate flush control signals must be asserted for one clock cycle. When the MC88410 recognizes the flush control signal encoding, it asserts the $\overline{\text{FBSY}}$ signal one clock later and begins the flush or invalidate operation. Once one of the three operations has been initiated, the flush control signals are not sampled again by the MC88410 until the current operation completes and the signals are restored to the no operation state. When the flush or invalidate operation has completed, the MC88410 negates the $\overline{\text{FBSY}}$ signal. The $\overline{\text{FBSY}}$ signal may be used to clear the external control register that drives the flush control signals.

Figure 2-19 shows the hardware required for one possible method for controlling the flush and invalidate mechanism. This illustration shows a control register and an address decoder added to the basic MC88110/MC88410 configuration. To initiate a flush or invalidate operation, the MC88110 performs a cache-inhibited write to the address of the control register. The data for this transaction is then latched into the control register, which asserts the appropriate flush control signals. The address of the control register must be encoded in address bits 31–20, because the lower 20 bits of the address are required to specify a page for the flush page operation.

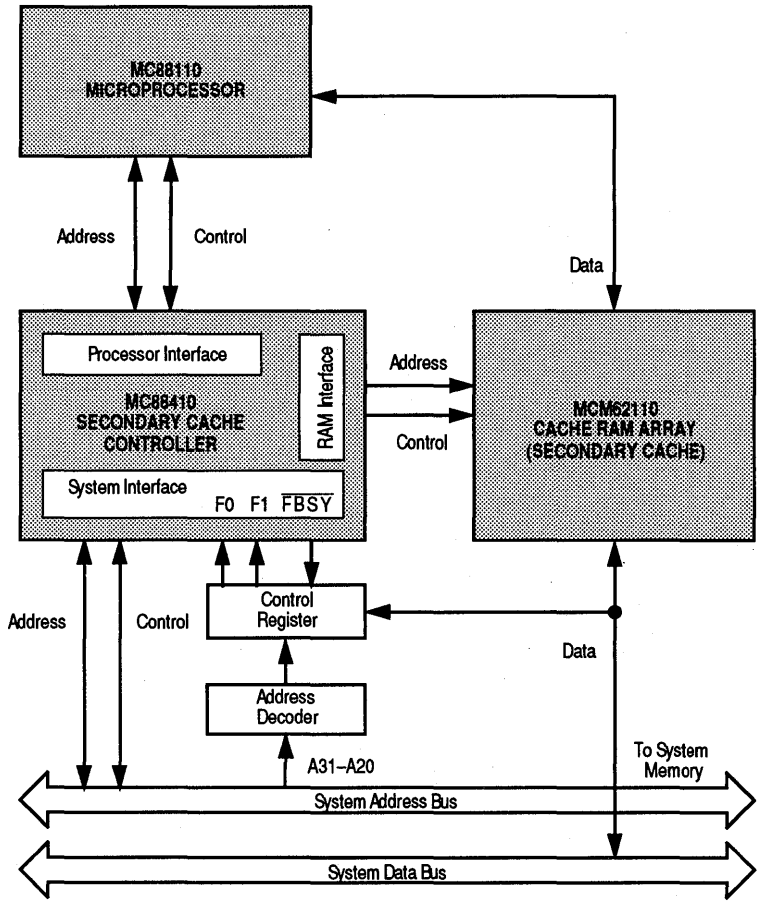


Figure 2-19. Flush Control Hardware

2.8.2 Flush Page and Flush All Operations

The flush page and flush all operations are coherent operations that cause the MC88410 to execute copybacks and primary cache invalidations as needed. The flow for the flush operations is shown in Figure 2-20. When the flush control signals indicate that a flush operation should occur, the MC88410 asserts $\overline{\text{FBSY}}$ and starts at the lowest address of the cache or of the specific page in the cache. For each MTAG entry, both the MTAG and the PTAG are checked. If the MTAG entry indicates that the line is unmodified, no action needs to be taken, the address is incremented, and the next MTAG entry can be checked.

If the MTAG entry indicates that the line is modified, the MC88410 must copy back the most recent copy of that line to main memory. If there is a hit in the PTAG with the inclusion bit set, the MC88410 must perform a primary cache invalidate transaction to determine whether the MC88110 has the most recent copy of the line. For more information about the primary cache invalidate transaction, see **Section 4 Processor Bus Interface**. If the MC88110 has a version of the line that is modified with respect to the secondary cache line, it performs a snoop copyback, the secondary cache is updated, and the inclusion bit of the PTAG entry is cleared. Once the MC88410 has ensured that it has the most recent copy of the cache line, it performs a copyback to main memory, clears the modified bit in the MTAG entry, and goes on to the next line.

To initiate a flush operation, the MC88110 can perform a cache-inhibited write to a control register on the system bus. The address is decoded as shown in Figure 2-21. Address bits 31–20 are used for the control register decoder, and address bits 19–0 specify the page to be flushed. The data for the cache-inhibited write transaction is latched into the control register and sets the flush control pins to the flush page encoding. When the MC88410 recognizes the flush page encoding on the flush control signals, it asserts $\overline{\text{FBSY}}$ and begins the flush.

Note that for the flush page operation, the lower 20 bits of the 32-bit address is needed to specify the page to be flushed. Since the page address issued by the MC88110 occupies the lower three bits of the address bus, the misaligned access exception of the MC88110 must be disabled by setting the MXM bit in the processor status register of the MC88110. When a misaligned access is attempted with the MXM bit set, the processor asserts the full misaligned address on the address bus, but performs the access to the next lower properly aligned boundary.

When initiating a flush page operation, the system must be careful to ensure that the proper page address is latched in the MC88410 when it recognizes the flush page encoding. Each time the MC88410 sees a cache-inhibited write on the processor bus, it latches the page address into the flush counter. The write is then propagated through the MC88410 and onto the system bus. When the write transaction is complete, the MC88110 is free to initiate another transaction. It is possible for the system to assert $\overline{\text{S\_TA}}$ before the cache-inhibited write transaction has modified the control register, thus asserting the flush control signals. The MC88110 must not initiate another cache-inhibited write transaction to the MC88410 before the MC88410 recognizes the flush page encoding on the flush control signals and initiates the flush.

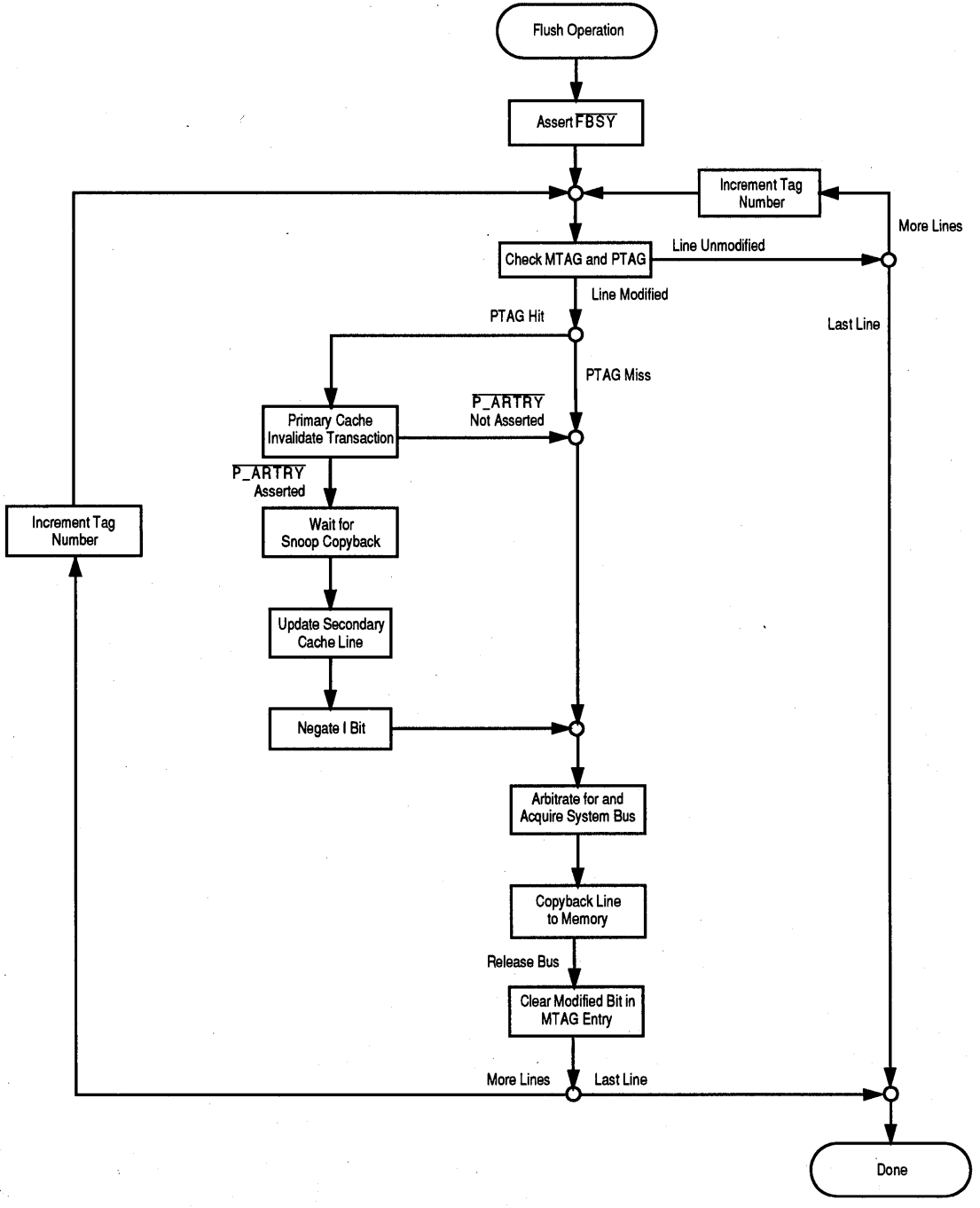


Figure 2-20. Flush Operation Transaction Flow

Otherwise, a new address is latched into the flush counter, and when the MC88410 recognizes the flush page encoding, it begins the flush operation to the wrong page. Since the page address is ignored for the flush all operation, it is not necessary for the MC88110 to avoid all cache-inhibited write transactions before the flush all begins.

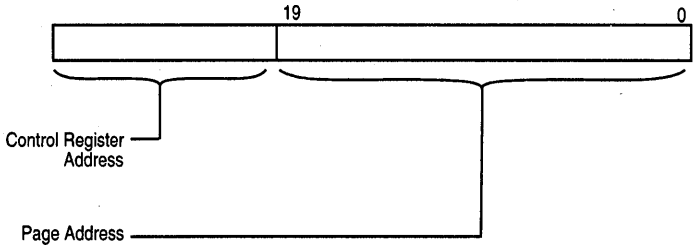


Figure 2-21. Flush Address Decode

Once the MC88410 has begun the flush, the MC88110 is free to perform any memory transaction it needs, including cache-inhibited write transactions. The MC88410 processes the requests of the MC88110 as it would normally. Snooping is also enabled during the flush operations. If there is contention in the PTAG or the MTAG during the flush, the flush is given the lowest priority.

The duration of the flush page depends on the number of modified lines in the specified page and the level of activity on the processor and system buses. The flush page requires two clock cycles for each line on the page, plus the time it takes to copy back any modified lines to memory. The minimum number of clocks for a flush page with no copybacks is 260. In addition, the flush page operation may be delayed if there is any contention in the MTAG. The duration of the flush all depends on the number of modified lines in the cache and the level of activity on the processor and system buses. The minimum number of clocks for a flush all with no copybacks is 16,388 for a 256K cache with a 32-byte line size. The minimum number of clocks for a flush all with no copybacks is 8,192 for a 256K cache with a 64-byte line size.

2.8.3 Invalidate All Operation

The invalidate all operation is initiated in the same way as the flush all operation. For example, the MC88110 performs a cache-inhibited write to a control register on the system bus. Address bits 31–20 are used for the control register decoder. The MC88410 ignores the page address and invalidates the whole secondary cache. The data for the cache-inhibited write transaction is latched into the control register and sets the flush control pins to the invalidate all encoding. When the MC88410 recognizes the invalidate all encoding on the flush control signals, it asserts $\overline{\text{FBSY}}$ and begins the invalidation. Since the page address is ignored, it is not necessary for the MC88110 to avoid all cache-inhibited write transactions before the invalidate begins.

The MC88110 cannot access the MC88410 during the invalidate all operation; therefore, the MC88410 delays any processor transaction by not asserting $\overline{\text{P\_TA}}$ until the invalidate has been completed. Snooping is also disabled during an invalidate all operation.

In normal operation, the secondary cache must be invalidated at reset. The MC88410 performs an invalidate all operation at reset only if the flush control signals are both asserted when the reset signal is negated and if the signals stay asserted for at least three clock cycles after reset is negated. The invalidate all operation lasts 16,388 clock cycles for a 256K cache with 32-byte line size and 8,192 clock cycles for a 256K cache with 64-byte line size. For diagnostic purposes, the secondary cache invalidation can be omitted by negating these bits during reset.

NOTE

If the secondary cache is invalidated when there is valid data in the primary data cache, the PTAG clears the inclusion bit for data which is still valid in the primary cache. This could cause the MC88410 to incorrectly interpret subsequent snoop transactions from the system bus as misses rather than hits, and data corruption can occur. This can be avoided by invalidating the primary cache before invalidating the secondary cache.

2.9 BUS SNOOPING PROTOCOL

The MC88410 can automatically maintain coherency between cached and in-memory copies of data. To maintain this coherency, the MC88410 uses a write invalidate with intervention protocol on the external bus to ensure that, at all times, only one processor node in the system has a modified copy of a given cache line. The protocol allows other caches on the bus to have local copies that are all consistent. When an MC88410 writes data to a memory location shared by other processors, the other processors are notified that their copy of the line containing that data is stale and must be invalidated.

The MC88410 snoops bus transactions by monitoring externally initiated bus transactions and comparing all global addresses to the internal data cache tags. A snoop hit occurs when the address tag for a valid MTAG entry matches the address on the bus. A primary cache snoop hit occurs when the address tag for a valid PTAG entry matches the address on the bus. In this case, the MC88410 must determine whether the MC88110 must be notified of the snooped transaction.

When monitoring external bus transactions, if a global address that matches one of the entries in the MTAG is detected ($\overline{\text{S\_GBL}}$ signal asserted during the transaction), a snoop hit occurs. When a snooping MC88410 hits with a modified entry, the snooping MC88410 asserts the $\overline{\text{S\_SSTAT1}}$ signal. The $\overline{\text{S\_SSTAT1}}$ output may then be directly or indirectly coupled to the address retry input of each processor node, forcing the processor node that initiated the access to retry the access after the modified data has been written to memory by the MC88410 that had the snoop hit. This protocol is referenced as a snoop retry exchange throughout the remainder of this section.

Figure 2-22 shows the transaction flow followed by the MC88410 for a snoop operation. The following paragraphs describe the operations depicted in the flow diagram.

2.9.1 Transaction without Intent-to-Modify

Read-without-intent-to-modify transactions that affect cache coherency are cache line fill transactions that occur due to read misses. When a read-without-intent-to-modify occurs, other caches on the bus must copy back any modified versions of the cache line and mark their cache line shared-unmodified. If another processor node on the system bus recognizes the address as global and has a modified copy of the data in its cache, it signals a snoop retry. Upon receipt of the retry signal, the initiating processor node aborts the cache line fill transaction and relinquishes the bus. The snooping processor node then acquires the bus and updates memory with its copy of the cache line. The initiating processor node then arbitrates for mastership of the bus and attempts the aborted cache line fill again.

The MC88410 performs the following actions when snooping an external read transaction. These actions represent the logical flow of operations; since the MC88410 employs a high degree of concurrency, some of the operations are performed in parallel.

When an MC88410 snoops a global read transaction that hits in the secondary cache, it determines if the cached data is modified or not. If the line is marked exclusive-unmodified, the MC88410 marks the line as shared-unmodified. In this manner, the MC88410 recognizes that other processor nodes have read access to the global data. If the line is already marked as shared-unmodified, no action is taken. For lines that are unmodified in the secondary cache, the primary cache status can be assumed to be shared-unmodified, because the $\overline{\text{SHD}}$ input signal to the MC88110 should be grounded. Therefore, the MC88110 does not need to be notified of a snoop hit to the primary cache if the transaction is not intent-to-modify.

If the line is internally modified, the MC88410 signals a snoop retry to the processor node that initiated the transfer. The initiating processor node should then abort its transaction and release the bus. If the snooping MC88410 determines that the line is also in the primary cache (PTAG hit), the MC88410 initiates a primary cache invalidate transaction. If the MC88110 has the most recent version of the line, the MC88110 copies back its copy of the line and the snooping MC88410 updates the secondary cache line and negates the inclusion bit in the PTAG. Note that if the secondary cache line size is 64 bytes, the MC88410 must check two PTAG entries and may have to perform two primary cache invalidate transactions.

The snooping MC88410 then arbitrates for mastership of the bus, writes its modified copy of the line to memory, and marks the line as shared-unmodified in its cache. The initiating processor node then arbitrates for mastership of the bus and attempts the aborted transaction again. The initiating processor node snoops the bus while it is waiting to retry the aborted transaction.

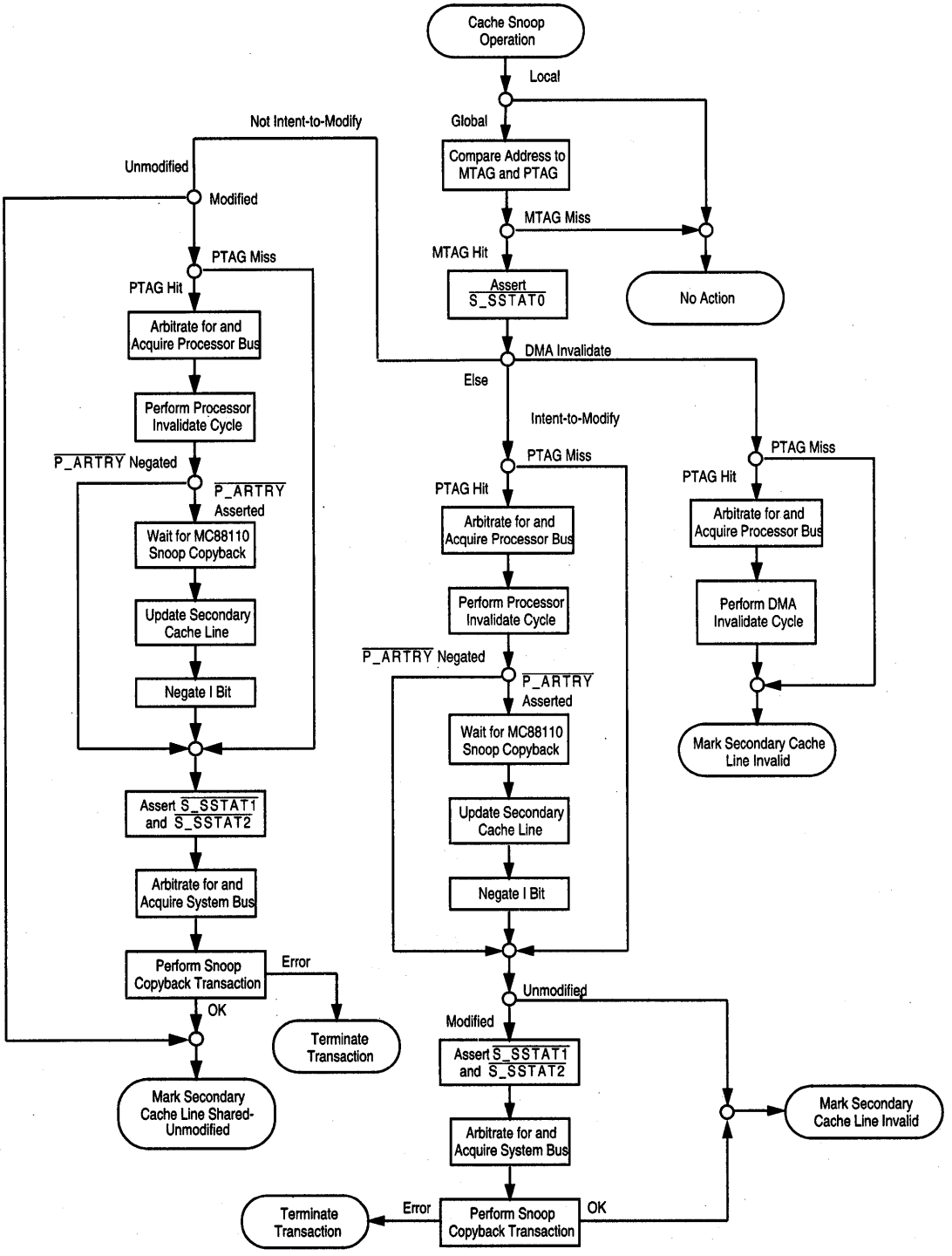


Figure 2-22. Cache Snoop Operation Transaction Flow

2.9.2 Transaction with Intent-to-Modify

Read-with-intent-to-modify transactions that affect cache coherency are locked read/write transactions (initiated by the **xmem** instruction of the MC88110), cache line fill transactions (reads) that occur due to write misses, and allocate loads. For a locked read operation followed by a write, a snooping MC88410 can hit if the read-with-intent-to-modify transaction is global, copy back its modified data, and invalidate the line in the cache (after notifying the MC88110 if necessary). The snooping processor node then monitors the write-locked transaction, but never hits since the line was already copied back and invalidated. When the locked transactions are a write followed by a read, a snooping processor node can hit if the write is global and then cause the write portion of the transaction to be retried.

When a read-with-intent-to-modify access caused by a write miss or an allocate load occurs, other caches on the bus must invalidate local copies of that cache line. If another processor node on the bus recognizes the address as global and has a modified copy of the data in its cache, it signals a snoop retry. Upon receipt of the retry signal, the initiating processor node aborts the cache line fill transaction and relinquishes the bus. The snooping processor node then acquires the bus and updates memory with its copy of the cache line. The initiating processor node then arbitrates for mastership of the bus and attempts the aborted cache line fill again.

The MC88410 performs the following actions when snooping an external write or an external read-with-intent-to-modify transaction on the bus. These actions represent the logical flow of operations; since the MC88410 employs a high degree of concurrency, some of the operations are performed in parallel.

A snooping processor that has a snoop hit during a global single-beat write or global read-with-intent-to-modify operation must determine whether the cache line is modified and if the cache line is included in the primary cache. If the cache line that hit is unmodified and is in the secondary cache only, no additional bus transaction occurs, but the cache line is marked as invalid. If the cache line is unmodified and is in the primary cache, the MC88410 performs a primary cache invalidate before marking the secondary cache line invalid.

If the cache line that hit is modified, the snooping MC88410 signals a snoop retry to the processor that initiated the transfer. The initiating processor then aborts its transaction and releases the bus. If the snooping MC88410 determines that the line is also in the primary cache (PTAG hit), the MC88410 initiates a primary cache invalidate transaction. If the MC88110 has the most recent version of the line, the MC88110 then copies back its copy of the line and the snooping MC88410 updates the secondary cache line and negates the inclusion bit in the PTAG. Note that if the secondary cache line size is 64 bytes, then the MC88410 must check two PTAG entries and may have to perform two processor invalidate cycles.

The snooping MC88410 then arbitrates for mastership of the bus, writes its modified copy of the line to memory, and marks the line as invalid in its cache. The initiating processor node then arbitrates for mastership of the bus and attempts the aborted transaction again.

The initiating processor node snoops the bus while it is waiting to retry the aborted transaction.

2.9.3 DMA Invalidate Transaction

The MC88110 and MC88410 never perform a global burst write on the system bus. Therefore, if the MC88410 detects a global burst write, it assumes that the transaction must have been generated by an external device that is overwriting some portion of memory (for example, a DMA controller); thus, there is no reason to copy back the line before invalidating. If the MC88410 has a snoop hit during a global burst write, it invalidates the cache line without copying the line back regardless of the state of the cache line. If the MC88410 determines that the cache line is also in the primary cache, it initiates a DMA invalidate cycle on the processor bus before marking the secondary cache line invalid.

2.9.4 Snooping Protocol Examples

The following examples illustrate how snooping maintains cache coherency in a multiprocessor configuration. The examples assume that there are two MC88110/MC88410 nodes that share one common external system bus with main memory. Each of the figures show a cache line within MC88110-A, MC88410-A, MC88110-B, and MC88410-B, and the associated line address tags. The state of the cache line (invalid (INV), shared-unmodified (SU), exclusive-unmodified (EU), or exclusive-modified (EM)) is also shown as well as the next state of the line as a result of bus transactions or snooping. Examples 1 and 3 only show one line in the primary data caches for simplicity. Example two shows both primary cache lines.

Examples 1 and 3 assume that the MC88410 is configured with a 32-byte line size. Example 2 shows the MC88410 configured for a 64-byte cache line. Also, the starting address is shown as $0000. Address $0008 corresponds to double word 1, address $0010 corresponds to double word 2, etc. Line read operations perform four consecutive double-word reads from memory addresses $0000, $0008, $0010, and $0018 to the cache line using the efficient burst mode transfer mechanism of the MC88410 with streaming to the MC88110. Line copyback operations write (burst) the four double words from the secondary cache line back to memory.

For these examples, all addresses are assumed to be mapped as global, write-back, cacheable, and not write-protected. Also, the primary caches are assumed to be operating in the three-state model since the $\overline{\text{SHD}}$ input signal of the MC88110 is grounded. The secondary caches are assumed to be operating in the four-state model since the $\overline{\text{SHD}}$ input signal is connected to $\overline{\text{S\_SSTATO}}$ (for more information about the four-state model, see **2.4.2 Lateral Coherency**).

2.9.4.1 Example 1—Snoop Hit without Intent-to-Modify, PTAG Hit

This example illustrates the progression of events for the case of a snoop hit for a transaction without intent-to-modify. Figure 2-23 shows the caches in their initial state, with the cache lines invalidated in all locations and their contents unknown. This is the

state of the data cache after reset, assuming that the system software has invalidated all the cache lines in both the primary and secondary caches.

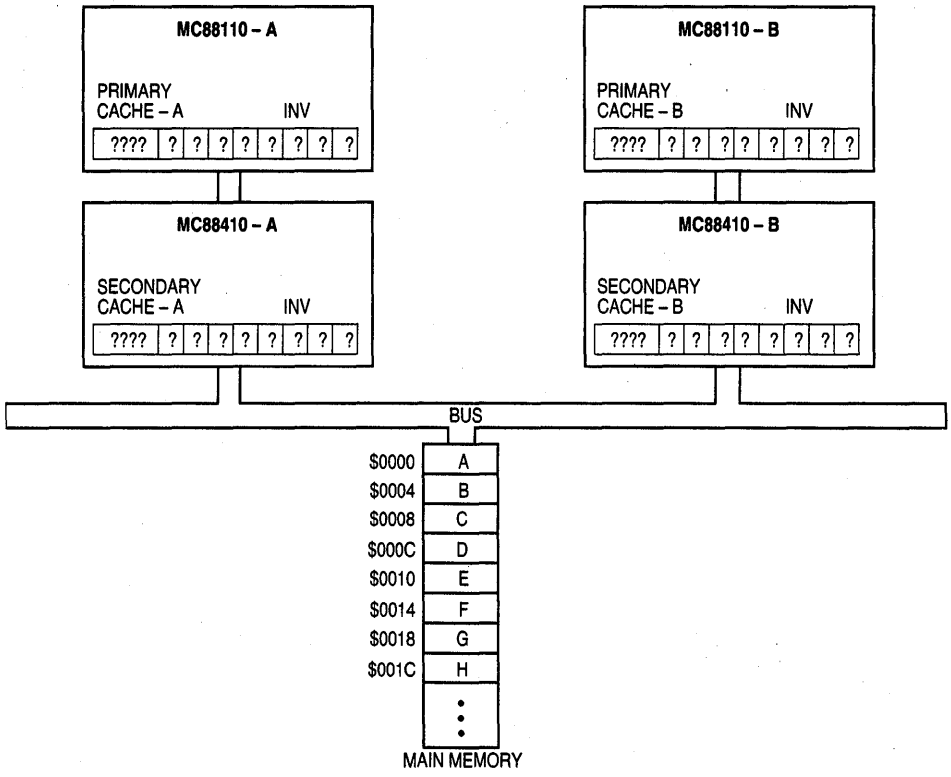


Figure 2-23. Initial State – Example 1

Figure 2-24 shows MC88110-B performing a load word operation from location $0000. There is a cache miss in both the primary and secondary caches, and MC88410-B reads a line from memory to fill the secondary cache line while streaming the data to MC88110-B. MC88410-A monitors (snoops) the bus transaction, but does not find a match in the MTAG (a miss) since the entire data cache is marked as invalid. MC88410-B updates the state of the secondary cache line to exclusive-unmodified, and MC88110-B updates the state of the primary cache line to shared-unmodified.

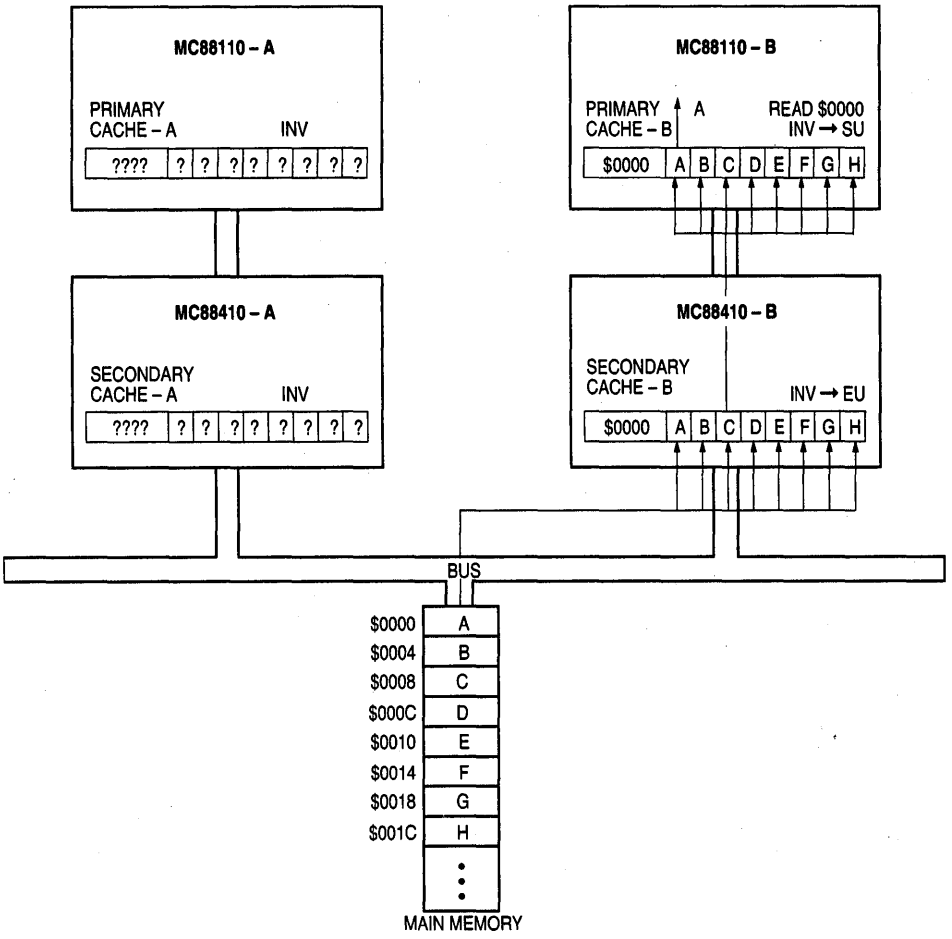


Figure 2-24. MC88110 – B Load, Data Cache Miss

Figure 2-25 shows MC88110-A reading a word from address $0008, which misses for the selected cache line. A line fill operation is performed as before, with MC88410-A reading the line from memory and streaming the data to MC88110-A. MC88410-B snoops the global transaction and finds a tag match (a snoop hit). The state of the line changes to shared-unmodified in both secondary caches and remains shared-modified in primary cache A since both nodes have a copy of the data that is unmodified with respect to memory. Note that since the transaction is not intent-to-modify, MC88110-B is not informed of the snoop hit.

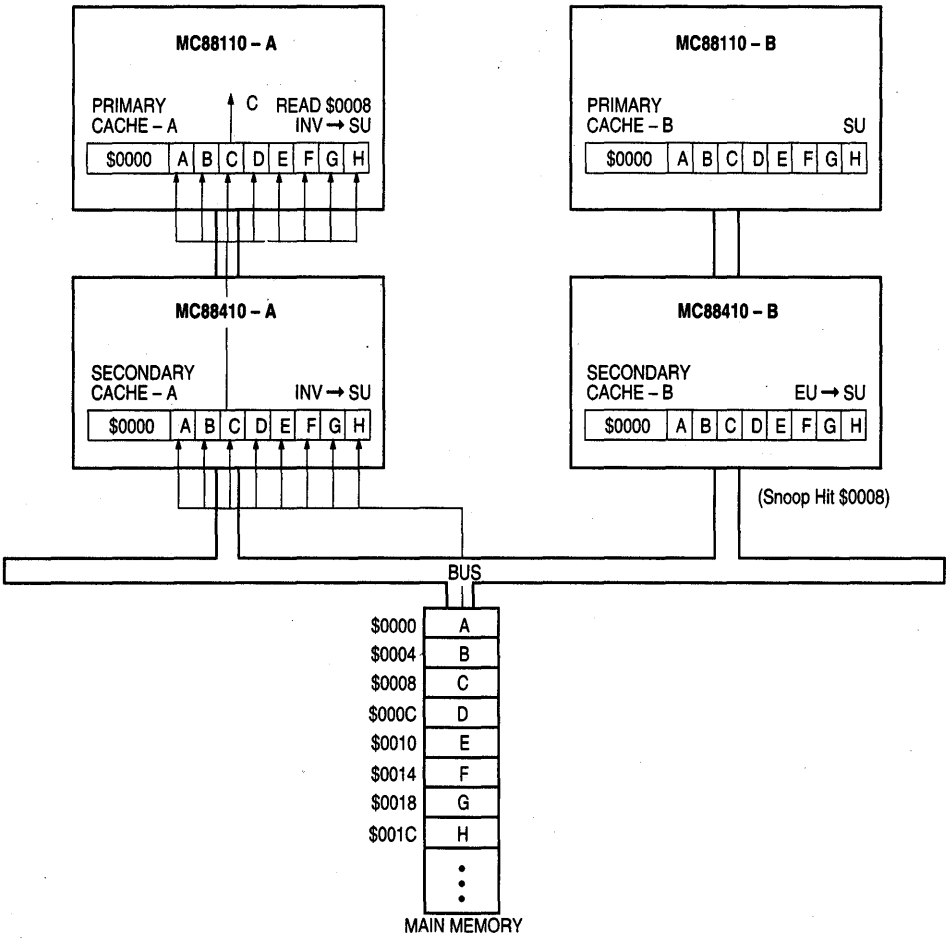


Figure 2-25. MC88110 – A Load, Data Cache Miss

Figure 2-26 shows MC88110-B performing a store operation of a word to address $0000. A cache hit occurs, and since the address was global, a processor invalidate transaction is performed. MC88410-B then performs a system bus invalidate transaction. The invalidate transaction notifies MC88410-A that its local copy of the line is no longer valid, so MC88410-A marks its cache line as invalid and performs a primary cache invalidate so that MC88110-A invalidates its line. When the invalidate transaction completes successfully, MC88410-B updates its secondary cache line and marks its line exclusive-modified. MC88110-B then updates the primary cache line with the new data and marks the line exclusive-modified.

Processor node B now has exclusive ownership of the entire line of data that is modified with respect to memory. The exclusive status guarantees processor node B that no other processor node on the bus can cache a valid copy of the line. All subsequent load and

store operations performed by MC88110-B that map to this line complete without accessing MC88410-B or the system bus. Note that although the copy of the line in MC88410-B is valid, the image of the line in the secondary cache is stale. This example shows that the exclusive-modified status in the secondary cache does not mean that the MC88410 has exclusive ownership of the line, but that the processor node has exclusive ownership of the line. This is also why the MC88410 must perform a processor invalidate cycle before copying a line that is marked exclusive-modified back to main memory.

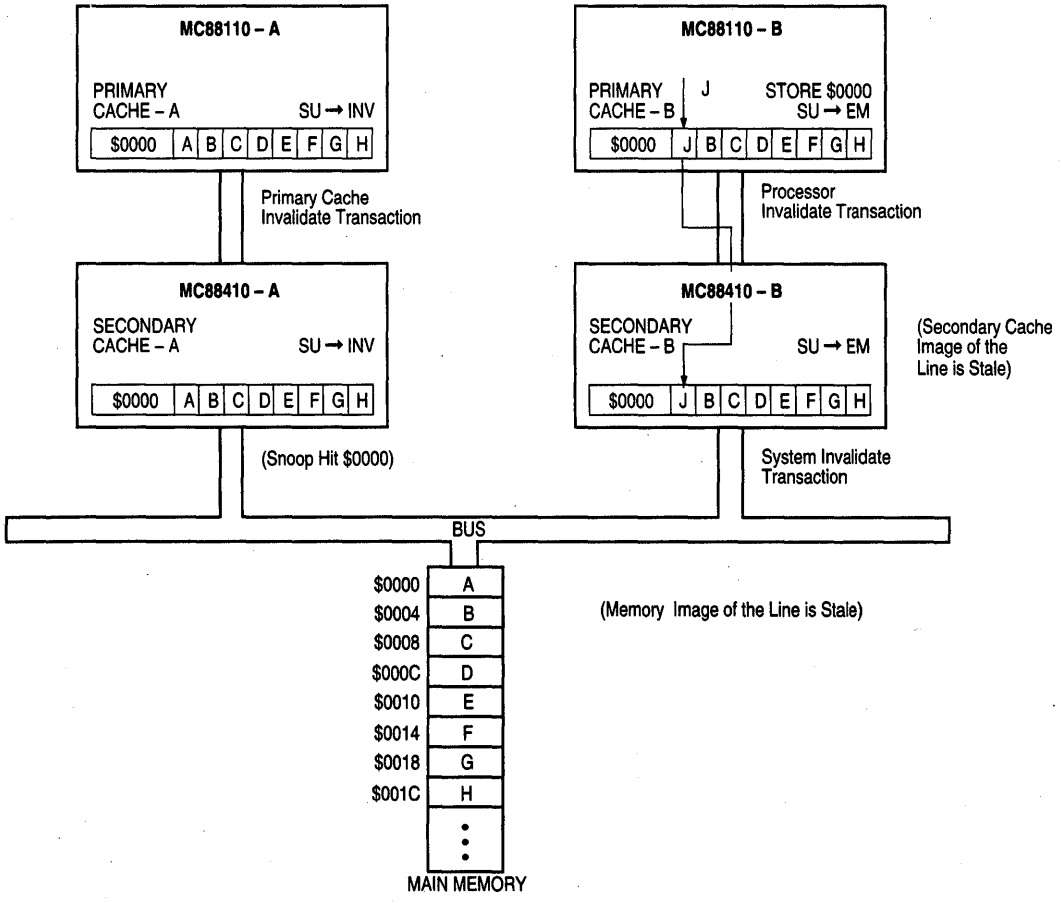


Figure 2-26. MC88110 – B Store, Data Cache Hit

Figure 2-27 shows MC88110-A attempting a load from location $0008. The transaction misses in both the primary and secondary caches because the lines in both cases are marked as invalid, which forces MC88410-A to perform a read-without-intent-to-modify transaction. MC88110-B snoops the access, recognizes that it has cached modified data requested by processor node A, and retries the line read operation by MC88410-A.

MC88410-B then arbitrates for the processor bus and performs a primary cache invalidate transaction. The snoop hardware on MC88110-B then performs a snoop copyback, and invalidates the primary cache line (because the primary cache invalidate transaction is always intent-to-modify). The MC88410 updates the secondary cache line and is ready to perform its snoop copyback to main memory.

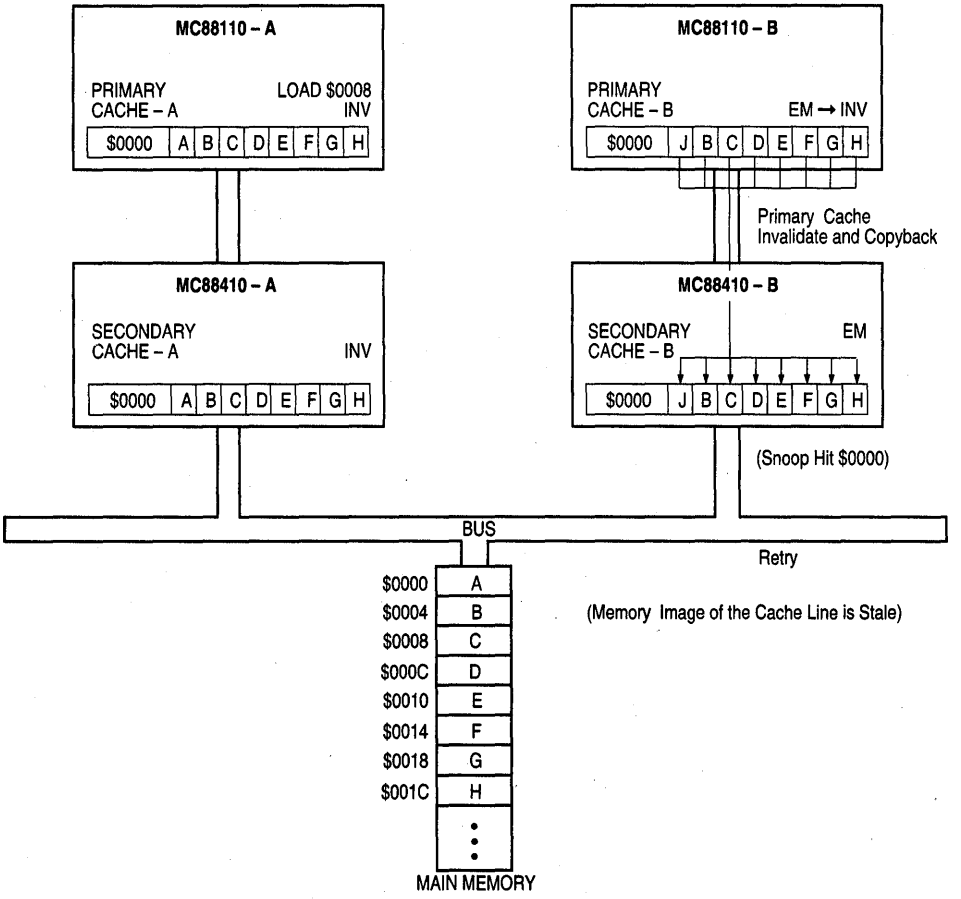


Figure 2-27. MC88110 – A Load, Cache Miss, Line Read Retried

Figure 2-28 shows MC88410-B copying back the exclusive-modified line to memory and marking the cache line as shared-unmodified (because the snooped transaction was not intent-to-modify). Since snoop copybacks are not global, no other processor nodes snoop the transaction.

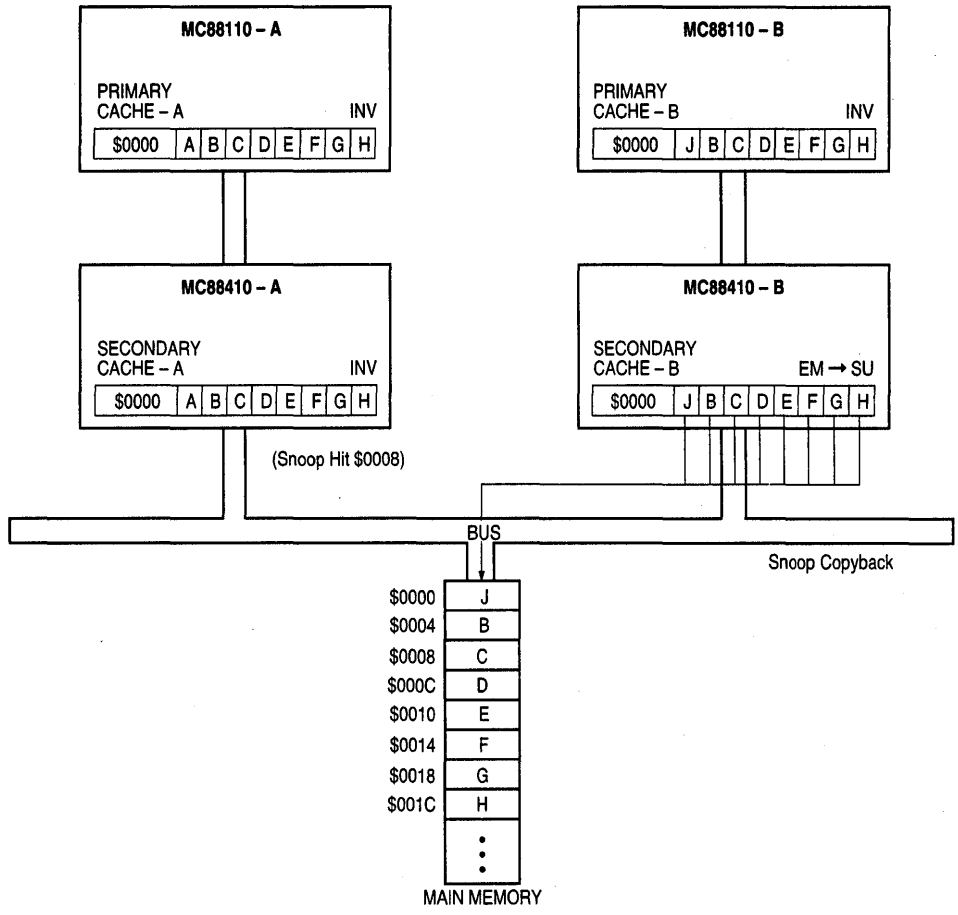


Figure 2-28. MC88110 – B Snoop Copyback

Figure 2-29 shows MC88410-A regaining control of the bus to complete the read that was previously retried by MC88410-B. MC88410-A reads the cache line and updates the secondary cache line while streaming the data to MC88110-A. The line is marked as shared-unmodified in both the primary and secondary caches.

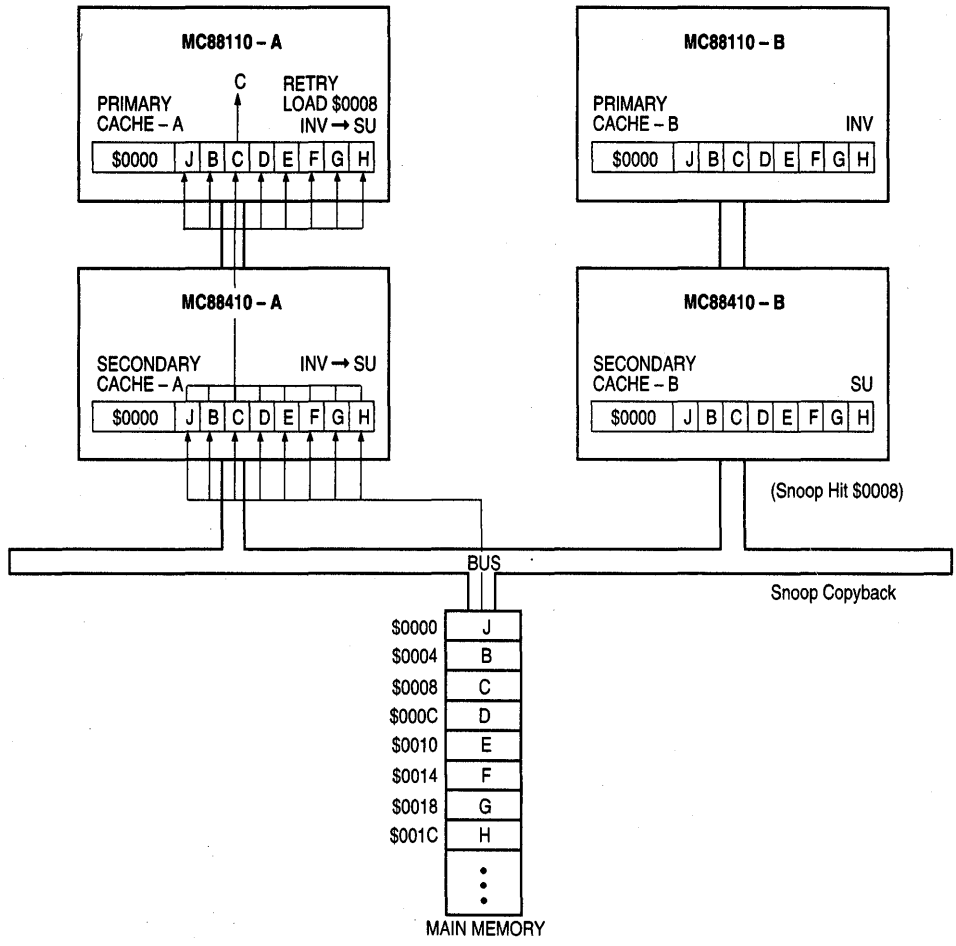


Figure 2-29. Completion of MC88110 – A Load, Cache Miss

2.9.4.2 Example 2—Snoop Hit without Intent-to-Modify, 64-Byte Secondary Cache Line

This example illustrates the progression of events for a snoop hit from a transaction without-intent-to-modify when the secondary cache is configured with a 64-byte line size. Figure 2-30 shows the caches in their initial state, with the cache lines invalidated in all locations and their contents unknown. This is the state of the data cache after reset, assuming that the system software has invalidated all the cache lines in both the primary and secondary caches.

Figure 2-31 shows MC88110-B performing a load word operation from location $0000. There is a cache miss in both the primary and secondary caches, and MC88410-B reads a 64-byte line from memory to fill the secondary cache line while streaming the first half of the line to MC88110-B. MC88410-A monitors (snoops) the bus transaction, but does not find a match in the MTAG (a miss) since the entire data cache is marked as invalid.

MC88410-B updates the state of the secondary cache line to exclusive-unmodified, and MC88110-B updates the state of the primary cache line to shared-unmodified.

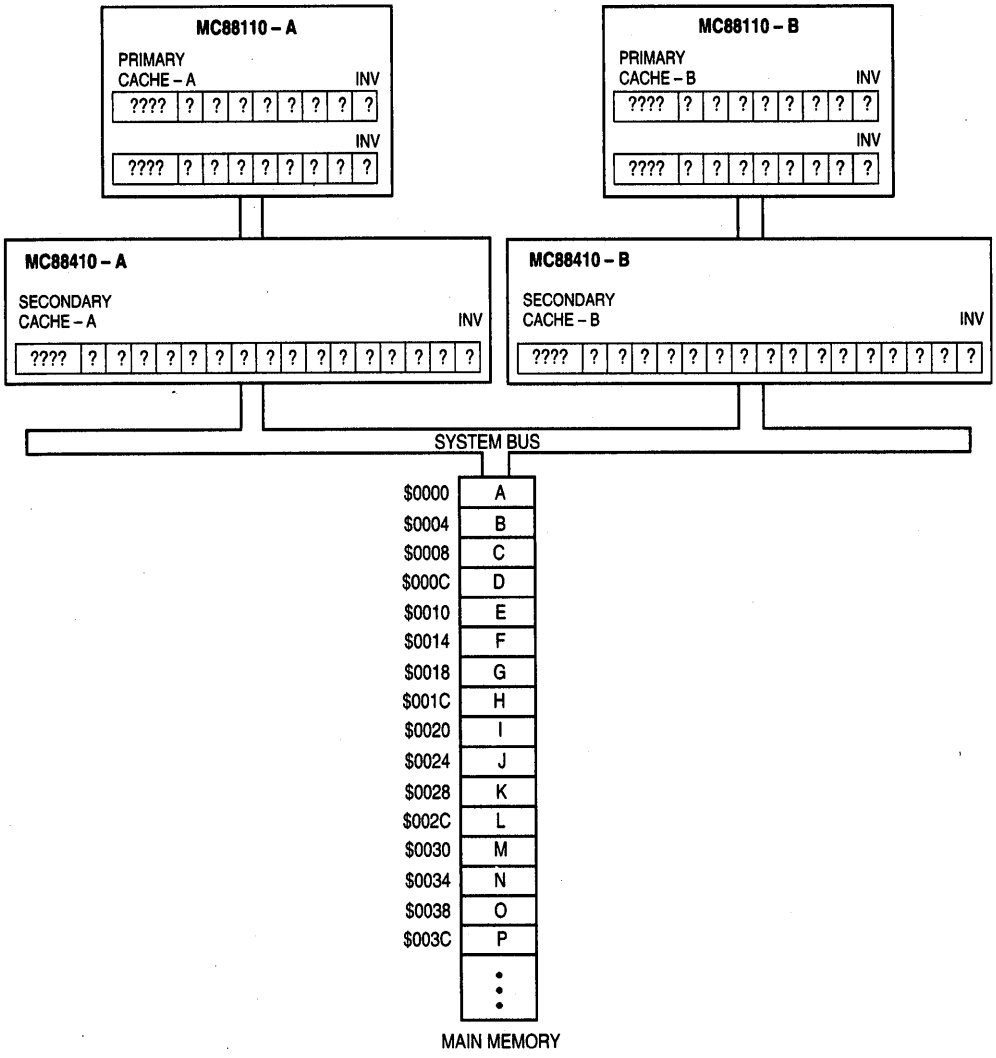


Figure 2-30. Initial State – Example 2

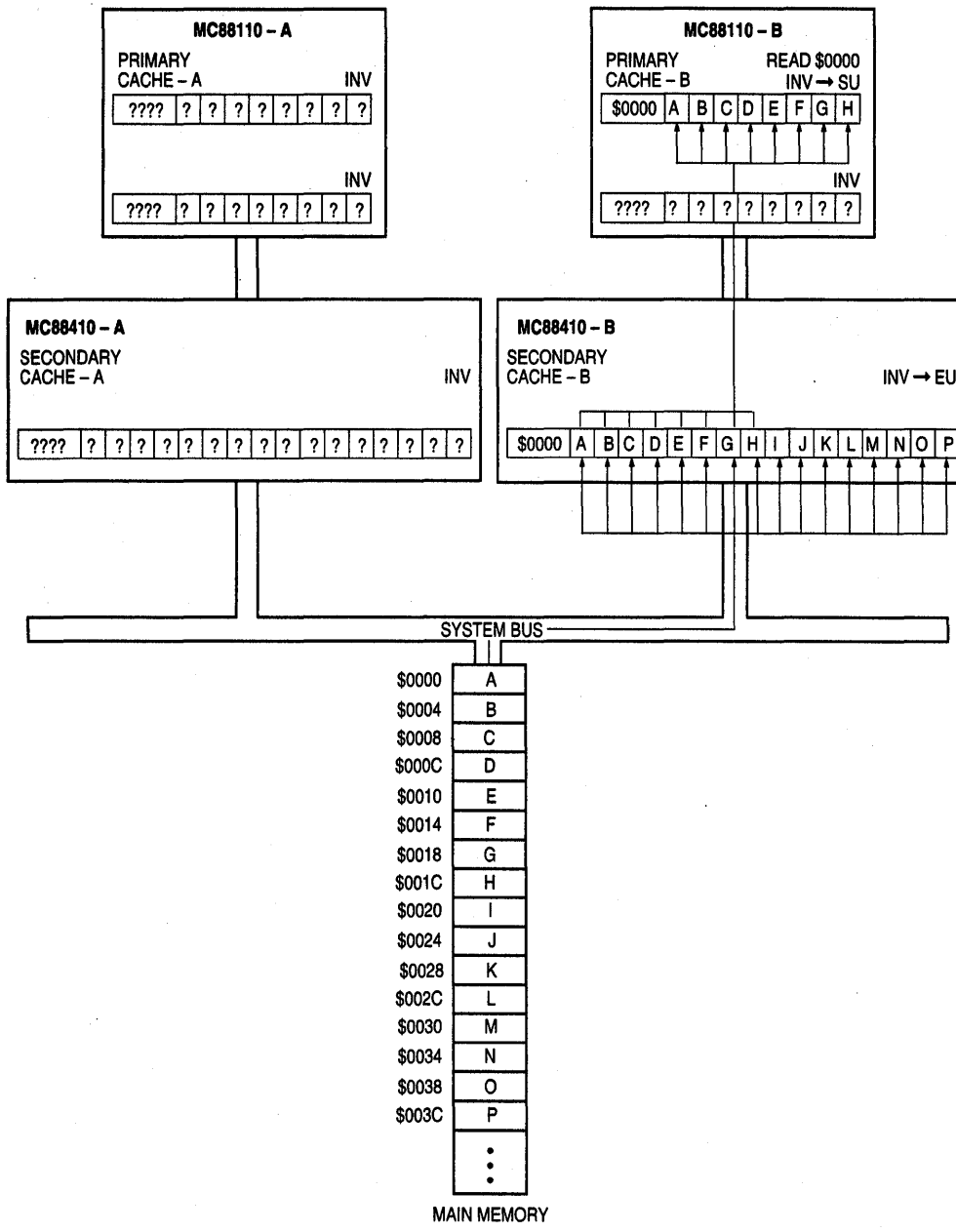


Figure 2-31. MC88110 – B Load, Data Cache Miss

Figure 2-32 shows MC88110-A reading a word from address $0028, which misses for the selected cache line. MC88410-A reads the 64-byte line from memory and streams the second half of the line to MC88110-A. MC88410-B snoops the global transaction and finds

a tag match (a snoop hit). The state of the line changes to shared-unmodified in the secondary caches and remains shared-unmodified in primary cache A since both nodes have a copy of the data that is unmodified with respect to memory. Note that since the transaction was not intent-to-modify, MC88110-B is not informed of the snoop hit.

Figure 2-33 shows MC88110-B performing a store operation on a word to address $0000. A cache hit occurs, and since the address was global, a processor invalidate transaction is performed. MC88410-B then latches the data, marks its line exclusive-modified, and performs a system bus invalidate transaction. The invalidate transaction notifies MC88410-A that its local copy of the line is no longer valid, so MC88410-A marks its cache line as invalid.

Since MC88410-A has a 64-byte line size, it must check the PTAG twice, once for each half of the line. The second half of the cache line results in a PTAG hit, and MC88410-A performs a primary cache invalidate transaction so that MC88110-A invalidates its line. Note that in this case, the primary cache line gets invalidated even though it is not stale. MC88110-B then updates the line with the new data and marks the line exclusive-modified.

Processor node B now has exclusive ownership of the entire line of data that is modified with respect to memory. The exclusive status guarantees processor node B that no other processor on the bus can cache a valid copy of the line. All subsequent load and store operations performed by MC88110-B that map to this line complete without accessing MC88410-B or the system bus.

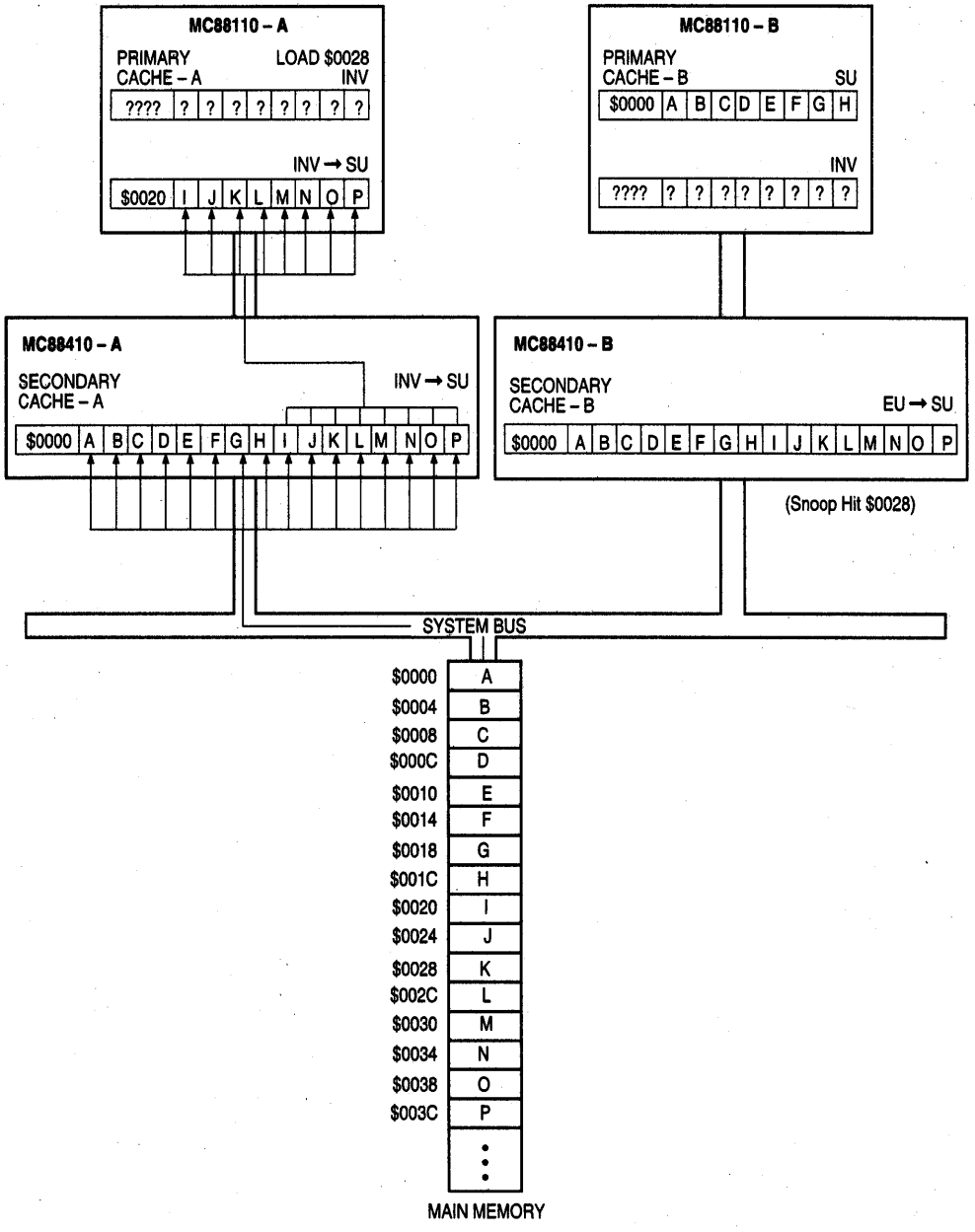


Figure 2-32. MC88110 – A Load, Data Cache Miss

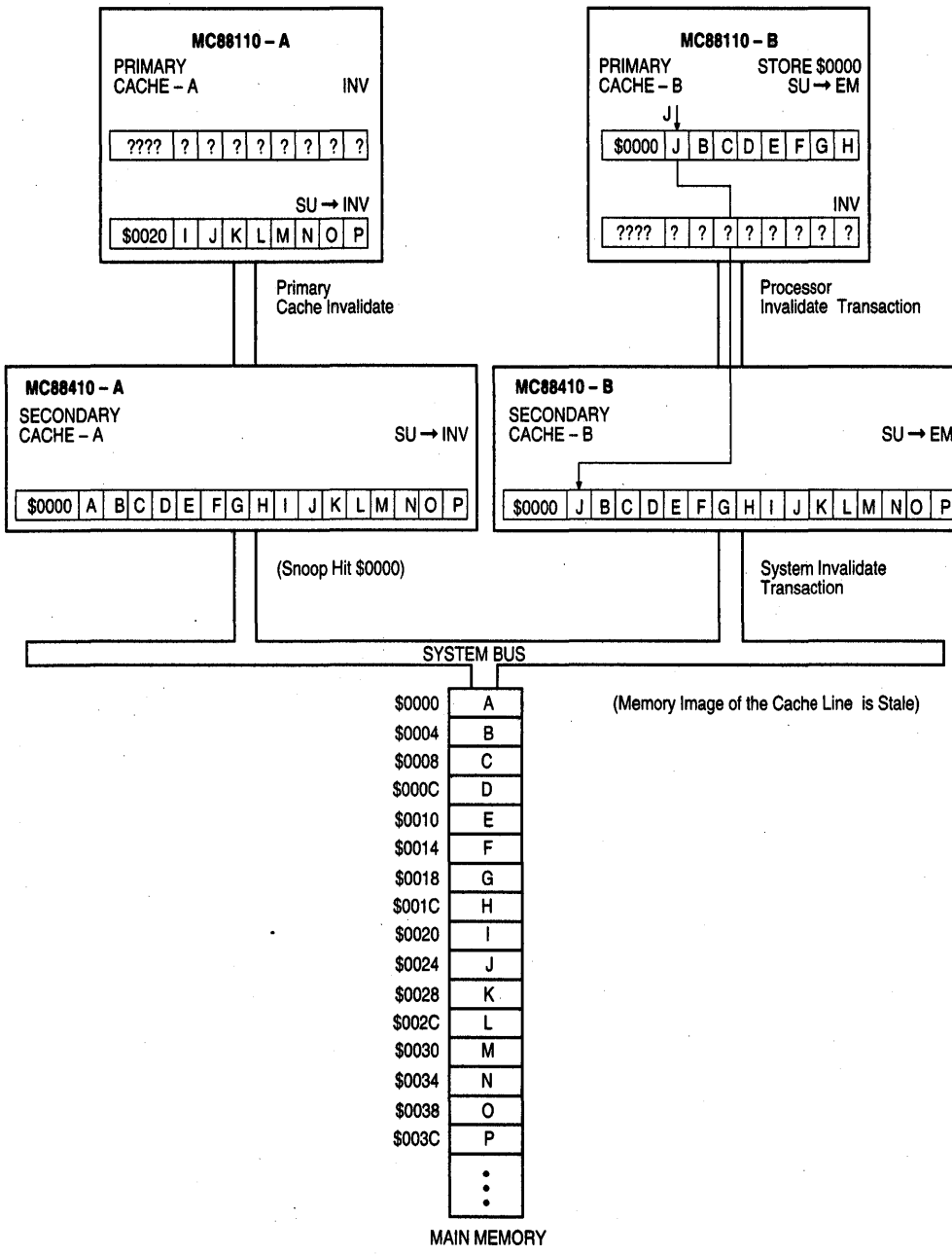


Figure 2-33. MC88110–B Store, Data Cache Hit

Figure 2-34 shows MC88110-A attempting a load from location $0028. The transaction misses in both the primary and secondary caches because the lines in both cases are marked as invalid, which forces MC88410-A to perform a read-without-intent-to-modify

transaction. MC88110-B snoops the access, recognizes that it has cached modified data requested by processor node A, and retries the line read operation by MC88410-A. MC88410-B then checks the PTAG for both halves of the line, finds a PTAG hit for the first half of the line, arbitrates for the processor bus, and performs a primary cache invalidate. The snoop hardware on MC88110-B then performs a snoop copyback, and invalidates the primary cache line (because the primary cache invalidate transaction is always intent-to-modify). The MC88410 updates the secondary cache line and is ready to perform its snoop copyback to main memory.

Figure 2-35 shows MC88410-B writing back the exclusive-modified 64-byte line to memory and marking the cache line as shared-unmodified (because the snooped transaction was not intent-to-modify). Since snoop copybacks are not global, no other processor nodes snoop the transaction.

Figure 2-36 shows MC88410-A regaining control of the bus to complete the read that was previously retried by MC88410-B. MC88410-A reads the cache line and updates the secondary cache line while streaming the second half of the line to MC88110-A. The line is marked as shared-unmodified in both the primary and secondary caches.

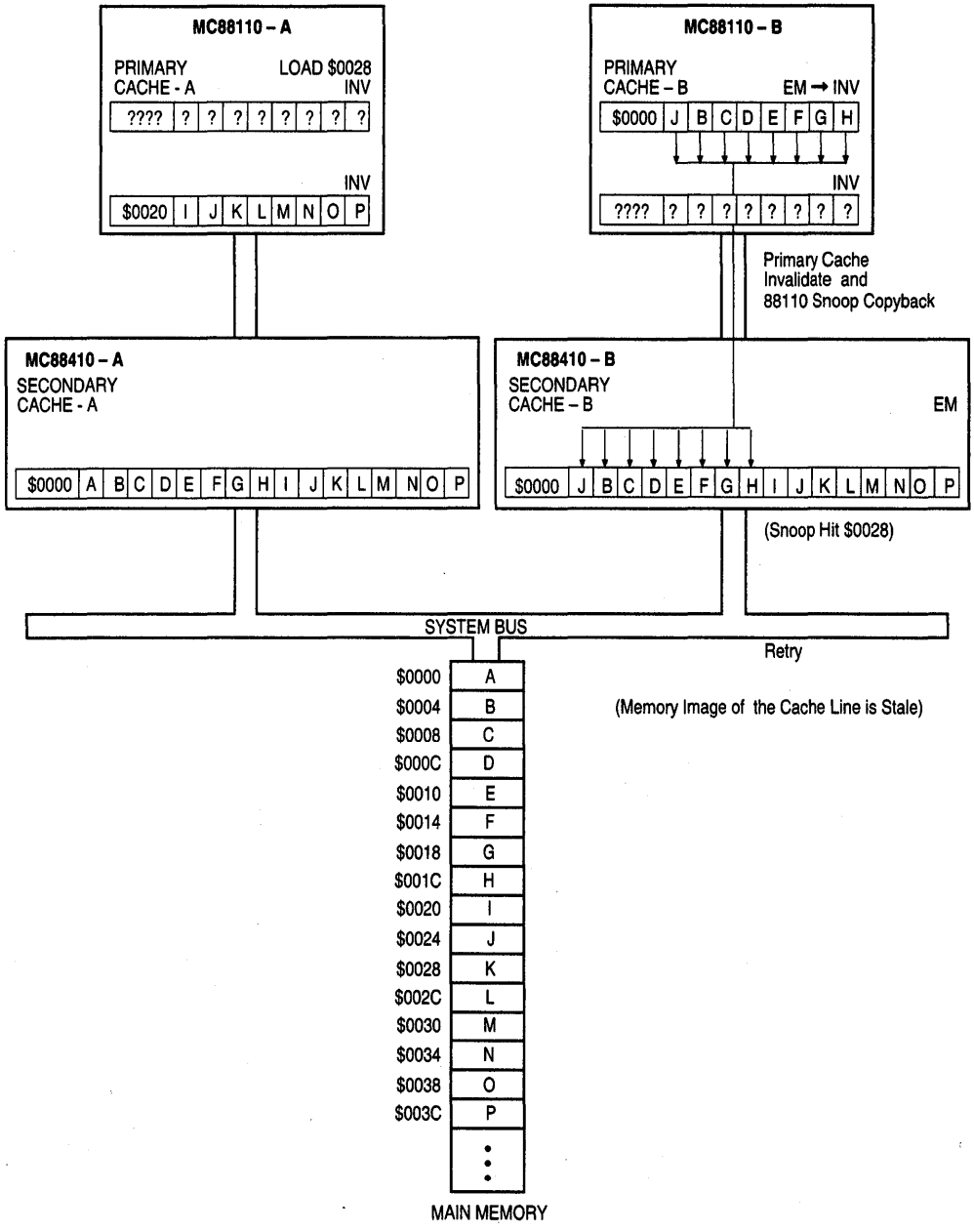


Figure 2-34. MC88110–A Load, Data Cache Miss, Line Read Retried

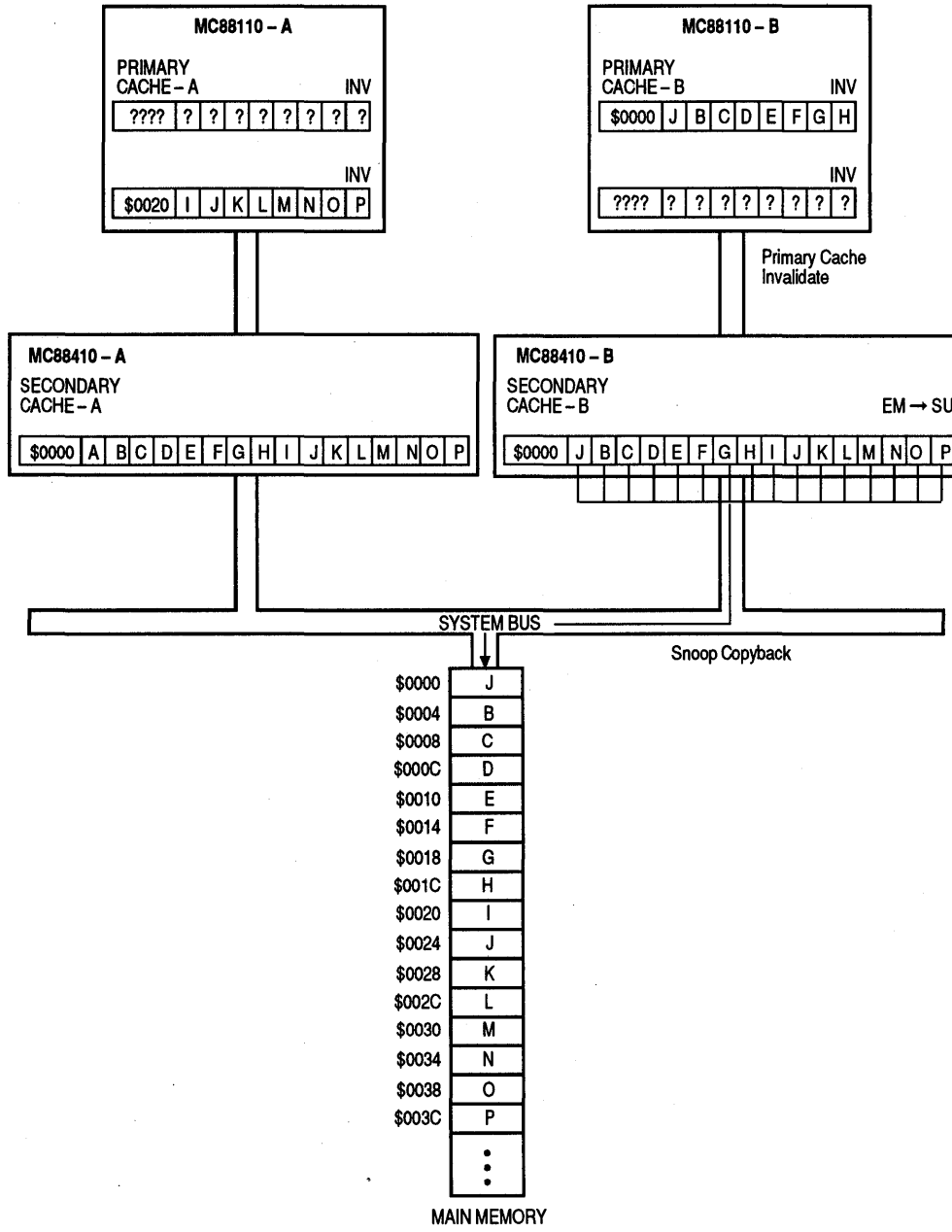


Figure 2-35. MC88110–A Snoop Copyback

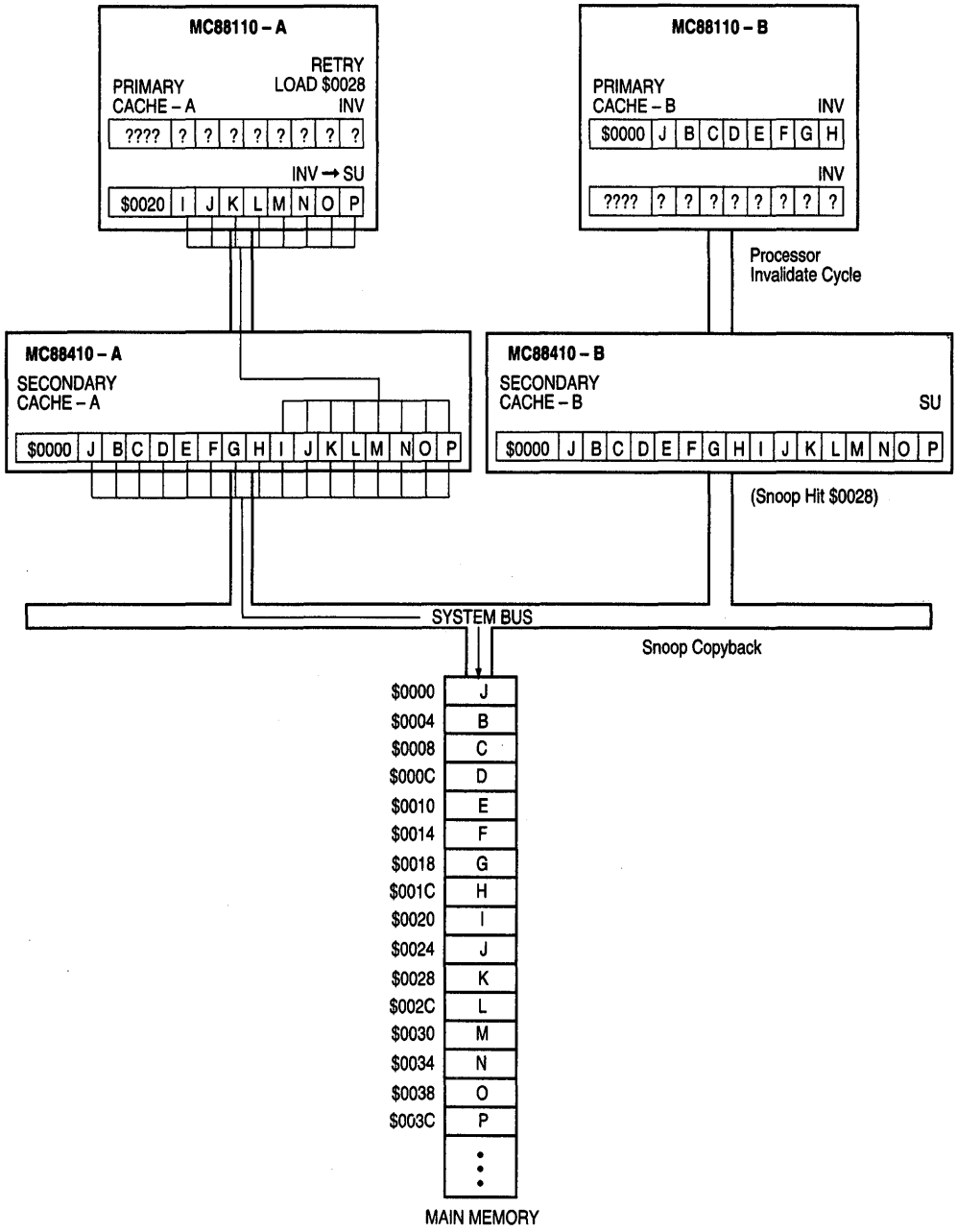


Figure 2-36. Completion of MC88110–A Load, Cache Miss

2.9.4.3 Example 3—Simultaneous Write Misses with Secondary Cache Hits

This example illustrates the progression of events for the case of two MC88110s simultaneously attempting a write to a line which is in the secondary cache only. Figure 2-37 shows the caches in their initial state for this example. In both processor nodes, the primary cache lines are invalid, but the secondary cache lines are shared-unmodified.

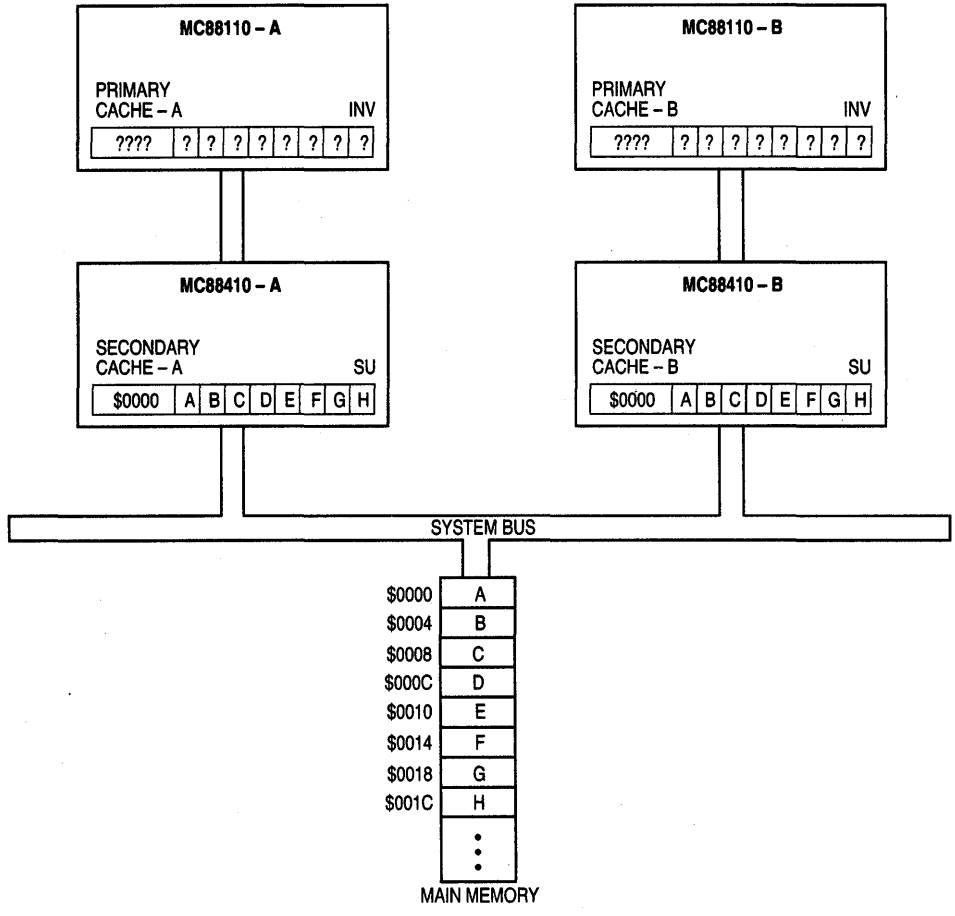


Figure 2-37. Initial State - Example 3

Figure 2-38 shows both MC88110s attempting to perform a write to the same cache line. Since neither primary cache has the data, both processors initiate a read-with-intent-to-modify cycle. Both MC88410s then have secondary cache hits, but since the cache lines are shared-unmodified and the transaction is intent-to-modify, both MC88410s request the system bus for a system invalidate transaction. The arbitration circuitry must grant the bus to only one of the MC88410s; in this example, the arbiter grants the bus to MC88410-A. MC88410-B then has a snoop hit and invalidates its cache line. Once the

system invalidate transaction is complete, MC88410-A writes the line to MC88110-A, and MC88110-A completes its store instruction. At the end of the transaction, the line is marked exclusive-modified in both the primary and secondary caches.

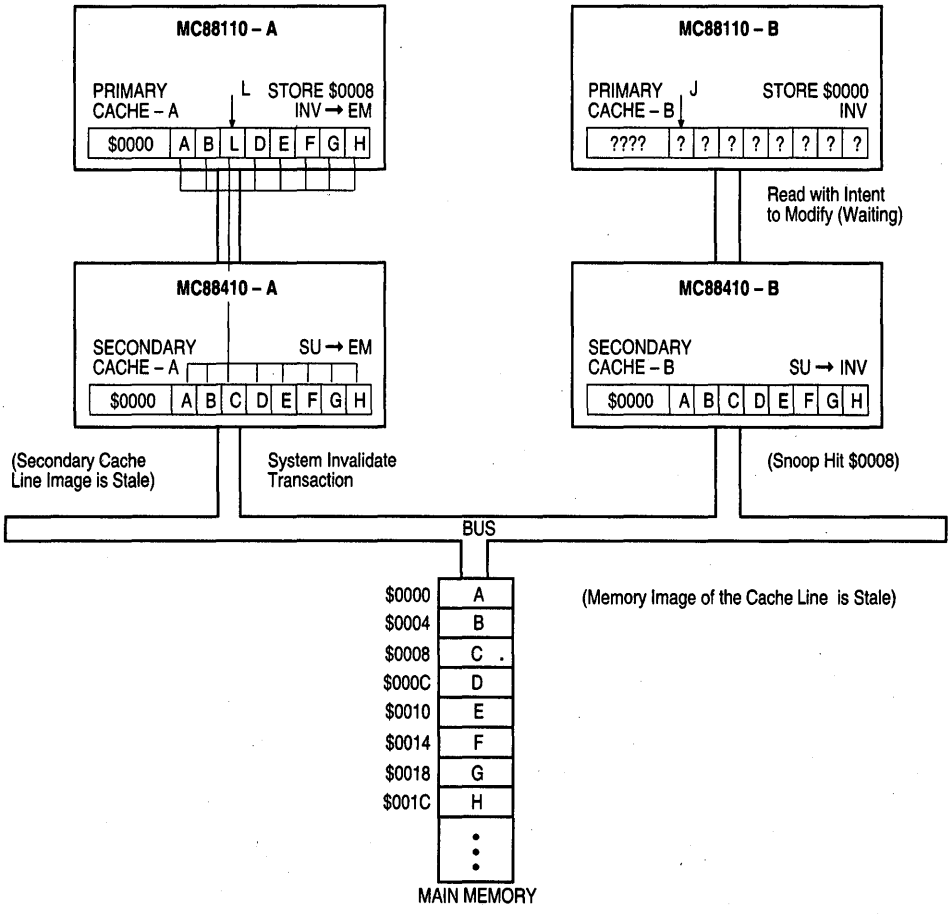


Figure 2-38. Simultaneous MC88110 Stores, Data Cache Miss, MC88410–A System Invalidate

Since MC88410-B is forced to invalidate its cache line while it is waiting for the arbiter to grant it the bus, MC88410-B re-evaluates the transaction and the processor transaction misses in the secondary cache so it initiates a read-with-intent-to-modify transaction instead (shown in Figure 2-39). MC88110-A snoops the access, recognizes that it has cached modified data requested by processor node B, and retries the line read operation by MC88410-B. MC88410-A then arbitrates for the processor bus and performs a primary cache invalidate transaction. The snoop hardware on MC88110-A then performs a snoop copyback, and invalidates the primary cache line. MC88410-A updates the secondary cache line and is ready to perform its snoop copyback to main memory.

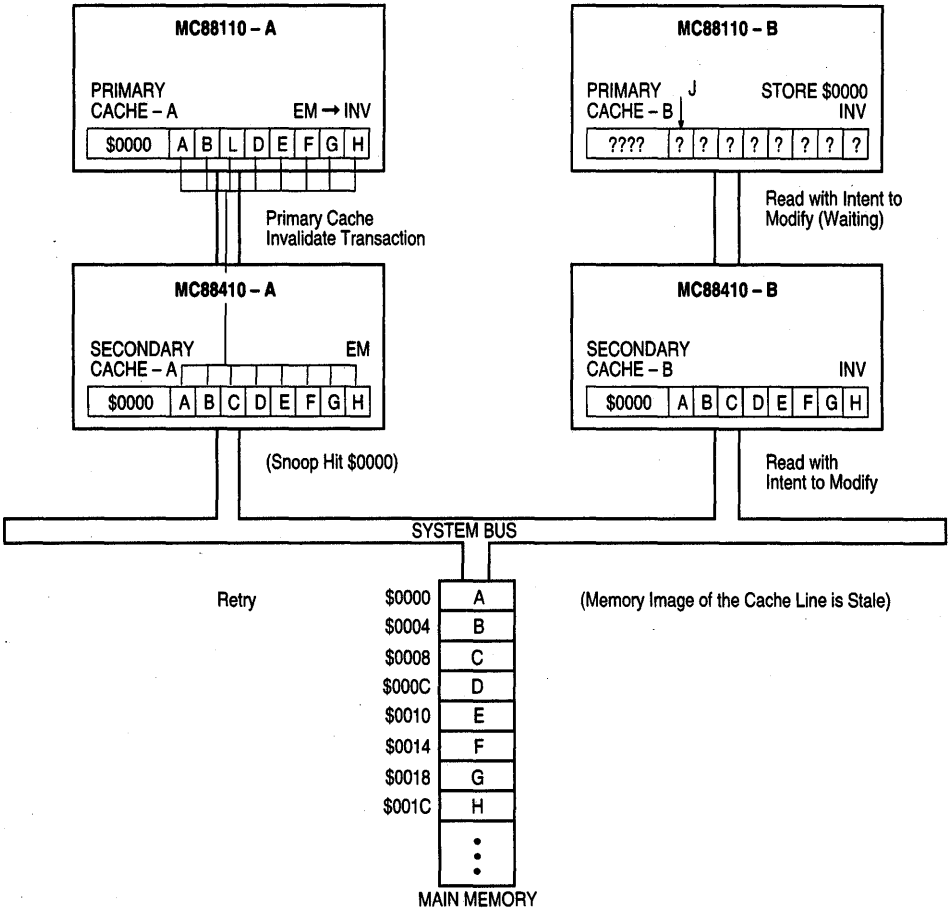


Figure 2-39. MC88410–B Read-with-Intent-to-Modify, Retried

Figure 2-40 shows MC88410-A writing the exclusive-modified line back to memory. Note that in this case the snoop copyback is due to an intent-to-modify transaction, so the secondary cache line is marked invalid. Since snoop copybacks are not global, no other processor nodes snoop the transaction.

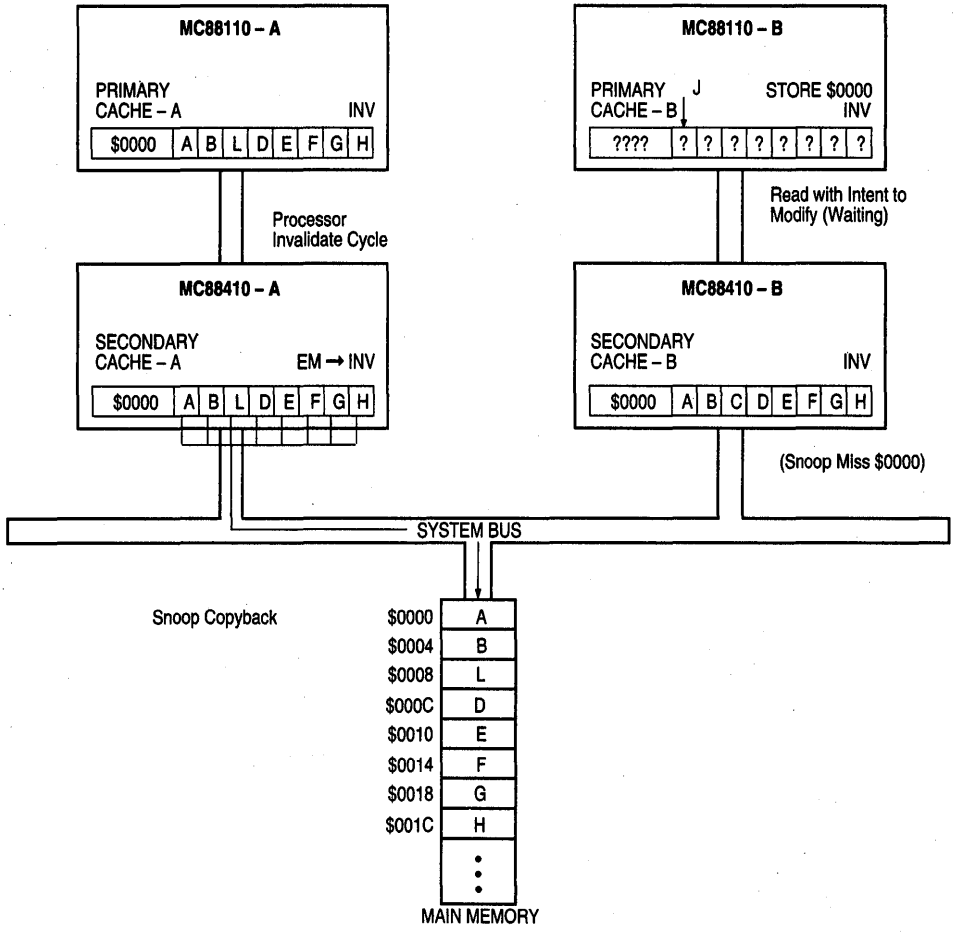


Figure 2-40. MC88410–A Snoop Copyback

Figure 2-41 shows MC88410-B regaining control of the bus to complete the read that was previously retried by MC88410-A. MC88410-B reads the cache line and updates the secondary cache line while streaming the data to MC88110-B. MC88110-B then updates the primary cache line and completes the store. Since the initial transaction from the MC88110 was intent-to-modify, both the primary and secondary cache lines are marked exclusive-modified.

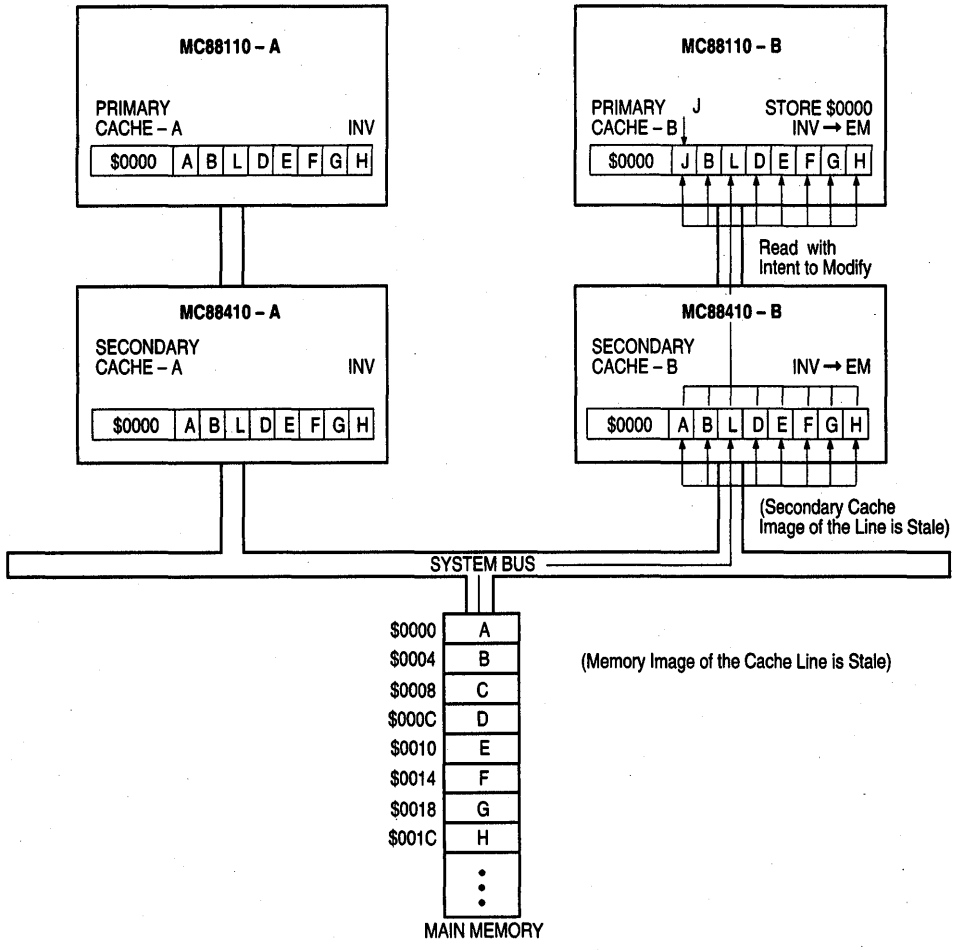


Figure 2-41. Completion of MC88110–A Load, Cache Miss

2.9.4.4 Example 4—Simultaneous First Write Hits with Secondary Cache Hits

This example illustrates the progression of events for the case of two MC88110s simultaneously attempting a write to a line that is in both the primary and the secondary caches. Figure 2-42 shows the caches in their initial state for this example. In both processor nodes, the primary and secondary cache lines are shared-unmodified.

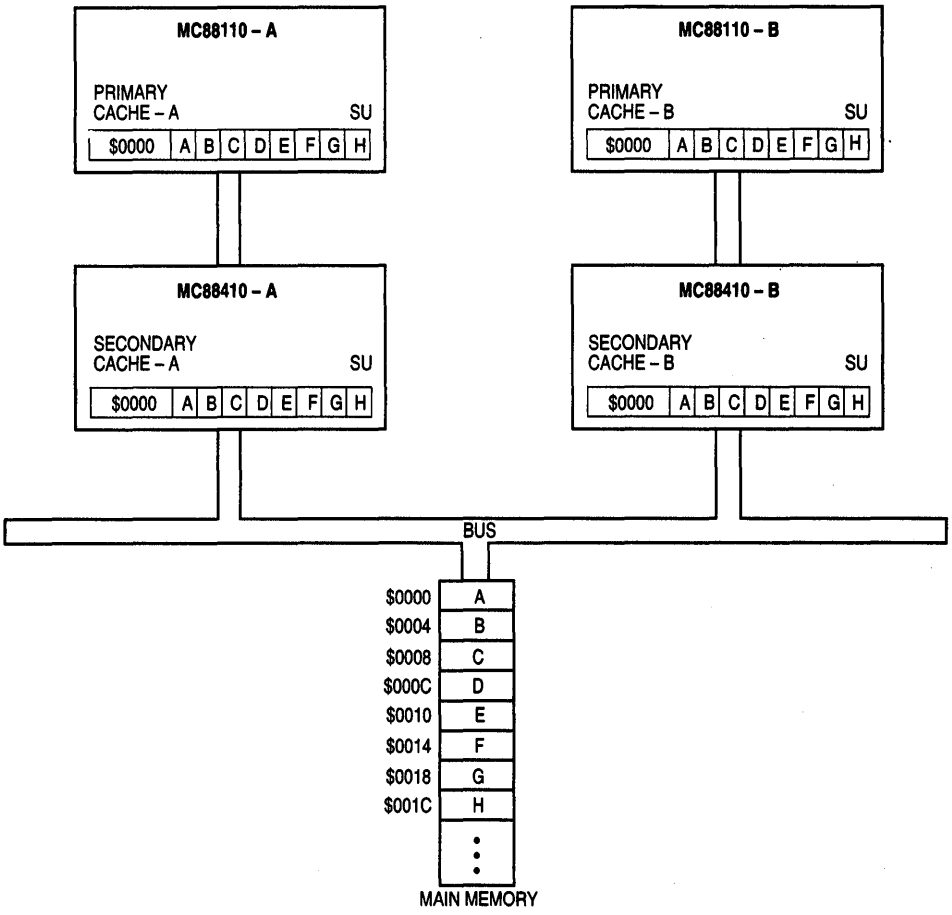


Figure 2-42. Initial State – Example 4

Figure 2-43 shows both MC88110s attempting to perform a write to the same cache line. Since both primary caches have the data, both processors initiate a processor invalidate transaction. Both MC88410s then have secondary cache hits, but since the cache lines are shared-unmodified and the transaction is intent-to-modify, both MC88410s request the system bus for a system invalidate transaction. The arbitration circuitry must grant the bus to only one of the MC88410s; in this example, the arbiter grants the bus to MC88410-A. MC88410-B then has a snoop hit and invalidates its cache line. Since the cache line is also in the primary cache (PTAG hit), the MC88410 retries the transaction from the MC88110 so that the MC88410 can perform a primary cache invalidate transaction. Once the system invalidate transaction from MC88410-A is complete, MC88410-A updates its secondary cache line, and MC88110-A completes its store instruction. At the end of the transaction, the line is marked exclusive-modified in both the primary and secondary caches of processor node A.

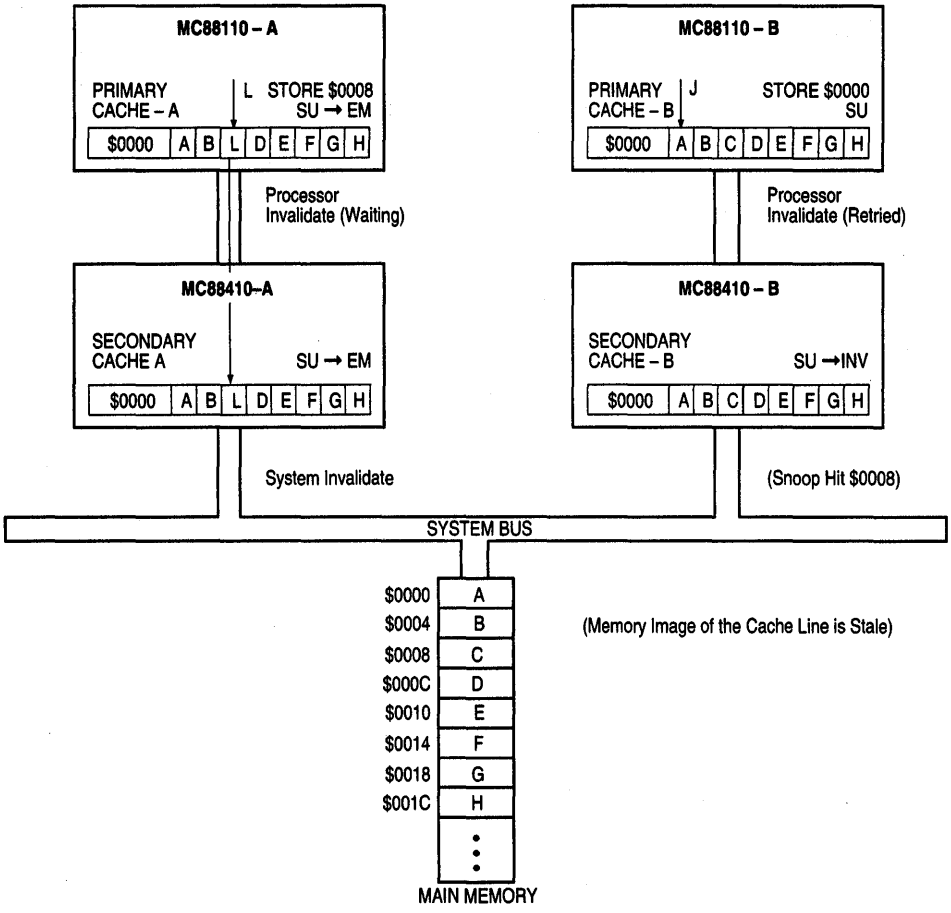


Figure 2-43. Simultaneous MC88110 Stores, Data Cache Miss, MC88410–A System Invalidate

Figure 2-44 shows MC88410-B performing the primary cache invalidate transaction. Since the primary cache line is shared-unmodified, MC88110-B marks the cache line invalid without needing to perform a snoop copyback.

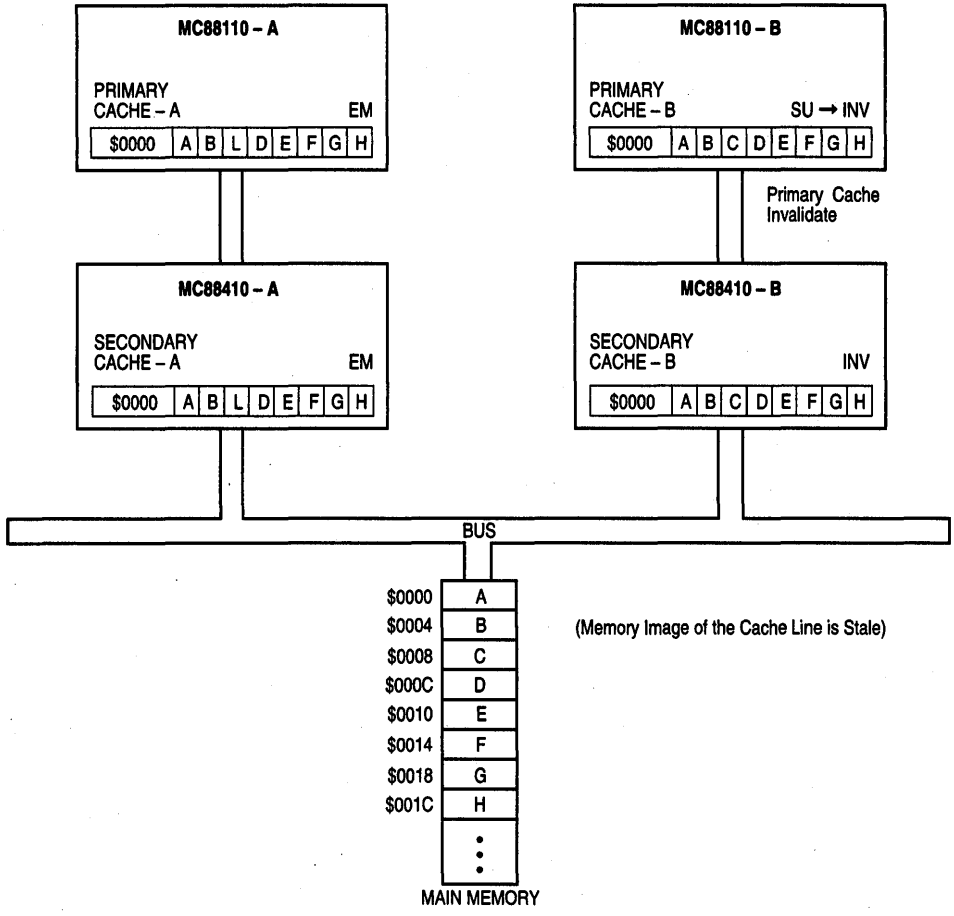


Figure 2-44. MC88410–B Processor Invalidate Transaction

After the completion of the primary cache invalidate transaction from MC88410-B, MC88110-B attempts the store that was retried by MC88410-B. Since the line was invalidated due to the snoop hit, there is now a miss in the primary and secondary caches and MC88410-B initiates a read-with-intent-to-modify transaction shown in Figure 2-45. MC88110-A snoops the access, recognizes that it has cached modified data requested by processor node B, and retries the line read operation by MC88410-B. MC88410-A then arbitrates for the processor bus and performs a primary cache invalidate transaction. The snoop hardware on MC88110-A then performs a snoop copyback, and invalidates the primary cache line. The MC88410 updates the secondary cache line and is ready to perform its snoop copyback to main memory.

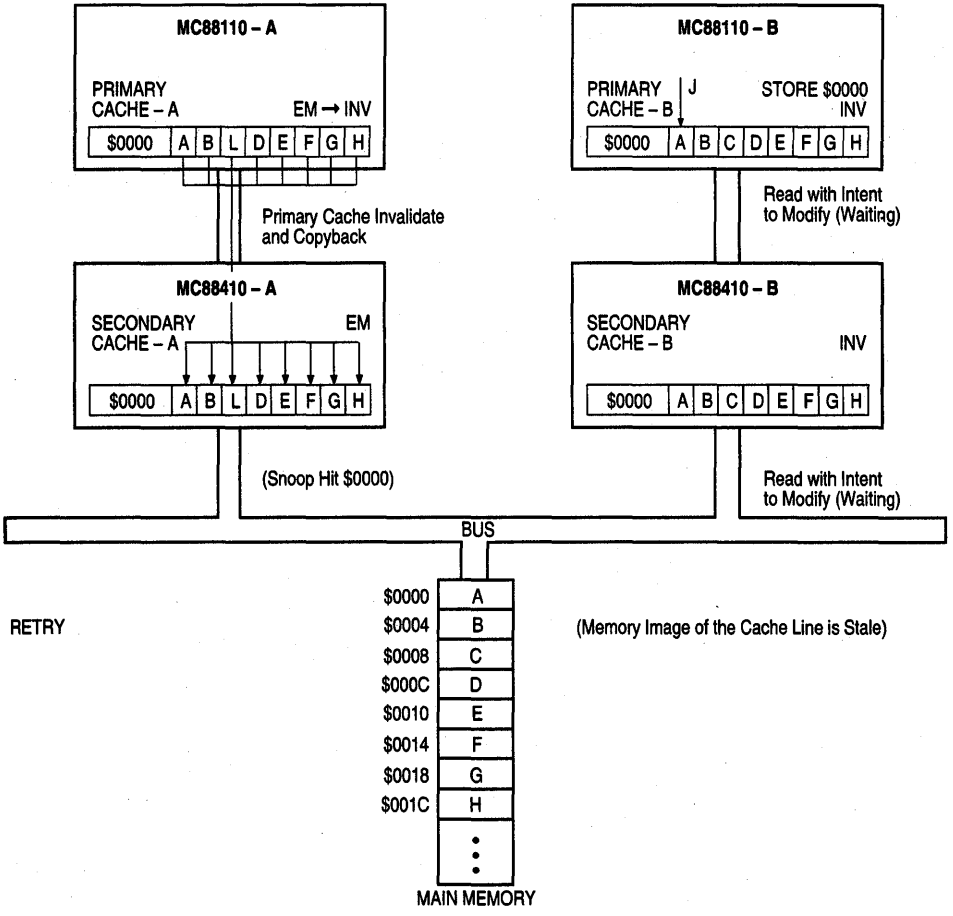


Figure 2-45. MC88410–B Read-with-Intent-to-Modify, Retried

Figure 2-46 shows MC88410-A writing back the exclusive-modified line to memory. Note that in this case the snoop copyback is due to an intent-to-modify transaction, so the secondary cache line is marked invalid. Since snoop copybacks are not global, no other processor nodes snoop the transaction.

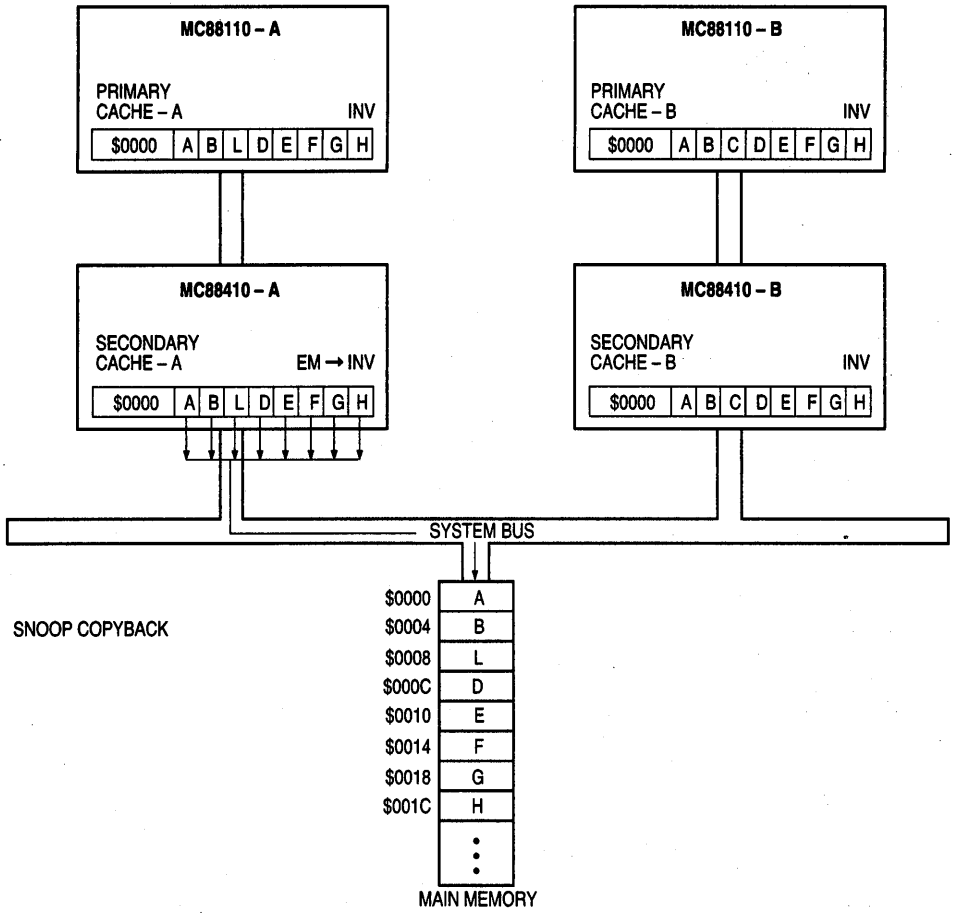


Figure 2-46. MC88410–A Snoop Copyback

Figure 2-47 shows MC88410-B regaining control of the bus to complete the store that was previously retried by MC88410-A. MC88410-B reads the cache line and updates the secondary cache line while streaming the data to MC88110-B. MC88110-B then updates the primary cache line and completes the store. Since the initial transaction from MC88110-B was intent-to-modify, both the primary and secondary cache lines are marked exclusive-modified.

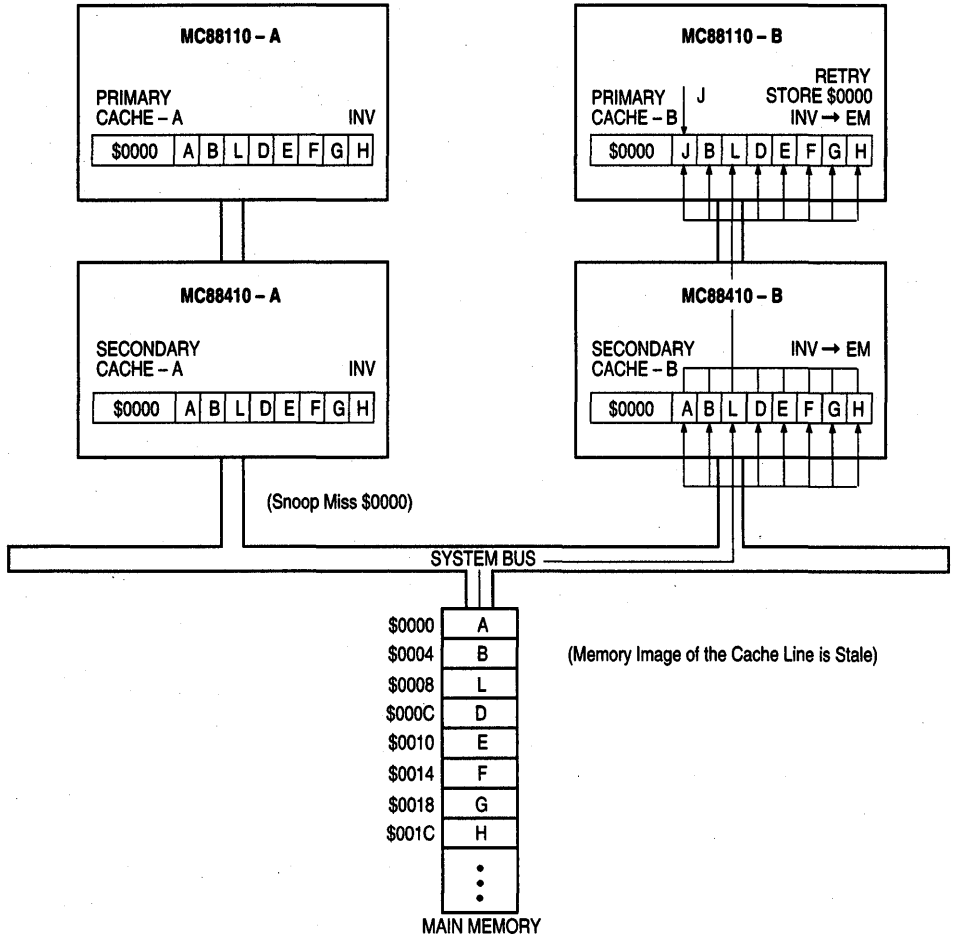


Figure 2-47. Completion of MC88110–A Load, Cache Miss

SECTION 3
SIGNAL DESCRIPTION

This section describes the MC88410 input and output signals in their functional groups. Figure 3-1 shows the functional organization of the MC88410 bus signals. The functional groups are the following:

- Processor interface signals
- System interface signals
- RAM interface signals
- System configuration signals
- Test signals.

The MC88410 processor interface signals correspond with the MC88110 input and output signals (see Section 11 in the *MC88110 Second Generation RISC Microprocessor User's Manual*) with the omission of the data and data bus arbitration signals and status signals. The MC88410 system interface is similar to the MC88110 system interface with the addition of a late shared signal and an additional snoop status signal. Both the processor and system interface include 32 address I/O signals. Table 3-1 provides the mnemonic, type, and state out of reset for the MC88410 processor interface and system interface signals.

The RAM interface includes 17 address I/O signals and control signals for the MCM62110 secondary cache RAM array. The system configuration signals are set during reset and then used for tag monitoring during MC88410 operation. Finally, the test signals are included for system diagnostics. Table 3-2 provides the mnemonic, type, and state out of reset for the RAM interface, system configuration, and test signals.

NOTE

The terms assert and negate are used extensively in this manual to avoid confusion between active-high and active-low signals. **Assert** or **assertion** indicates that a signal is active or true, regardless of whether the signal is active high or active low. **Negate** or **negation** indicates that the signal is inactive or false.

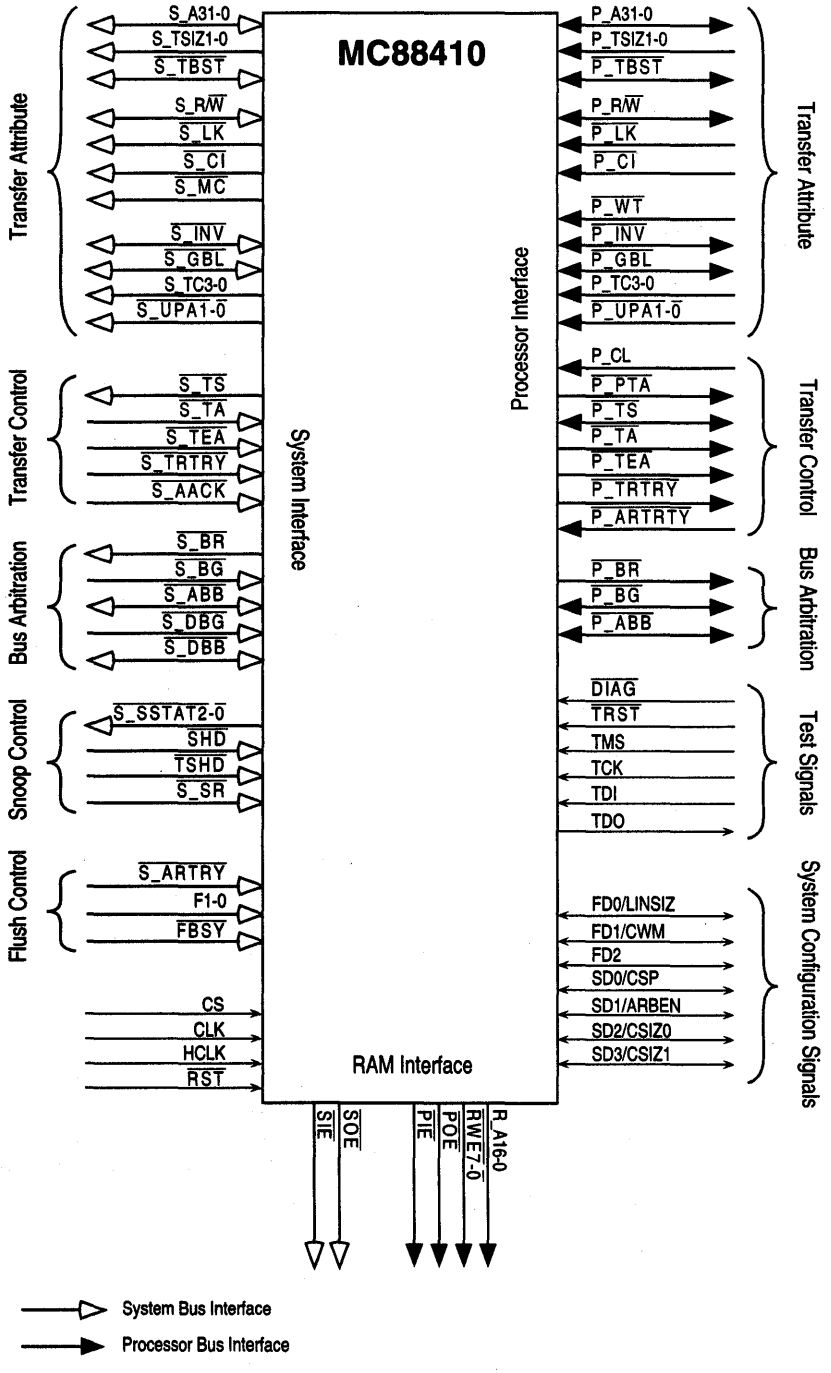


Figure 3-1. MC88410 Signals

Table 3-1. Transaction Signal Summary

Function	Processor Interface Signals			System Interface Signals		
	Mnemonic	Type	Reset	Mnemonic	Type	Reset
Address Bus	P_A31–P_A0	I/O	Three-state	S_A31–S_A0	I/O	Three-state
Transfer Attribute Signals						
Read/Write	P_R/$\overline{W}$	I/O	Three-state	S_R/$\overline{W}$	I/O	Three-state
Lock	$\overline{P\_LK}$	Input	Three-state	$\overline{S\_LK}$	Output	Three-state
Cache-inhibit	$\overline{P\_CI}$	Input	Three-state	$\overline{S\_CI}$	Output	Three-state
Write-through	$\overline{P\_WT}$	Input	Three-state	Not available on system bus		
User page attributes	$\overline{P\_UPA1–0}$	Input	Three-state	$\overline{S\_UPA1–0}$	Output	Three-state
Transfer burst	$\overline{P\_TBST}$	I/O	Three-state	$\overline{S\_TBST}$	I/O	Three-state
Transfer size	P_TSIZ1–0	Input	Three-state	S_TSIZ1–0	Output	Three-state
Transfer code	P_TC3-0	Input	Three-state	S_TC3–0	Output	Three-state
Invalidate	$\overline{P\_INV}$	I/O	Three-state	$\overline{S\_INV}$	I/O	Three-state
Global	$\overline{P\_GBL}$	I/O	Three-state	$\overline{S\_GBL}$	I/O	Three-state
Cache line	P_CL	Input	Three-state	Not available on system bus		
Memory cycle	Not available on processor bus			$\overline{S\_MC}$	Output	Three-state
Transfer Control Signals						
Transfer start	$\overline{P\_TS}$	I/O	Three-state	$\overline{S\_TS}$	Output	Three-state
Transfer acknowledge	$\overline{P\_TA}$	Output	Three-state	$\overline{S\_TA}$	Input	Three-state
Pretransfer acknowledge	$\overline{P\_PTA}$	Output	Three-state	Not available on system bus		
Transfer error acknowledge	$\overline{P\_TEA}$	Output	Three-state	$\overline{S\_TEA}$	Input	—
Transfer retry	$\overline{P\_TRTRY}$	Output	Three-state	$\overline{S\_TRTRY}$	Input	—
Address acknowledge	Not available on processor bus			$\overline{S\_AACK}$	Input	—
Snoop Control Signals						
Address retry	$\overline{P\_ARTRY}$	Input	—	$\overline{S\_ARTRY}$	Input	—
Snoop request	Not available on processor bus			$\overline{S\_SR}$	Input	—
Snoop status	Not available on processor bus			$\overline{S\_SSTAT2–0}$	Output	Three-state
Shared	Not available on processor bus			$\overline{SHD}$	Input	—
Transfer shared	Not available on processor bus			$\overline{TSHD}$	Input	—
Bus Arbitration Signals						
Bus request	$\overline{P\_BR}$	Output	Negated	$\overline{S\_BR}$	Output	Negated
Bus grant	$\overline{P\_BG}$	I/O	Negated	$\overline{S\_BG}$	Input	—
Address bus busy	$\overline{P\_ABB}$	I/O	Three-state	$\overline{S\_ABB}$	I/O	Three-state
Data bus grant	Not available on processor bus			$\overline{S\_DBG}$	Input	—
Data bus busy	Not available on processor bus			$\overline{S\_DBB}$	I/O	Three-state

Table 3-2. RAM Interface, Configuration, and Test Signals

Function	Mnemonic	Type	Reset
RAM Interface Signals			
Address bus	R_A16–R_A0	Output	Low
RAM write enable	$\overline{\text{RWE7–0}}$	Output	Negated
Processor input enable	$\overline{\text{PIE}}$	Output	Negated
Processor output enable	$\overline{\text{POE}}$	Output	Negated
System input enable	$\overline{\text{SIE}}$	Output	Negated
System output enable	$\overline{\text{SOE}}$	Output	Negated
Flush Signals			
Flush control	F1–F0	Input	—
Flush busy	$\overline{\text{FBSY}}$	Output	—
System Configuration Signals			
Chip select	CS	Input	Negated
System clock	CLK	Input	—
Half-speed system clock	HCLK	Input	—
Reset	$\overline{\text{RST}}$	Input	—
Tag function descriptor 0/line size	FD0/LINSIZ	I/O	Asserted
Tag function descriptor 1/critical word mode	FD1/CWM	I/O	Asserted
Tag function descriptor 2	FD2	I/O	Asserted
Tag status descriptor 0/chip select polarity	SD0/CSP	I/O	Asserted
Tag status descriptor 1/external arbiter enable	SD1/ARBEN	I/O	Asserted
Tag status descriptor 2/cache size 0	SD2/CSIZ0	I/O	Asserted
Tag status descriptor 3/cache size 1	SD3/CSIZ1	I/O	Asserted
Test Signals			
Diagnostic	$\overline{\text{DIAG}}$	Input	—
JTAG test reset	$\overline{\text{TRST}}$	Input	—
JTAG test mode select	TMS	Input	—
JTAG test clock	TCK	Input	—
JTAG test data input	TDI	Input	—
JTAG test data output	TDO	Output	—

3.1 PROCESSOR INTERFACE SIGNALS

The processor interface signals can be functionally grouped into the processor address bus, processor transfer attribute, processor transfer control, processor address retry (snoop control), and processor bus arbitration signals. The processor interface signals are identified by the prefix P_.

Processor Address Bus (P_A31–P_A0)

The P_A31–P_A0 signals comprise the address bus for all processor bus transactions. These signals are outputs when the MC88410 has mastership of the address bus and inputs when the MC88110 has mastership of the processor bus (see **2.4 Cache Coherency**). These signals are three-stated at all other times.

3.1.1 Processor Transfer Attribute Signals

The MC88410 processor transfer attribute signals differ from the MC88110 by the absence of the $\overline{\text{MC}}$ signal. The timing for each of the transfer attribute signals is the same as the timing for the address bus signals, except during a locked transaction. Since the MC88410 parks the MC88110 ($\overline{\text{P\_BG}}$ asserted) between the two transactions of a locked transaction, the transfer attribute signals remain asserted during both transactions.

Processor Read/Write (P_R/$\overline{\text{W}}$)

The P_R/$\overline{\text{W}}$ signal indicates whether the transaction is a read (P_R/$\overline{\text{W}}$ high) or a write (P_R/$\overline{\text{W}}$ low) transaction. The P_R/$\overline{\text{W}}$ signal is an input when the MC88110 drives an address and an output only during a primary cache invalidate transaction (see **2.4 Cache Coherency**). It is three-stated at all other times.

Processor Lock ($\overline{\text{P\_LK}}$)

The MC88110 drives the $\overline{\text{P\_LK}}$ signal to indicate that an access is part of an atomic data-access sequence. The MC88110 asserts the $\overline{\text{P\_LK}}$ signal during locked transactions only.

During the execution of the locked transaction, the MC88110 asserts the $\overline{\text{P\_LK}}$ signal for both the read and write portions of the locked transaction. The $\overline{\text{P\_LK}}$ signal is asserted to indicate that the bus arbitration circuitry should not allow another bus master to alter the data that the locked transaction accesses between the read and write transactions.

The MC88410 drives the $\overline{\text{S\_LK}}$ signal to the system bus in response to the MC88110 assertion of the $\overline{\text{P\_LK}}$ signal. The state of $\overline{\text{P\_LK}}$ determines the status of the $\overline{\text{S\_LK}}$ signal.

Processor Cache-Inhibit ($\overline{\text{P\_CI}}$)

The $\overline{\text{P\_CI}}$ signal indicates that the data will not be written into the MC88110 data cache. For single-beat transactions, **xmem** transactions, and touch and allocate load transactions, the $\overline{\text{P\_CI}}$ signal reflects the value of the CI bit in the address translation cache entry of the MC88110. For all other transactions, the $\overline{\text{P\_CI}}$ signal is negated.

The $\overline{\text{P\_CI}}$ input signal causes the MC88410 to treat the current transaction as cache-inhibited. If the transaction hits in the secondary cache tags, the secondary cache line is flushed and invalidated before the access proceeds.

Processor Write-Through ($\overline{\text{P\_WT}}$)

The $\overline{\text{P\_WT}}$ input signal determines the memory update policy of the secondary cache. See **Section 2 Secondary Cache Operation** for more information.

Processor User Page Attributes ($\overline{\text{P\_UPA1}}$–$\overline{\text{P\_UPA0}}$)

The $\overline{\text{P\_UPA1}}$ and $\overline{\text{P\_UPA0}}$ input signals reflect the user attribute bits in the ATC entry of the MC88110. During MC88110 copyback operations, these signals are negated. The signals are received from the MC88110 and passed to the system interface for all transactions that require system bus interface mastership.

Processor Transfer Burst ($\overline{\text{P\_TBST}}$)

The $\overline{\text{P\_TBST}}$ signal indicates whether the transaction is single-beat or burst. It is an input when the MC88110 is driving an address and an output during a primary cache invalidate transaction. It is three-stated at all other times. When the $\overline{\text{P\_TBST}}$ signal is asserted by the MC88110, the transaction is an eight-word burst. If it is negated, the transaction is a single-beat transaction and the size of the data to be transferred is encoded in the P_TSIZ1–P_TSIZ0 signals. Note that $\overline{\text{P\_TBST}}$ is ignored as an input if $\overline{\text{P\_CI}}$ is asserted. The MC88410 asserts the $\overline{\text{P\_TBST}}$ signal for a primary cache DMA invalidate transaction and negates it for a primary cache invalidate transaction.

Processor Transfer Size (P_TSIZ1–P_TSIZ0)

The P_TSIZ1 and P_TSIZ0 signals indicate the size of the requested data transfer as shown in Table 3-3. All transfers are aligned to their respective size boundaries. The P_TSIZ1–P_TSIZ0 signals may be used along with P_A2–P_A0 to determine which portion of the data bus contains valid data for a write transaction or which portion of the bus should contain valid data for a read transaction. Note that the P_TSIZ1–P_TSIZ0 signals indicate the size of the requested data transfer independent of the value of $\overline{\text{P\_TBST}}$, so it is possible for the processor transfer size signals to indicate a byte, half word, or word transfer when the $\overline{\text{P\_TBST}}$ signal is asserted. Therefore, if the $\overline{\text{P\_TBST}}$ signal is asserted, the MC88410 transfers double words regardless of the P_TSIZ1–P_TSIZ0 encoding.

Table 3-3. Processor Transfer Size Signal Encoding

P_TSIZ1–P_TSIZ0	Transfer Size
0 0	Double word (64 bits)
0 1	Word (32 bits)
1 0	Half word (16 bits)
1 1	Byte (8 bits)

Processor Transfer Code (P_TC3–P_TC0)

The P_TC3–P_TC0 signals provide supplemental information about the corresponding address. The transfer code signals are encoded as shown in Table 3-4.

Table 3-4. Processor Transfer Code Signal Encoding

P_TC3–P_TC0	Transfer Code
0 0 0 0	Reserved
0 0 0 1	User data access
0 0 1 0	User touch, flush, or allocate access
0 0 1 1	Data MMU table search operation
0 1 0 0	Replacement copyback
0 1 0 1	Supervisor data access
0 1 1 0	Supervisor touch, flush, or allocate access
0 1 1 1	Snoop copyback operation
1 0 0 0	Reserved
1 0 0 1 .	User instruction access
1 0 1 0	Reserved
1 0 1 1	Instruction MMU table search operation
1 1 0 0	Reserved
1 1 0 1	Supervisor instruction access
1 1 1 0	Reserved
1 1 1 1	Reserved

Processor Invalidate ($\overline{\text{P\_INV}}$)

When asserted by the MC88410, the $\overline{\text{P\_INV}}$ output signal indicates that the MC88110 should invalidate the cache line on a snoop hit. If the snoop hit is to a modified primary cache line, the line is copied back before being invalidated. This signal is an input when the MC88110 is driving an address and an output when the MC88410 performs a primary cache invalidate transaction to the MC88110. It is three-stated at all other times.

Processor Global ($\overline{\text{P\_GBL}}$)

The processor address bus master asserts the $\overline{\text{P\_GBL}}$ signal to indicate that the transaction in progress is marked as "global." The $\overline{\text{P\_GBL}}$ signal reflects the value specified for the memory reference in the corresponding MC88110 memory management unit. The MC88410 asserts the $\overline{\text{P\_GBL}}$ signal during a primary cache invalidate transaction, so the MC88110 snoops on the address being driven. When the MC88110 is driving the processor address bus, the $\overline{\text{P\_GBL}}$ signal is an input to the MC88410.

Processor Cache Line (P_CL)

The P_CL input signal indicates which line in the MC88110 cache is involved in the current data transfer as shown in Table 3-5.

Table 3-5. Processor Cache Line Signal

P_CL	Cache Line
0	Line 0
1	Line 1

3.1.2 Processor Transfer Control Signals

The MC88410 transfer control signals are discussed in the following paragraphs.

Processor Transfer Start ($\overline{\text{P\_TS}}$)

The MC88410 asserts the $\overline{\text{P\_TS}}$ output signal to indicate that a transaction has started and the driven address is valid. The MC88410 asserts the $\overline{\text{P\_TS}}$ signal for one clock and then negates it for the duration of the transaction. The $\overline{\text{P\_TS}}$ signal is an output when the MC88410 is the processor bus master and initiates an invalidate transaction. It is an input when the MC88410 is not performing a transaction. The $\overline{\text{P\_TS}}$ signal should be connected to both the MC88110 $\overline{\text{TS}}$ and $\overline{\text{SR}}$ signals.

Processor Transfer Acknowledge ($\overline{\text{P\_TA}}$)

During a processor read transaction, the MC88410 asserts $\overline{\text{P\_TA}}$ on every clock that new data is valid. During a write transaction, the MC88410 asserts $\overline{\text{P\_TA}}$ on every clock that new data from the MC88110 has been latched by the secondary cache or main memory.

Processor Pretransfer Acknowledge ($\overline{\text{P\_PTA}}$)

The MC88410 asserts the $\overline{\text{P\_PTA}}$ output signal to indicate that the initial (or only) assertion of the transaction may follow on the next rising clock edge. During the clock between the assertion of the $\overline{\text{P\_TS}}$ and $\overline{\text{P\_PTA}}$ signals, the data unit of the MC88110 can continue to access the data cache (cache hits only) even though a bus transaction is in progress. The MC88410 asserts $\overline{\text{P\_PTA}}$ in the clock following the assertion of the $\overline{\text{P\_TS}}$ by the MC88110. For systems that do not require decoupled cache accesses, this signal may be connected to ground.

Processor Transfer Error Acknowledge ($\overline{\text{P\_TEA}}$)

The MC88410 asserts the $\overline{\text{P\_TEA}}$ output signal to indicate that a bus transaction error has occurred. The assertion of $\overline{\text{P\_TEA}}$ results in the immediate termination of the transfer in progress. The $\overline{\text{P\_TEA}}$ signal reflects the status of the $\overline{\text{S\_TEA}}$ input signal. The actions of the MC88410 after the transfer is terminated are described in **5.6.4 Transfer Error Termination.**

Processor Transfer Retry ($\overline{\text{P\_TRTRY}}$)

The MC88410 asserts the $\overline{\text{P\_TRTRY}}$ output signal to indicate that the current processor transaction should be terminated and reinitiated. The assertion of $\overline{\text{P\_TRTRY}}$ results in the immediate termination of the transaction. The assertion of the signal allows the MC88410 to regain processor bus mastership during a processor transaction. The actions of the MC88410 after the transfer is terminated are described in **5.6.2 Transfer Retry**

Termination. If the $\overline{\text{P\_TRTRY}}$ signal is asserted at the same time as the $\overline{\text{P\_TEA}}$ signal, the $\overline{\text{P\_TEA}}$ signal has priority and an error termination occurs.

Processor Address Retry ($\overline{\text{P\_ARTRY}}$)

The $\overline{\text{P\_ARTRY}}$ signal (used for snoop control) is an input that indicates to the MC88410 that it should terminate the transaction and reinitiate the transaction later. The MC88410 responds to the signal by releasing the processor bus to allow a copyback transaction by the processor. The $\overline{\text{P\_ARTRY}}$ signal of the MC88410 should be connected to the $\overline{\text{SSTAT1}}$ signal of the MC88110.

3.1.3 Processor Bus Arbitration Signals

The processor bus arbitration signals are discussed in the following paragraphs.

Processor Bus Request ($\overline{\text{P\_BR}}$)

The MC88410 asserts the $\overline{\text{P\_BR}}$ output signal to request processor bus mastership in systems using external arbitration on the processor bus. The MC88410 continues to assert the signal until it receives a qualified bus grant. A qualified bus grant is $\overline{\text{P\_BR}}$ asserted and $\overline{\text{P\_ABB}}$ negated.

The $\overline{\text{P\_BR}}$ signal is only asserted by the MC88410 to perform the primary cache DMA invalidate and primary cache invalidate transactions.

Processor Bus Grant ($\overline{\text{P\_BG}}$)

The $\overline{\text{P\_BG}}$ signal is an input and output. The $\overline{\text{P\_BG}}$ signal is an input when external arbitration is used and an output when MC88410 on-chip arbitration is used.

The external bus arbiter asserts the $\overline{\text{P\_BG}}$ input signal to indicate to the MC88410 that it has been granted address bus mastership. The MC88410 assumes address bus mastership only if $\overline{\text{P\_BG}}$ is asserted and the bus is not already in use ($\overline{\text{P\_ABB}}$ negated). The external arbiter may "park" the MC88410 on the bus by keeping $\overline{\text{P\_BG}}$ asserted after the $\overline{\text{P\_BR}}$ signal has been negated (see **4.3 Processor Bus Arbitration**). As an output signal, the MC88410 on-chip arbiter asserts $\overline{\text{P\_BG}}$ to grant the bus to the MC88110, and negates it to take control of the bus.

Processor Address Bus Busy ($\overline{\text{P\_ABB}}$)

The current address bus master asserts the $\overline{\text{P\_ABB}}$ signal to indicate that potential bus masters must wait to take mastership of the address bus. Potential address bus masters use this input to qualify $\overline{\text{P\_BG}}$.

The $\overline{\text{P\_ABB}}$ signal is an input when the MC88410 arbitrates for the processor bus and uses the signal to qualify the bus grant. The MC88410 asserts the $\overline{\text{P\_ABB}}$ signal as an output when it has mastership of the bus.

3.2 SYSTEM INTERFACE SIGNALS

The system interface signals can be functionally grouped into the system address bus, system transfer attribute, system transfer control, system snoop control, and system bus arbitration signals. The system interface signals are identified by the prefix **S_**.

System Address Bus (S_A31–S_A0)

The S_A31–S_A0 signals comprise the address bus for all system bus transactions. These signals are outputs when the MC88410 has mastership of the address bus and inputs when the MC88410 has mastership of the system bus (see **2.4 Cache Coherency**). These signals are three-stated at all other times.

3.2.1 System Transfer Attribute Signals

The MC88410 system transfer attribute signals differ from the MC88110 signals by the absence of the write-through ($\overline{WT}$) and cache line (CLINE) signals. The timing for each of the transfer attribute signals is the same as the timing for the address bus signals, except during a locked transaction (**xmem**). If the external arbiter parks the MC88410 on the system bus between the two transactions of an **xmem** operation, the transfer attribute signals remain asserted during both transactions.

System Read/Write (S_R/$\overline{W}$)

The S_R/$\overline{W}$ signal indicates whether the transaction is a read (S_R/$\overline{W}$ high) or a write (S_R/$\overline{W}$ low) transaction. The S_R/$\overline{W}$ signal is an output when the MC88410 is driving an address and an input when the MC88410 is snooping (see **2.4 Cache Coherency**). It is three-stated at all other times.

System Lock ($\overline{S\_LK}$)

The MC88110 drives the $\overline{P\_LK}$ signal to indicate that an access is part of an atomic data-access sequence. The MC88410 asserts the $\overline{S\_LK}$ signal during locked transactions only.

During the execution of the locked transaction, the MC88410 asserts the $\overline{S\_LK}$ signal for both the read and write portions of the locked (**xmem**) transaction. The $\overline{S\_LK}$ signal is asserted to indicate that the system bus arbitration circuitry should not allow another bus master to alter the data that the locked transaction accesses between the read and write transactions. The state of $\overline{P\_LK}$ determines the status of the $\overline{S\_LK}$ signal.

System Cache-Inhibit ($\overline{S\_CI}$)

The $\overline{S\_CI}$ signal indicates that the data will not be written into the secondary cache. The state of $\overline{P\_CI}$ determines the status of the $\overline{S\_CI}$ signal.

The $\overline{P\_CI}$ signal causes the MC88410 to treat the current transaction as cache-inhibited. If the transaction hits in the secondary cache tags, the secondary cache line is flushed and invalidated before the access proceeds.

System User Page Attributes ($\overline{S\_UPA1}$–$\overline{S\_UPA0}$)

The $\overline{S\_UPA1}$ and $\overline{S\_UPA0}$ input signals reflect the user attribute bits in the ATC entry of the MC88110. During MC88110 copyback operations, these signals are negated. The signals are received from the MC88110 and passed to the system interface for all transactions that require system bus interface mastership.

System Transfer Burst ($\overline{S\_TBST}$)

The $\overline{S\_TBST}$ signal indicates whether the transaction is single-beat or burst. It is an output when the MC88410 is driving an address and an input when the MC88410 is snooping an address. It is three-stated at all other times. When the $\overline{S\_TBST}$ signal is asserted, the transaction is a line burst. If it is negated, the transaction is a single-beat transaction, and the size of the data to be transferred is encoded in the S_TSIZ1–S_TSIZ0 signals.

System Transfer Size (S_TSIZ1–S_TSIZ0)

The S_TSIZ1 and S_TSIZ0 signals indicate the size of the requested data transfer as shown in Table 3-6. All transfers are aligned to their respective size boundaries. The S_TSIZ1–S_TSIZ0 signals may be used along with S_A2–S_A0 to determine which portion of the data bus contains valid data for a write transaction or which portion of the bus should contain valid data for a read transaction. Note that S_TSIZ1–S_TSIZ0 indicate the size of the requested data transfer independent of the value of $\overline{S\_TBST}$, so it is possible for the system transfer size signals to indicate a byte, half word, or word transfer when the $\overline{S\_TBST}$ signal is asserted. However, if $\overline{S\_TBST}$ is asserted, the memory system must transfer double words regardless of the S_TSIZ1–S_TSIZ0 encoding.

Table 3-6. System Transfer Size Signal Encoding

S_TSIZ1–S_TSIZ0	Transfer Size
0 0	Double word (64 bits)
0 1	Word (32 bits)
1 0	Half word (16 bits)
1 1	Byte (8 bits)

System Transfer Code (S_TC3–S_TC0)

The S_TC3–S_TC0 signals provide supplemental information about the corresponding address. The transfer code signals are encoded as shown in Table 3-7.

Table 3-7. System Transfer Code Signal Encoding

S_TC3–S_TC0	Transfer Code
0 0 0 0	Reserved
0 0 0 1	User data access
0 0 1 0	User touch, flush, or allocate access
0 0 1 1	Data MMU table search operation
0 1 0 0	Replacement copyback operation
0 1 0 1	Supervisor data access
0 1 1 0	Supervisor touch, flush, or allocate access
0 1 1 1	Snoop copyback operation
1 0 0 0	Reserved
1 0 0 1	User instruction access
1 0 1 0	Reserved
1 0 1 1	Instruction MMU table search operation
1 1 0 0	Reserved
1 1 0 1	Supervisor instruction access
1 1 1 0	Reserved
1 1 1 1	Reserved

System Invalidate ($\overline{S\_INV}$)

When asserted, the $\overline{S\_INV}$ output signal indicates that snooping MC88410s should invalidate their secondary cache lines on a snoop hit. If the snoop hit is to a modified line, the secondary cache line is copied back before being invalidated. If the snoop hit is to a line included in the primary cache, a primary cache invalidate transaction occurs before the secondary cache snoop copyback operation. The $\overline{S\_INV}$ signal is an input when the MC88410 is driving an address and an output during MC88410 system transactions. It is three-stated at all other times. The $\overline{S\_INV}$ signal is asserted for all write transactions and read transactions that are intent-to-modify. It is negated for all other read transactions.

System Memory Cycle ($\overline{S\_MC}$)

When asserted, the $\overline{S\_MC}$ output signal indicates that a data transfer transaction is in progress. When $\overline{S\_MC}$ is negated, the current bus transaction is an invalidate cycle and no data is transferred. During invalidate cycles, valid data is driven, but the memory system is not required to execute the data write. The $\overline{S\_MC}$ signal is an output when the MC88410 is driving an address and an input when the MC88410 is snooping. It is three-stated at all other times.

System Global ($\overline{S\_GBL}$)

The system address bus master asserts the $\overline{S\_GBL}$ signal to indicate that the transaction in progress is marked as "global." Normally, $\overline{S\_GBL}$ reflects the value specified for the memory reference in the corresponding MC88110 MMU. The MC88410 asserts the $\overline{S\_GBL}$

signal during a system invalidate transaction, so other devices will snoop the address being driven. When the MC88410 is snooping, $\overline{\text{S\_GBL}}$ is an input.

3.2.2 System Transfer Control Signals

The MC88410 system transfer control signals differ from the MC88110 signals by the absence of the $\overline{\text{PTA}}$ signal. The MC88410 system transfer control signals are discussed in the following paragraphs.

System Transfer Start ($\overline{\text{S\_TS}}$)

The MC88410 asserts the $\overline{\text{S\_TS}}$ output signal to indicate that a transaction has started on the system bus and the driven address is valid. The MC88410 asserts the $\overline{\text{S\_TS}}$ signal for one clock and then negates it for the duration of the transaction. The $\overline{\text{S\_TS}}$ signal is an output when the MC88410 initiates a system bus transaction. It is three-stated at all other times.

System Transfer Acknowledge ($\overline{\text{S\_TA}}$)

The assertion of $\overline{\text{S\_TA}}$ (while $\overline{\text{S\_TRTRY}}$ and $\overline{\text{S\_TEA}}$ are negated) indicates normal transaction termination to the MC88410.

In the full-speed mode, assertion of $\overline{\text{S\_TA}}$ indicates that the memory system is ready to supply or latch the data in the following clock. For a read transaction, the data is valid on the data bus and may be latched by the MC88410 in the following clock. For a write transaction, the memory system can accept the data in the following clock. In the half-speed mode, assertion of $\overline{\text{S\_TA}}$ indicates to the MC88410 that the current data transfer has completed successfully. For a read transaction, the data is valid on the data bus and may be latched by the MC88410. For a write transaction, the data has been accepted by the memory system.

System Transfer Error Acknowledge ($\overline{\text{S\_TEA}}$)

The memory system asserts the $\overline{\text{S\_TEA}}$ output signal to indicate that a bus transaction error has occurred. The assertion of $\overline{\text{S\_TEA}}$ results in the immediate termination of the transfer in progress. If the system transaction is the result of a processor transaction, the MC88410 asserts $\overline{\text{P\_TEA}}$. The actions of the MC88410 after the transfer is terminated are described in **5.6.4 Transfer Error Termination.**

System Transfer Retry ($\overline{\text{S\_TRTRY}}$)

The memory system asserts the $\overline{\text{S\_TRTRY}}$ output signal to indicate that the current system bus transaction should be terminated and reinitiated. The assertion of $\overline{\text{S\_TRTRY}}$ results in the immediate termination of the transaction. The actions of the MC88410 after the transfer is terminated are described in **5.6.2 Transfer Retry Termination.** If $\overline{\text{S\_TRTRY}}$ is asserted at the same time as the $\overline{\text{S\_TEA}}$ signal, $\overline{\text{S\_TEA}}$ has priority.

System Address Acknowledge ($\overline{\text{S\_AACK}}$)

The MC88410 $\overline{\text{S\_AACK}}$ signal is used in systems that have the split-bus capability to terminate address bus mastership. When the $\overline{\text{S\_AACK}}$ input signal is asserted, the MC88410 stops driving an address on the address bus and negates the $\overline{\text{S\_ABB}}$ signal.

3.2.3 System Snoop Control Signals

The MC88410 system snoop control signals are discussed in the following paragraphs.

System Snoop Request ($\overline{S\_SR}$)

The $\overline{S\_SR}$ input signal indicates to all snooping MC88410s that the current address on the system interface should be latched because a snoop lookup may be required. The $\overline{S\_SR}$ signal can be connected to the $\overline{S\_TS}$ signal of other bus masters. Since the MC88410 ignores $\overline{S\_SR}$ when it drives $\overline{S\_TS}$, $\overline{S\_SR}$ may also be connected to the $\overline{S\_TS}$ signal of the same MC88410. Note that $\overline{S\_SR}$ must only be asserted for one clock and therefore should not be connected to $\overline{S\_GBL}$. The $\overline{S\_SR}$ signal must be negated and reasserted between two accesses that need to be snooped, or it will be ignored on the second access. The MC88410 only snoops transactions when both the $\overline{S\_SR}$ and $\overline{S\_GBL}$ signals are asserted.

System Address Retry ($\overline{S\_ARTRY}$)

The $\overline{S\_ARTRY}$ signal is an input that indicates to the current bus master that another bus master has requested that it terminate the address bus tenure and reinitiate the transaction later. An MC88410 that is the current bus master responds to the signal by releasing the system bus to allow a copyback operation. The MC88410 detects a qualified $\overline{S\_ARTRY}$ on the clock edge following the assertion of $\overline{S\_TS}$. The $\overline{S\_ARTRY}$ signal is qualified with $\overline{S\_AACK}$ or with the first beat of data.

System Snoop Status ($\overline{S\_SSTAT2}$–$\overline{S\_SSTAT0}$)

The $\overline{S\_SSTAT2}$–$\overline{S\_SSTAT0}$ signals indicate the status of the MC88410 transaction as shown in Table 3-8. These output signals are asserted by a snooping bus master when it detects a snoop hit or collision. The $\overline{S\_SSTAT2}$ signal indicates that a copyback operation will occur. $\overline{S\_SSTAT1}$ is asserted for both snoop hits and collisions, so it can be directly or indirectly connected to the $\overline{S\_ARTRY}$ signal of other processors. The $\overline{S\_SSTAT0}$ signal is asserted for all snoop hits, so it can be directly or indirectly connected to the $\overline{SHD}$ signal of other bus masters. When $\overline{S\_SSTAT1}$ and $\overline{S\_SSTAT0}$ are asserted but $\overline{S\_SSTAT2}$ is negated, then each signal must have been asserted by a different snooping device.

Table 3-8. Snoop Status Signals

$\overline{SSTAT2}$	$\overline{SSTAT1}$	$\overline{SSTAT0}$	Status
Three-state	Three-state	Three-state	No collision, no snoop hit
Three-state	Three-state	Asserted	Snoop hit shared
Three-state	Asserted	Three-state	Pipeline collision
Asserted	Asserted	Asserted	Snoop hit modified, will copy back

Shared ($\overline{SHD}$)

The assertion of the $\overline{SHD}$ signal indicates that the cache line currently being read into the data cache should be marked as shared-unmodified. If $\overline{SHD}$ is negated, the cache line is marked as exclusive-unmodified. If the $\overline{S\_INV}$ signal is asserted for the transaction, the

line is marked exclusive-unmodified regardless of the state of the $\overline{\text{SHD}}$ signal. The timing of the $\overline{\text{SHD}}$ input is the same as the timing for $\overline{\text{S\_ARTRY}}$.

Transfer Shared ($\overline{\text{TSHD}}$)

The $\overline{\text{TSHD}}$ input signal provides a mechanism for other devices on the system interface to specify during the data bus tenure that the current secondary cache line should be marked shared-unmodified. The memory system can derive $\overline{\text{TSHD}}$ from the $\overline{\text{S\_SSTATO}}$ signal. The MC88410 samples $\overline{\text{TSHD}}$ at the same time as data, allowing the memory system to collect and pipeline snoop responses from distant snooping devices that have a long response time.

Note that if the MC88410 detects the $\overline{\text{TSHD}}$ signal asserted during the first data beat of a secondary cache line allocation, the cache line is placed into a shared-unmodified state instead of an exclusive-modified state.

Flush Control (F1–F0)

The MC88410 system interface has two flush control input signals, namely the F1 (flush 1) and F0 (flush 0) signals. The MC88410 initiates the flush and invalidate operations when it detects the appropriate encoding of flush control signals for at least one clock cycle. The encoding for F1 and F0 is shown in Table 3-9.

Table 3-9. Flush Control Signal Encoding

F1	F0	Function
0	0	No operation
0	1	Flush page
1	0	Flush all
1	1	Invalidate all

Flush Busy ($\overline{\text{FBSY}}$)

The $\overline{\text{FBSY}}$ signal is an MC88410 system interface output indicating when a flush or invalidate operation is in progress. The $\overline{\text{FBSY}}$ signal is asserted at the beginning of a flush operation and is negated when the flush operation is complete.

To initiate a flush or invalidate operation, the appropriate flush control signals must be asserted for one clock cycle. When the MC88410 recognizes the flush control signal encoding, it asserts the $\overline{\text{FBSY}}$ signal one clock later and begins the flush or invalidate operation. The $\overline{\text{FBSY}}$ signal may be used to clear the external control register which drives the flush control signals.

3.2.4 System Bus Arbitration Signals

The MC88410 system bus arbitration signals are discussed in the following paragraphs.

System Bus Request ($\overline{S\_BR}$)

The MC88410 asserts the $\overline{S\_BR}$ output signal to request system bus mastership and continues to assert the signal until it receives a qualified bus grant or determines that it does not need the bus. A qualified bus grant is $\overline{S\_BG}$ asserted and $\overline{S\_ABB}$ negated.

To avoid the overhead of arbitration, it may be desirable to park the MC88410 on the system address bus. The MC88410 is parked when $\overline{S\_BG}$ is asserted whether or not the MC88410 is requesting bus mastership. If $\overline{S\_BG}$ remains asserted until an internal bus request occurs, the MC88410 completes the arbitration sequence without any overhead and can begin the transaction without asserting $\overline{S\_BR}$. Thus bus parking provides a performance advantage because bus accesses occur without any delay in the arbitration protocol.

System Bus Grant ($\overline{S\_BG}$)

The $\overline{S\_BG}$ signal is an input that the external arbiter uses to grant bus ownership to the MC88410 in response to a bus request.

The external bus arbiter asserts the $\overline{S\_BG}$ input signal to indicate to the MC88410 that it has been granted address bus mastership. The MC88410 assumes address bus mastership only if $\overline{S\_BG}$ is asserted and the bus is not already in use ($\overline{S\_ABB}$ negated). The external arbiter may "park" the MC88410 on the bus by keeping $\overline{S\_BG}$ asserted after the $\overline{S\_ABB}$ has been negated (see **5.4.5 System Bus Parking**).

System Address Bus Busy ($\overline{S\_ABB}$)

The current address bus master asserts the $\overline{S\_ABB}$ signal to indicate that potential bus masters must wait to take mastership of the address bus. Potential address bus masters use this input to qualify $\overline{S\_BG}$. It is an output when the MC88410 is the address bus master and an input at all other times.

The $\overline{S\_ABB}$ signal may be a shared signal among multiple MC88410s or other bus masters. It must be connected to a pull-up resistor so that it remains negated when no devices have control of the address bus.

System Data Bus Grant ($\overline{S\_DBG}$)

The $\overline{S\_DBG}$ input signal is used by the external bus arbiter to grant data bus mastership in response to a data bus request. The assertion of $\overline{S\_TS}$ serves as the data bus request. The MC88410 only assumes data bus mastership if $\overline{S\_DBG}$ is asserted and the data bus is not already busy ($\overline{S\_DBB}$ negated).

System Data Bus Busy ($\overline{S\_DBB}$)

The $\overline{S\_DBB}$ signal is asserted by the current data bus master to indicate that potential data bus masters must wait to take mastership of the data bus. Potential data bus masters use this input to qualify data bus grant. The $\overline{S\_DBB}$ signal is an input when the MC88410 is arbitrating to obtain data bus mastership and an output when the MC88410 is the data bus master. It is three-stated at all other times.

3.3 RAM INTERFACE SIGNALS

The MC88410 controls the MCM62110 secondary cache through the RAM interface. The RAM interface signals are output signals that the MC88410 drives to the MCM62110 array. The R_A16–R_A0, $\overline{\text{RWE7}}$–$\overline{\text{RWE0}}$, $\overline{\text{SIE}}$, and $\overline{\text{SOE}}$ signals are involved in system bus transactions.

RAM Address Bus (R_A16–R_A0)

The R_A16–R_A0 output signals provide the MCM62110 array with the address of the transaction. For processor transactions, the R_A16–R_A0 signals are asserted in the clock after the address is driven on the processor bus (P_A31–P_A0). For system bus burst transactions, the MC88410 automatically increments the address to the secondary cache in the clock before data is valid. For this reason, $\overline{\text{S\_TA}}$ must be asserted one clock before the data is valid for all full-speed transactions.

RAM Write Enable ($\overline{\text{RWE7}}$–$\overline{\text{RWE0}}$)

The $\overline{\text{RWE7}}$–$\overline{\text{RWE0}}$ signals allow individual bytes of the 64-bit word to be written into the secondary cache. The $\overline{\text{RWE7}}$–$\overline{\text{RWE0}}$ signals are asserted or negated during system or processor data bus mastership depending upon the transaction. Data is written into the corresponding MCM62110 array when $\overline{\text{RWE7}}$–$\overline{\text{RWE0}}$ are asserted.

Processor Input Enable ($\overline{\text{PIE}}$)

The $\overline{\text{PIE}}$ signal is asserted by the MC88410 to allow data to be driven from the MC88110 to the MCM62110 array for processor write transactions.

Processor Output Enable ($\overline{\text{POE}}$)

The $\overline{\text{POE}}$ signal is asserted by the MC88410 to allow data to be driven by the MCM62110 array to the MC88110 data signals for processor read requests.

System Input Enable ($\overline{\text{SIE}}$)

The $\overline{\text{SIE}}$ signal is asserted by the MC88410 to allow data to be driven from the system data bus to the MCM62110 array for secondary cache line fills or to the processor from memory for cache-inhibited read transactions.

System Output Enable ($\overline{\text{SOE}}$)

The $\overline{\text{SOE}}$ signal is asserted by the MC88410 to allow data to be driven by the MCM62110 array to the system data bus for copy-back operations, write-through operations, and cache-inhibited write transactions.

3.4 SYSTEM CONFIGURATION SIGNALS

The system configuration and tag monitoring signals are discussed in the following paragraphs.

The MC88410 provides a weak driver for some signals to program them to a default configuration. After reset these signals can be used for tag monitoring. Note that the tag monitoring signals (FD2–FD0 and SD3–SD0) indicate the status of the MC88410 in the

previous clock cycle (i.e. are delayed one clock). For information regarding their use for tag monitoring, refer to **Section 6 JTAG and Diagnostics.**

Chip Select (CS)

The CS signal is used to enable the MC88410. The CSP signal determines if an MC88410 is selected when the CS signal is connected high or when it is connected to ground.

In single-MC88410 configurations, the CS signal should be grounded and the CSP configuration signal should not be driven during reset. There is an internal pull up on the CSP signal that drives the chip select polarity into the proper state so that the MC88410 is always chip selected.

In dual-MC88410 configurations, the CS input signal is used to dynamically select the MC88410. The MC88410 CS signal allows the address space to be divided evenly between two MC88410 devices servicing a single MC88110. The CS signals on both the MC88410s should be connected to a single processor address bit.

System Clock (CLK)

The CLK input signal generates the internal timing signals for the MC88410. The leading edge of the clock is used as the MC88410 internal and external timing reference.

Half-Speed System Clock (HCLK)

The HCLK input signal generates the internal timing signals for the MC88410 in half-speed mode. The leading edge of the half-speed clock is used as the MC88410 internal and external timing reference. The MC88410 operates in full-speed clock mode when HCLK is held high. The CLK and HCLK signals must be in phase with each other and the rising edge of HCLK must coincide with a rising edge of CLK within the electrical specification timing.

RESET ($\overline{\text{RST}}$)

The $\overline{\text{RST}}$ signal is used to perform an orderly restart of the processor, bringing it to a known state and beginning program execution at address $0 (the reset vector). When $\overline{\text{RST}}$ is asserted, all current operations of the MC88410 are suspended. When $\overline{\text{RST}}$ is negated, all the configuration bits are latched to enable the MC88410 operation.

Tag Function Descriptor 0/Line Size (FD0/LINSIZ)

The FD0/LINSIZ signal is sampled to configure the line size during reset operation. The FD0/LINSIZ signal selects a 32-byte line length (default) when asserted and a 64-byte line length when negated. After the reset operation, the signal is used to drive the function descriptor for tag monitoring.

Tag Function Descriptor 1/Critical Word Mode (FD1/CWM)

The FD1/CWM signal is sampled to configure the critical word bursting. The FD1/CWM signal selects critical-word-first ordering (default) when asserted and zero-word-first ordering when it is negated. After reset operation, the signal is used to drive the function descriptor for tag monitoring.

Tag Function Descriptor 2 (FD2)

The FD2signal is used to drive the function descriptor for tag monitoring.

Tag Status Descriptor 0/Chip Select Polarity (SD0/CSP)

The SD0/CSP signal is sampled to configure the chip select polarity. The SD0/CSP is asserted to select the MC88410 when CS is high (default) and negated to select the MC88410 when CS is low. After reset operation, the signal is used to drive the status descriptor for tag monitoring.

In single-MC88410 configurations, the CS signal should be grounded and the CSP configuration signal should not be driven during reset. There is an internal pull up on the CSP signal that drives the chip select polarity into the proper state so that the MC88410 is always chip selected. In dual-MC88410 configurations, the CSP signal should be driven to opposite states on the two MC88410 devices at reset to provide mutually-exclusive MC88410 chip selects based on the chosen address line.

Tag Status Descriptor 1/External Arbiter Enable (SD1/ARBEN)

The SD1/ARBEN signal is sampled to configure the arbiter enable. The SD1/ARBEN signal selects on-chip arbitration when it is asserted (default) and external arbitration when it is negated. After reset operation, the signal is used to drive the status descriptor for tag monitoring.

Tag Status Descriptor 2/Cache Size 0 (SD2/CSIZ0)

The SD2/CSIZ0 signal is sampled to configure the cache size as shown in Table 3-10. After reset operation, the signal is used to drive the status descriptor for tag monitoring.

Tag Status Descriptor 3/Cache Size 1 (SD3/CSIZ1)

The SD3/CSIZ1 signal is sampled to configure the cache size as shown in Table 3-10. After reset operation, the signal is used to drive the status descriptor for tag monitoring.

Table 3-10. Cache Size Configuration

SD2/CSIZ1	SD1/CSIZ0	Cache Size
1*	1*	256 Kbyte
1	0	Reserved
0	1	1 Mbyte
0	0	Reserved

* Default configuration

3.5 TEST SIGNALS

The test signals for the MC88410 are discussed in the following paragraphs. For more information, refer to **Section 6 JTAG and Diagnostics.**

Diagnostic ($\overline{\text{DIAG}}$)

When the $\overline{\text{DIAG}}$ signal is asserted, the MC88410 is placed in diagnostic mode. All subsequent processor transactions are interpreted as diagnostic accesses. Snooping is disabled for all system bus addresses. The MC88410 recognizes FO and F1 while $\overline{\text{DIAG}}$ is asserted but does not begin flush operation until $\overline{\text{DIAG}}$ is negated. If flush or invalidate is already in an operation when $\overline{\text{DIAG}}$ is asserted, the MC88410 halts operation, resuming at the next set index upon negation of $\overline{\text{DIAG}}$. The $\overline{\text{DIAG}}$ signal must be held asserted throughout a diagnostic sequence as it is not latched by the MC88410. The $\overline{\text{DIAG}}$ signal must not be negated until two clocks after the MC88410 negates $\overline{\text{P\_TA}}$ to ensure that write transactions will complete internally.

Clock Monitor (CKMON)

This signal is an output used for factory test. It should be left unconnected.

JTAG Test Reset ($\overline{\text{TRST}}$)

Assertion of the $\overline{\text{TRST}}$ signal causes asynchronous initialization of the internal JTAG test access port (TAP) controller. The $\overline{\text{TRST}}$ signal conforms to the *IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture.*

JTAG Test Mode Select (TMS)

The TMS signal is decoded by the internal JTAG TAP controller to distinguish the primary operations of the test support circuitry. This signal conforms to the *IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture.*

JTAG Test Clock (TCK)

The TCK signal clocks the internal boundary scan test support circuitry. This signal conforms to the *IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture.*

JTAG Test Data Input (TDI)

The state of the TDI signal is clocked into the selected JTAG test instruction or data register on the rising edge of TCK. This signal conforms to the *IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture.*

JTAG Test Data Output (TDO)

The contents of the selected internal register or data test register are shifted out onto this signal on the falling edge of TCK. This signal conforms to the *IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture.*

SECTION 4
PROCESSOR BUS INTERFACE

This section provides a functional description of the MC88410 processor bus interface, the signals that control the processor bus, and the detailed timing of the bus cycles that perform data transfer operations between the MC88110 processor and the MC88410 secondary cache controller. It also includes the descriptions of bus arbitration, transaction types, and termination of bus cycles.

NOTES

The terms assert and negate are used extensively in this manual to avoid confusion between active-high and active-low signals. **Assert** or **assertion** indicates that a signal is active or true, regardless of whether the signal is active high or active low. **Negate** or **negation** indicates that the signal is inactive or false.

The MC88410/MCM62110 secondary cache contains both data and instruction data types. The term **data** is used in this manual to refer to both data and instructions unless specifically noted otherwise.

In this section, the timing for the signals is only accurate to within a half-clock cycle and is intended to demonstrate functional relationships. The input and output signals are synchronous in that all setup and hold times are specified in reference to edges of the clock signal. The MC88410 outputs are driven from a clock edge, and a maximum delay is specified. In addition, minimum hold times are specified in relation to the clock. The minimum setup and hold times must be met to guarantee proper device operation. For detailed timing information, refer to the *MC88410 Electrical Specifications.*

4.1 PROCESSOR BUS INTERFACE OVERVIEW

The MC88410 processor bus interface connects the MC88410 to the MC88110 processor address and control signals. The processor bus interface includes the signals that connect the MC88410 and the MC88110 and some of the RAM interface signals that connect the MC88410 and MCM62110 secondary cache RAM array as shown in Figure 4-1. The system bus interface signals are described in **Section 5 System Bus Interface.**

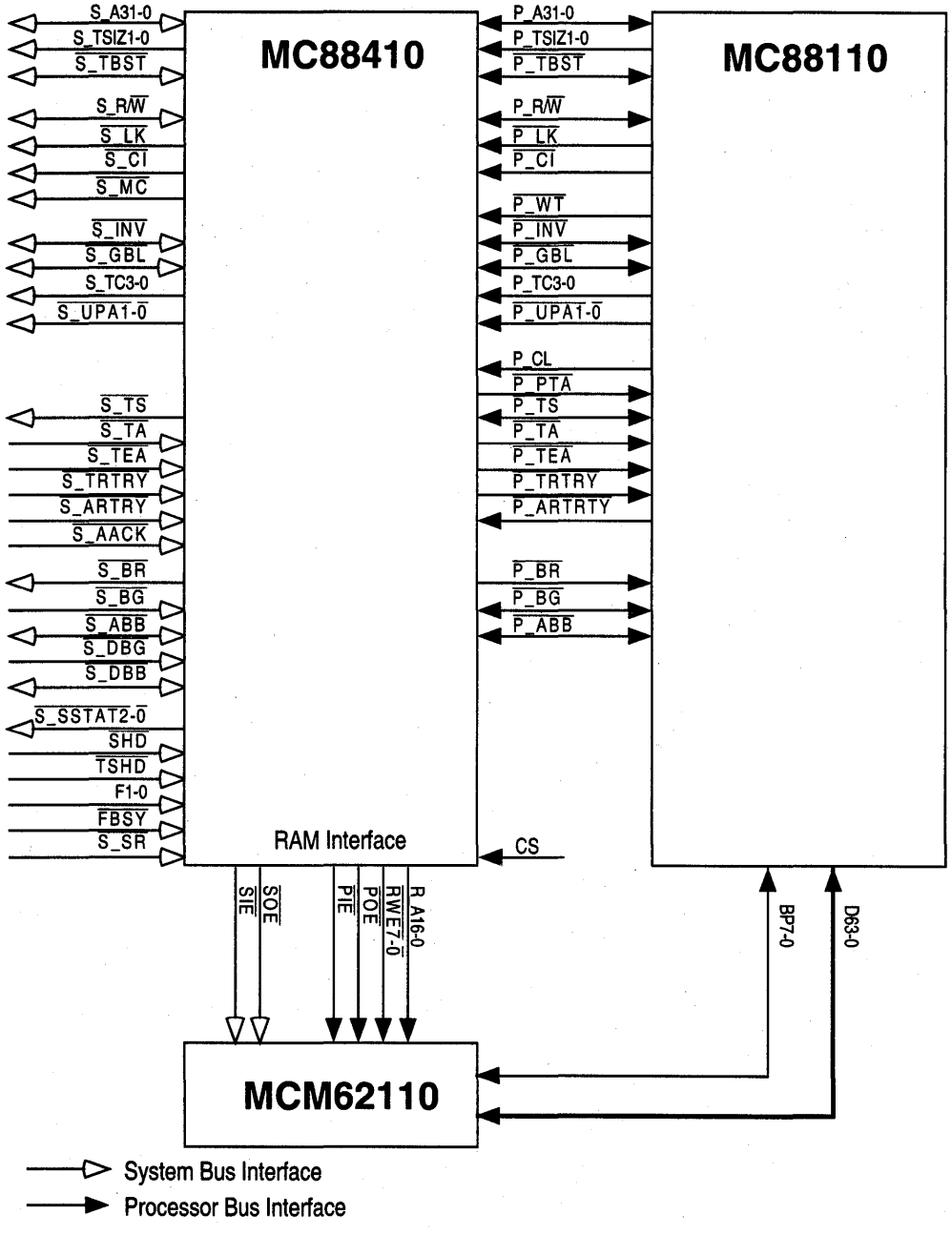


Figure 4-1. Processor Bus Interface

The interface between the MC88410 and the MC88110 consists of the address, transfer attribute, control, arbitration, and termination signals. The RAM interface between the MC88410 and the MCM62110 array consists of the address, RAM write enable, and data I/O control signals. The interface between the MC88110 and the MCM62110 array consists of the data bus and byte parity signals. Note that there is no connection between the data bus and the MC88410 because there is no cache memory on the MC88410.

Up to two MC88410 secondary cache controllers can be connected to a single MC88110 processor. Although two MC88410s and one MC88110 can have the capability of driving the processor bus, there can be only one device controlling the bus at any one time. This device is the processor bus master. Bus arbitration is the protocol by which a device becomes the bus master.

In a single-MC88410 configuration (single MC88410 connected to a single MC88110), no logic is needed to connect the MC88410 to the MC88110. The MC88410 contains on-chip arbitration logic to provide processor bus arbitration for this configuration. In a dual-MC88410 configuration (two MC88410 devices connected to a single processor), external logic is required for processor bus arbitration. The MC88410 only requires processor bus mastership to perform a primary cache invalidate or a primary cache DMA invalidate transaction. These invalidate transactions are performed by the MC88410 to maintain cache coherency as described in **Section 2 Secondary Cache Operation.**

It is recommended that no more than one MC88110 and two MC88410 devices be connected to the processor interface. Additional devices such as boot EPROM, control registers, and scratch pad RAM should be placed on the system interface of the MC88410. This preserves the low-latency memory access of the MC88110 to the secondary cache.

A typical configuration using one MC88410 and one MC88110 is shown in Figure 4-1. When the MC88110 initiates a processor bus request, it drives the appropriate signals on its address, data, and control signals. The MC88410 decodes the transfer attribute signals and determines the actions required to carry out the MC88110 request. The MC88410 then drives the address and control signals to the MCM62110 array, enabling it to provide data to or latch data from the MC88110.

4.2 MC88410 SIGNAL INTERFACE

An MC88110/MC88410 node includes a single MC88410 secondary cache controller connected to a single MC88110 processor or two MC88410 secondary cache controllers connected to a single MC88110 processor as shown in Figures 4-2 and 4-3. Several of these nodes can be put together to provide a multiprocessing system.

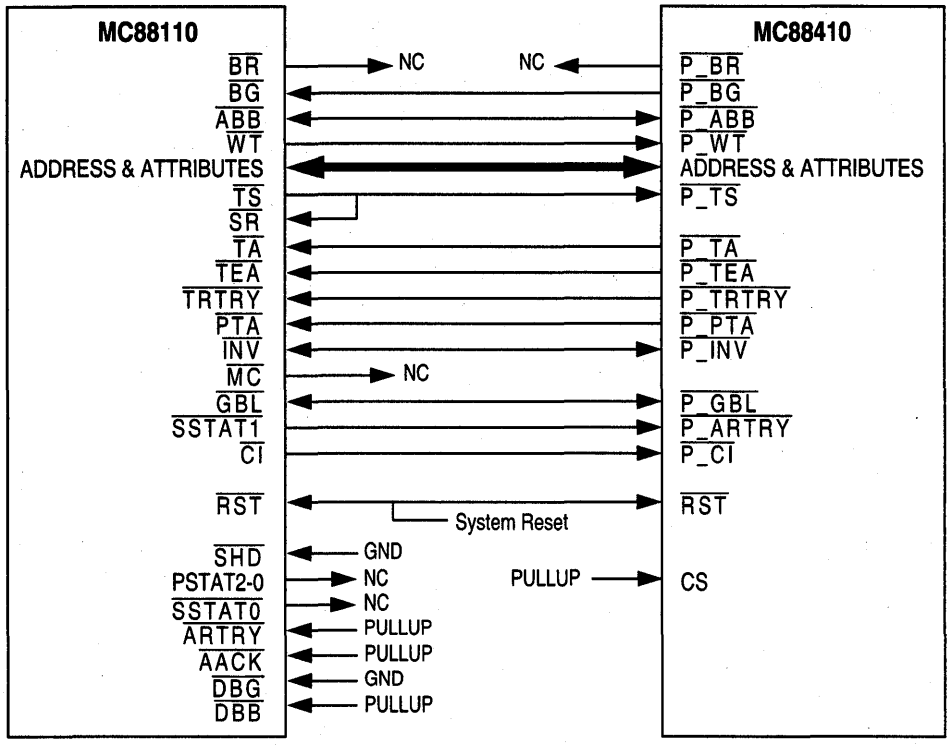


Figure 4-2. Single-MC88410 Configuration

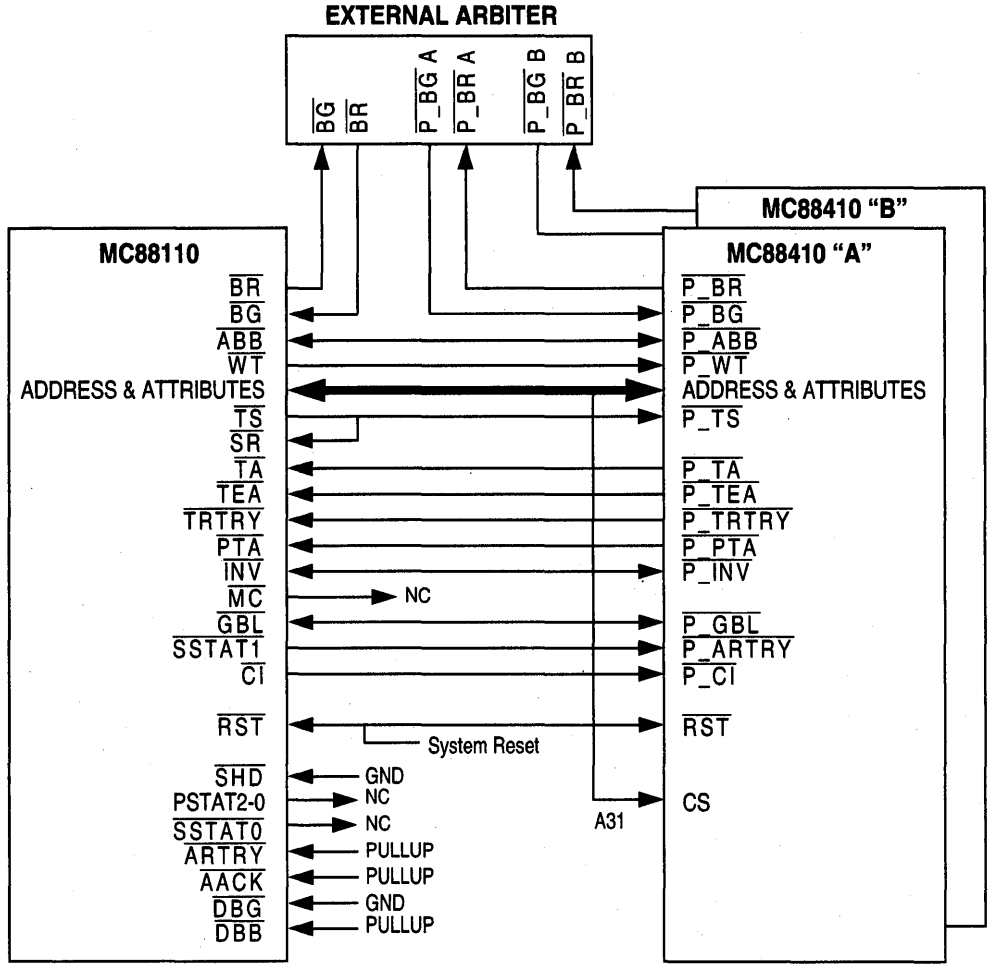


Figure 4-3. Dual-MC88410 Configuration

The following paragraphs describe the processor bus interface signals that perform similar functions on the MC88110 and MC88410, other signals necessary for connecting an MC88410 to an MC88110, and the signals used by the MC88410 to control the MCM62110 array.

4.2.1 MC88410/MC88110 Signal Relationship

Table 4-1 lists the MC88410 signals that connect directly to MC88110 signals. For more information about the function of these signals, refer to **Section 3 Signal Description**.

Table 4-1. Common MC88410/MC88110 Signals

MC88410 Signals	MC88110 Signals	Function
P_A31–P_A0	A31–A0	Address bus signals
$\overline{\text{P\_WT}}$	$\overline{\text{WT}}$	Write-through signal
$\overline{\text{P\_INV}}$	$\overline{\text{INV}}$	Invalidate signal
$\overline{\text{P\_GBL}}$	$\overline{\text{GBL}}$	Global signal
$\overline{\text{P\_CI}}$	$\overline{\text{CI}}$	Cache-inhibit signal
$\overline{\text{P\_ABB}}$	$\overline{\text{ABB}}$	Address bus busy signal
$\overline{\text{P\_TA}}$	$\overline{\text{TA}}$	Transfer acknowledge signal
$\overline{\text{P\_TEA}}$	$\overline{\text{TEA}}$	Transfer error acknowledge signal
$\overline{\text{P\_TRTRY}}$	$\overline{\text{TRTRY}}$	Transfer retry signal
$\overline{\text{P\_PTA}}$	$\overline{\text{PTA}}$	Pre-transfer acknowledge signal
$\overline{\text{RST}}$	$\overline{\text{RST}}$	Reset signal
$\overline{\text{P\_TS}}$	$\overline{\text{TS}}$	Transfer start signal
$\overline{\text{P\_TS}}$	$\overline{\text{SR}}$	Transfer start/snoop request
$\overline{\text{P\_ARTRY}}$	$\overline{\text{SSTAT1}}$	Address retry/snoop status 1

The signals in Table 4-1 are similar in function for the MC88110 and MC88410 devices except for the connection of the $\overline{\text{TS}}$, $\overline{\text{SR}}$, and $\overline{\text{SSTAT1}}$ signals.

The MC88110 asserts the $\overline{\text{TS}}$ output signal to indicate the start of a new transaction. The $\overline{\text{SR}}$ input signal indicates to the MC88110 that there is a valid address on the bus and that the MC88110 should snoop the address if the $\overline{\text{GBL}}$ I/O signal is asserted. The $\overline{\text{P\_TS}}$ I/O signal of the MC88410 must be connected to both the $\overline{\text{TS}}$ and $\overline{\text{SR}}$ signals of the MC88110.

The MC88110 asserts the $\overline{\text{SSTAT1}}$ signal to gain control of the processor bus to perform a snoop copyback transaction. If the MC88410 $\overline{\text{P\_ARTRY}}$ input signal is asserted, the MC88410 relinquishes mastership of the processor bus. The MC88410 $\overline{\text{P\_ARTRY}}$ input signal is connected to the MC88110 $\overline{\text{SSTAT1}}$ output signal to allow the MC88110 to become the processor bus master and perform a snoop copyback transaction.

When the $\overline{\text{P\_TRTRY}}$ signal is asserted by the MC88410, the MC88110 relinquishes mastership of the processor bus, reinitiating the transaction when necessary. The MC88410 asserts the $\overline{\text{P\_TRTRY}}$ signal to gain mastership of the processor bus to perform a primary cache invalidate transaction or a primary cache DMA invalidate transaction.

4.2.2 Static MC88110 Signals

Table 4-2 describes the MC88110 signals that should remain at a constant voltage level for MC88110/MC88410 systems and are not connected to the MC88410. The following paragraphs describe these static signals.

Table 4-2. Static MC88110 Signals

MC88110 Signals	Function	Level
$\overline{\text{AACK}}$	Address acknowledge signal	High
$\overline{\text{ARTRY}}$	Address retry signal	High
$\overline{\text{SHD}}$	Shared signal	Low
PSTAT2–PSTAT0	Processor status signals	Not connected
$\overline{\text{SSTAT0}}$	Snoop status signal	Not connected
$\overline{\text{DBG}}$	Data bus grant signal	Low
$\overline{\text{DBB}}$	Data bus busy signal	High
$\overline{\text{MC}}$	Memory cycle signal	Low

The MC88110 allows the address and data buses to operate independently of each other using split-bus arbitration. However, the MC88110 should not use split-bus arbitration for single-MC88410 or dual-MC88410 configurations. The MC88110 $\overline{\text{AACK}}$ input signal is used to terminate address bus mastership, enabling split-bus arbitration. For MC88110/MC88410 systems, the MC88110 $\overline{\text{AACK}}$ signal must be connected to a pull-up resistor to disable split-bus arbitration.

The MC88110 $\overline{\text{ARTRY}}$ input signal indicates to the current address bus master that it should terminate the transaction and reinitiate the transaction later. The $\overline{\text{TRTRY}}$ signal indicates to the current data bus master that it should terminate the transaction and reinitiate the transaction later. Since the address and data bus should not be split for MC88110/MC88410 systems, the address bus master is also the data bus master and only one of these signals is required. Thus, the $\overline{\text{ARTRY}}$ signal on the MC88110 should be connected to a pull-up resistor and the $\overline{\text{P\_TRTRY}}$ output signal on the MC88410 should be connected to the $\overline{\text{TRTRY}}$ input signal of the MC88110, enabling the MC88410 to retry a transaction when necessary.

The MC88110 cache state logic is implemented as a four-state design, and also supports a three-state model. When operating in the three-state model, all internal cache state transitions are visible on the processor bus to allow the MC88410 to maintain secondary cache coherency (see **2.4 Cache Coherency**). The distinction of whether the three- or four-state model is in use is determined by the status of the MC88110 $\overline{\text{SHD}}$ input signal. The MC88110 $\overline{\text{SHD}}$ signal should be connected to ground so that the MC88410 can track the MC88110 internal cache state transitions.

The PSTAT2–PSTAT0 signals of the MC88110 provide some visibility of the internal CPU status. The MC88410 does not require the information provided by the PSTAT signals and so they should be left unconnected.

The MC88110 $\overline{\text{SSTAT0}}$ signal is asserted when a snoop transaction is a snoop hit. Since the MC88410 uses the processor tag (PTAG) to monitor the contents of the MC88110 data cache, $\overline{\text{SSTAT0}}$ is not needed and therefore should be left unconnected.

The MC88110 $\overline{\text{DBG}}$ signal is used by an external bus arbiter to grant data bus mastership in response to a data bus request. The MC88110 assumes data bus mastership if $\overline{\text{DBG}}$ is

asserted and the data bus is not already busy ($\overline{DBB}$ is negated). Since $\overline{AACK}$ should be connected high, disabling split-bus transactions, $\overline{DBG}$ can be connected to ground to always grant the data bus to the MC88110. The $\overline{DBB}$ signal should be connected to a pull-up resistor to guarantee that the MC88110 always has a qualified data bus grant ($\overline{DBG}$ asserted and $\overline{DBB}$ negated).

The MC88110 $\overline{MC}$ signal indicates that the transaction is actually transferring data. Since the MC88410 does not require this information, the $\overline{MC}$ signal should be left unconnected. Note that the MC88410 interprets all MC88110 transactions as data transfers.

4.2.3 RAM Interface Signals

Although the MC88410 does not contain the data bus, it does control data transfer through the control of the MCM62110 array. The RAM interface consists of the R_A16–R_A0, $\overline{RWE7}$–$\overline{RWE0}$, $\overline{PIE}$, $\overline{POE}$, $\overline{SIE}$, and $\overline{SOE}$ signals. The R_A16–R_A0, $\overline{RWE7}$–$\overline{RWE0}$, $\overline{POE}$, and $\overline{PIE}$ signals are involved in processor bus transactions and are part of the processor bus interface. Refer to **Section 5 System Bus Interface** for information about the $\overline{SIE}$ and $\overline{SOE}$ signals that are involved in system bus transactions.

The R_A16–R_A0 signals provide the secondary cache with the address of the transaction. For processor transactions, the R_A16–R_A0 signals are driven to the secondary cache in the clock after the address is driven on the processor bus (P_A31–P_A0). For processor bus burst transactions, the MC88410 automatically increments the address to the MCM62110 array.

The $\overline{RWE7}$–$\overline{RWE0}$ signals allow individual bytes of the 64-bit word to be written into the secondary cache. The $\overline{RWE7}$–$\overline{RWE0}$ signals are asserted or negated during system or processor data bus mastership depending upon the transaction. The $\overline{POE}$ signal drives data from the secondary cache to the MC88110 D63–D0 signals and has similar timing to R_A16–R_A0. The $\overline{PIE}$ signal enables the MCM62110 array to latch data from the MC88110 D63–D0 signals and has identical timing to R_A16–R_A0.

4.3 PROCESSOR BUS ARBITRATION

Bus arbitration is the protocol by which a device becomes the bus master. The MC88110 requires an arbitration protocol in which an external arbiter controls bus arbitration and the MC88110 requests mastership of the bus from the arbiter to perform an external access.

The MC88410 includes arbitration circuitry on-chip as well as provisions for external arbitration. The MC88410 on-chip arbitration circuitry can be used in a configuration with a single MC88410, while external arbitration circuitry is required in a dual-MC88410 configuration. The arbitration circuitry to be used is determined by the SD1/ARBEN signal at reset. When the SD1/ARBEN signal is asserted at reset, the MC88410 on-chip arbiter is enabled. When the SD1/ARBEN signal is negated at reset, the MC88410 on-chip arbiter is disabled. For more information about the SD1/ARBEN signal, refer to **5.9 Reset Operation.**

4.3.1 Processor Bus Arbitration Signals

The bus arbitration signals of the MC88410 are listed in Table 4-3. The CS and CSP signals are used to divide the address space between MC88410 devices in dual-MC88410 configurations.

Table 4-3. Processor Bus Arbitration Signals

Function	Signals	Type
Processor bus request	$\overline{\text{P\_BR}}$	Output
Processor bus grant	$\overline{\text{P\_BG}}$	I/O
Processor address bus busy	$\overline{\text{P\_ABB}}$	I/O
Chip select	CS	Input
Chip select polarity	CSP	Input

The $\overline{\text{P\_BR}}$ signal is an output from an MC88410 requesting mastership of the processor bus to an external arbiter.

The $\overline{\text{P\_BG}}$ signal is both an input and an output of the MC88410. The $\overline{\text{P\_BG}}$ signal is an input when external bus arbitration is used (SD1/ARBEN is negated at reset) and an output when the MC88410 on-chip arbitration is used (SD1/ARBEN is asserted at reset). As an input signal, $\overline{\text{P\_BG}}$ is asserted by the external arbiter to indicate to the MC88410 that it has been granted mastership of the bus. As an output signal (from the MC88410 to the MC88110), $\overline{\text{P\_BG}}$ is used to grant the bus to the MC88110.

The $\overline{\text{P\_ABB}}$ signal is an input when the MC88410 arbitrates for the processor bus and is used to qualify its $\overline{\text{P\_BG}}$. The MC88410 receives a qualified bus grant when $\overline{\text{P\_BG}}$ is asserted and $\overline{\text{P\_ABB}}$ is negated as inputs. The MC88410 asserts the $\overline{\text{P\_ABB}}$ signal as an output to take mastership of the bus. The $\overline{\text{P\_ABB}}$ signal should be connected to pull-up resistors to keep the signal negated when no devices are driving the signal. For all timing diagrams the $\overline{\text{P\_ABB}}$ signal is shown with the assumption that pull-up resistors are being used.

The CS signal is used to enable the MC88410. The CSP signal determines if an MC88410 is selected when the CS signal is connected high or when it is connected to ground. For example, if the CSP and CS signals are connected high, the MC88410 is selected.

In single-MC88410 configurations, the CS signal should be grounded and the CSP configuration signal should not be driven during reset. There is an internal pull up on the CSP signal that drives the chip select polarity into the proper state so that the MC88410 is always chip selected.

In dual-MC88410 configurations, the CS signal allows the address space to be divided evenly between both MC88410s servicing a single MC88110. If the CS signals on both the MC88410 devices are connected to a single processor address bit, the CS signal determines which MC88410 is selected. The CSP signal should be driven to opposite states on the two

devices at reset to provide mutually exclusive MC88410 chip selects based on the chosen address line.

4.3.2 Processor Bus Arbitration Protocol

The MC88110 is parked when its $\overline{BG}$ input signal is asserted whether or not processor is requesting the bus. If the $\overline{BG}$ signal remains asserted until an internal bus request occurs, the MC88110 completes the arbitration sequence without any overhead and can begin the transaction without asserting the $\overline{BR}$ signal. Thus, bus parking provides a performance advantage because bus accesses occur without any delay in the arbitration sequence. The MC88410 on-chip arbiter leaves the MC88110 parked on the bus until the MC88410 requires bus mastership.

Figure 4-4 shows a timing diagram of the MC88410 processor bus arbitration signals assuming that the external arbiter uses bus parking. Initially, the MC88410 is the bus master and performs a primary cache invalidate transaction. The MC88410 begins the transaction in clock 1 by asserting $\overline{P\_TS}$. In clock 2, the external arbiter parks the MC88110 on the processor bus by asserting the $\overline{BG}$ input signal of the MC88110. During clock cycles 4 and 5, the $\overline{BG}$ signal continues to be asserted and the MC88110 stays parked on the bus, even though it does not use the bus.

In clock 6, the MC88110 initiates a transaction by driving the address and control information and asserting the $\overline{P\_TS}$ and $\overline{P\_ABB}$ signals. The $\overline{P\_ABB}$ signal is asserted to indicate that the address bus is in use. When a transaction is complete, the $\overline{P\_ABB}$ signal is negated. However, the $\overline{P\_ABB}$ signal remains asserted after the transaction is terminated if the MC88110 is immediately initiating another transaction and it is parked at the time that the initial transaction is normally terminated. Since $\overline{DBG}$ should be connected to ground, it is asserted and data bus arbitration is not necessary.

If no other device is requesting mastership of the bus, the MC88110 $\overline{BG}$ signal remains asserted by the external arbiter and the MC88110 remains parked on the bus. Otherwise, $\overline{BG}$ is negated and an alternate master can take control of the processor bus.

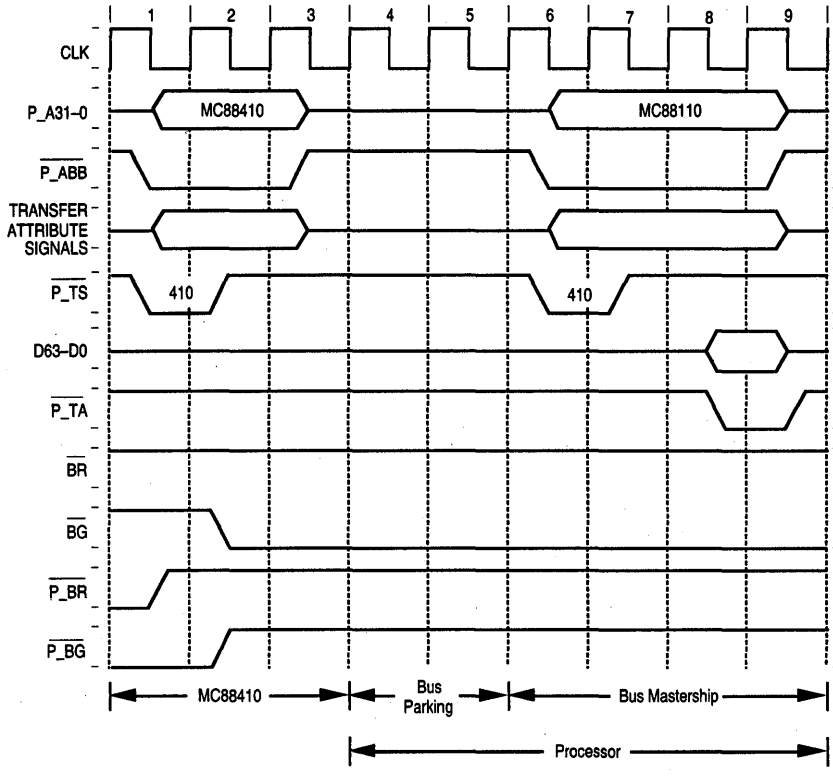


Figure 4-4. Bus Parking by External Arbiter

4.3.3 MC88410 On-Chip Processor Bus Arbitration

The MC88410 on-chip arbitration refers to the internal logic designed into the MC88410 that provides bus arbitration for the processor bus. When the MC88410 on-chip arbitration mode is selected, the on-chip arbiter is activated and it assumes that a single MC88410 and a single MC88110 are the only potential masters on the bus. The MC88410 on-chip arbiter leaves the MC88110 parked on the bus until the MC88410 requires bus mastership to perform a primary cache invalidate or a primary cache DMA invalidate transaction. In this case, the MC88410 on-chip arbiter negates $\overline{\text{P\_BG}}$ to grant itself mastership of the processor bus.

To activate the MC88410 on-chip arbiter in the single-MC88410 configuration, the SD1/ARBEN signal of the MC88410 should be asserted at reset. Since the MC88410 on-chip arbiter does not use the bus request signals of the MC88410 ($\overline{\text{P\_BR}}$) or MC88110 ($\overline{\text{BR}}$), they should be left unconnected.

At the start of a transaction using MC88410 on-chip arbitration, the following four scenarios can exist on the processor bus:

Scenario 1	The MC88110 is parked on the processor bus and it begins a transaction.
Scenario 2	The MC88110 is performing a transaction and the MC88410 retries the MC88110 to gain mastership of the processor bus and perform a primary cache invalidate transaction.
Scenario 3	The MC88410 is performing a primary cache invalidate transaction and the MC88110 is parked by the MC88410 on the processor bus.
Scenario 4	The MC88110 is parked on the processor bus and the MC88410 initiates a primary cache invalidate transaction.

Figure 4-5 illustrates scenarios 1 through 3. It shows the transfer of the processor bus mastership from the MC88110 to the MC88410.

Scenario 1 is shown in clock cycles 1 and 2. In clock 1, $\overline{\text{P\_BG}}$ (which is connected to the MC88110 $\overline{\text{BG}}$ input signal for on-chip arbitration) is asserted by the MC88410 and the MC88110 is parked on the bus. Note that the MC88410 on-chip arbiter controls the assertion of $\overline{\text{P\_BG}}$. In clock 2, the MC88110 asserts $\overline{\text{P\_TS}}$ to indicate the start of a new transaction and drives the address and control information on the processor bus.

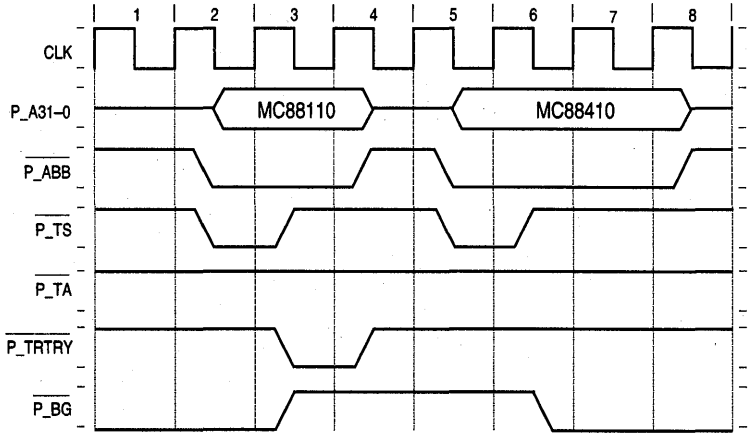


Figure 4-5. Bus Mastership Transfer from MC88110 to MC88410

Scenario 2 is shown in clock cycles 3 through 5. The MC88410 retries the MC88110 transaction in clock 3 to perform a primary cache invalidate transaction and the processor immediately gives up the bus. Note that the MC88410 negates $\overline{\text{P\_BG}}$ to grant itself the bus. At the end of clock 4, neither device owns the bus. In clock 5, the MC88410 asserts $\overline{\text{P\_ABB}}$ to take mastership of the processor bus, asserts $\overline{\text{P\_TS}}$, and drives the address. In clock 6, the MC88410 asserts the $\overline{\text{P\_BG}}$ signal to park the MC88110 on the processor bus. However, the MC88110 cannot take control of the bus and begin a new transaction until it gets a qualified bus grant. A qualified bus grant is defined as $\overline{\text{P\_BG}}$ asserted and $\overline{\text{P\_ABB}}$ negated.

Scenario 3 (clocks 6 through 8) shows the transfer of bus mastership from the MC88410 to the MC88110. The MC88110 does not receive a qualified bus grant in clock 6 even though the $\overline{\text{P\_BG}}$ signal is asserted. In clock 7, the MC88110 performs a data cache tag lookup to determine if it needs to perform a copyback transaction to enforce cache coherency (see **2.4 Cache Coherency**). In clock 8, the MC88110 receives a qualified bus grant when the $\overline{\text{P\_BG}}$ signal is asserted by the MC88410 and $\overline{\text{P\_ABB}}$ is negated. The MC88110 can begin a new transaction in clock 9, if necessary.

Scenario 4 is shown in Figure 4-6. It describes how the MC88410 obtains the processor bus to perform a primary cache invalidate transaction. Before clock 1 the MC88110 was parked on the bus. During clock 1 the MC88410 negates $\overline{\text{P\_BG}}$ and the MC88110 is no longer parked on the bus. In clock 2, the MC88410 asserts $\overline{\text{P\_TS}}$ for one clock cycle and begins its transaction. Notice in clock 3 that $\overline{\text{P\_BG}}$ is asserted again. However, the MC88110 cannot take control of the bus and begin a new transaction until it gets a qualified bus grant ($\overline{\text{P\_ABB}}$ negated). The MC88410 transaction completes in clock 5. At this point, the $\overline{\text{P\_BG}}$ signal remains asserted to the MC88110 and $\overline{\text{P\_ABB}}$ is negated and so the MC88110 can begin a transaction in clock 6, if necessary.

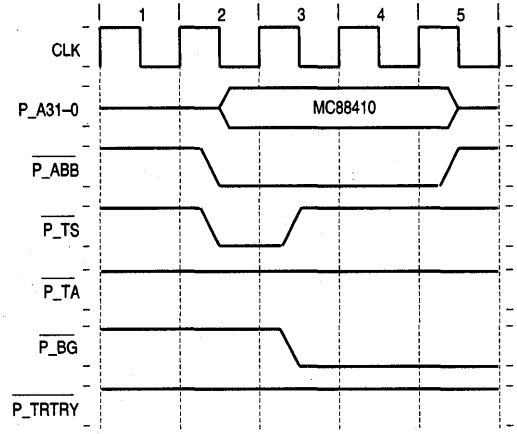


Figure 4-6. Parked MC88110 and MC88410 Bus Grant

4.3.4 External Processor Bus Arbitration

When the external arbiter mode is selected, the MC88410 on-chip arbiter is disabled. Therefore, external arbitration logic must be designed to accept the arbitration signals from both the MC88410s and the MC88110 on the processor bus. However, it should provide bus grant to only one device at a time. In configurations using external arbitration, the MC88410 should have higher arbitration priority than the processor to allow it to perform a primary cache invalidate. The following paragraphs describe the external arbitration interface and timing.

In dual-MC88410 configurations, the external arbitration logic must be used to control processor bus mastership. The SD1/ARBEN signal is negated at reset to disable MC88410

arbitration. The bus request and bus grant signals on the three devices (MC88110 and two MC88410s) are connected to the external arbitration circuitry as shown in Figure 4-3. The $\overline{\text{P\_BR}}$ is the bus request signal from the MC88410 to the external arbiter. Similarly, $\overline{\text{BR}}$ is the bus request signal from the MC88110 to the external arbiter. The external arbiter then asserts $\overline{\text{P\_BG}}$ to grant mastership to one of the two MC88410s or $\overline{\text{BG}}$ to grant mastership to the MC88110.

When the MC88410 needs mastership of the processor bus, it asserts $\overline{\text{P\_BR}}$ and continues to assert $\overline{\text{P\_BR}}$ until it is granted mastership of the bus and the bus becomes available. The external arbiter grants mastership of the bus to the potential master by asserting the $\overline{\text{P\_BG}}$ signal. Because the $\overline{\text{P\_ABB}}$ signal is asserted by the current master to indicate address bus mastership, the potential master determines that the bus is available when the $\overline{\text{P\_ABB}}$ signal is negated. A qualified bus grant is defined as $\overline{\text{P\_BG}}$ asserted and $\overline{\text{P\_ABB}}$ negated (as an input). The potential master does not assume address bus mastership until it receives a qualified bus grant.

When the MC88410 receives a qualified bus grant, it negates the $\overline{\text{P\_BR}}$ output signal and asserts the $\overline{\text{P\_ABB}}$ signal. At the same time, the MC88410 drives the address for the requested access onto the address bus and asserts the $\overline{\text{P\_TS}}$ signal to indicate the start of a new transaction.

The timing diagram in Figure 4-7 illustrates external arbitration. In clock 1, the MC88410 requests the processor bus by asserting the $\overline{\text{P\_BR}}$ signal. The bus is granted to the MC88410 on the rising edge of clock 2 when the external arbiter asserts $\overline{\text{P\_BG}}$. The MC88410 receives a qualified bus grant in clock 2 since $\overline{\text{P\_BG}}$ is asserted and $\overline{\text{P\_ABB}}$ is negated. It assumes bus mastership by asserting the $\overline{\text{P\_ABB}}$ signal in clock 2. At the same time, the MC88410 negates the $\overline{\text{P\_BR}}$ output signal, asserts the $\overline{\text{P\_TS}}$ signal, and drives the address for the requested access onto the processor bus.

When designing the external bus arbitration logic, it is important to note that the MC88110 may assert $\overline{\text{BR}}$ but never use the bus (assert $\overline{\text{P\_ABB}}$) after it receives the qualified bus grant. For example, this situation would occur if the MC88110 asserts $\overline{\text{BR}}$ to perform a replacement copyback transaction but an MC88410 performs a primary cache invalidate transaction for that cache line before the MC88110 is granted the bus. In this case, the copyback operation would be unnecessary and the MC88110 does not assert $\overline{\text{P\_ABB}}$.

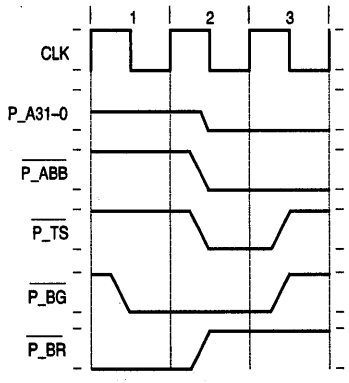


Figure 4-7. External Arbitration Timing

4.4 DATA TRANSFER MECHANISM

The following paragraphs describe the signals used in the transfer of data between the MC88410 and the MC88110 on the processor bus. The data transfer protocol and examples of the relative signal relationships for the different types of transactions are described.

4.4.1 Data Transfer Mechanism Signal Overview

The signals that implement the data transfer mechanism for the MC88410 are classified as data transfer signals, transfer attribute signals, and transfer control signals. The transfer attribute signals are summarized in Table 4-4. The $\overline{\text{P\_WT}}$ and P_CL processor bus signals do not have corresponding system bus signals.

Table 4-4. Processor Bus Transfer Attribute Signal Summary

Signal Name	Signal	Asserted	Negated
Read/Write	P_R/$\overline{W}$	Read	Write
Lock	$\overline{P\_LK}$	Transaction is one of two atomic transactions. Enables the MC88110 to lock the processor bus	Transaction is not part of an atomic sequence
Cache inhibit	$\overline{P\_CI}$	The address of the transaction in progress is not cached in the primary or secondary cache	The address of the transaction in progress may be cached in the primary and/or secondary cache
Write-through	$\overline{P\_WT}$	Write-through memory update mode	Write-back memory update mode
User page attributes	$\overline{P\_UPA1}$–$\overline{P\_UPA0}$	Reflect the state of the UPA1–UPA0 bits on the processor interface	Reflect the state of the UPA1–UPA0 bits on the processor interface
Transfer burst	$\overline{P\_TBST}$	Burst transaction	Single-beat transaction
Transfer size*	P_TSIZ1–P_TSIZ0	See Table 3-3	See Table 3-3
Transfer code	P_TC3–P_TC0	See Table 3-4	See Table 3-4
Invalidate	$\overline{P\_INV}$	This signal is broadcast to the MC88110 to invalidate the data cache line	No need to have snooping MC88110s invalidate the data cache line
Cache line	P_CL	Indicates data cache line 1	Indicates data cache line 0
Global	$\overline{P\_GBL}$	Data being transferred is global data	Data being transferred is local data

The * (asterisk) indicates that the signal should be ignored for burst cycles.

4.4.2 Data Transfer Transaction Summary

The processor bus interface initiates transactions in response to processor bus transactions and system bus snooping transactions. Data is transferred on the processor bus in either single-beat transactions or burst transactions. Transactions with $\overline{P\_TBST}$ negated are single-beat processor bus transactions and transactions with $\overline{P\_TBST}$ asserted are processor bus burst transactions. Single-beat processor transactions can result in burst system bus transactions ($\overline{S\_TBST}$ asserted).

Table 4-5 summarizes the state of transfer attribute and control signals for single-beat and burst processor bus transactions.

Table 4-5. Processor Bus Transaction Attribute and Control Signals

Transaction	P_R/$\overline{W}$	$\overline{P\_TBST}$	$\overline{P\_CI}$	$\overline{P\_WT}$	$\overline{P\_INV}$	$\overline{P\_LK}$	$\overline{P\_GBL}$	P_TSIZ1–0	P_TC3–0	P_CL
Single-Beat										
Single-beat read	R	N	MMU	MMU	N	N	MMU	b,h,w,d	I/D, U/S	Invalid
Locked read	R	N	MMU	MMU	A	A	MMU	b,w	D,U/S	Invalid
Table search	R	N	N	A	N	N	N	w	I/D, TSO	Invalid
Allocate load	R	N	MMU	MMU	A	N	MMU	h	TFA	Valid
Single-beat write	W	N	MMU	MMU	A	N	MMU	b,h,w,d	D,U/S	Invalid
Locked write	W	N	MMU	MMU	A	A	MMU	b,w	D,U/S	Invalid
Write-through	W	N	MMU	A	A	N	MMU	b,h,w,d	D,U/S	Invalid
Processor invalidate	W	N	N	N	A	N	A	b,h,w,d	D,U/S	Valid
Primary cache invalidate†	W	N	x	x	A	N	A	x	x	x
Burst										
Read miss line fill	R	A	N	N	N	N	MMU	d	I/D, U/S	Valid
Data cache read-with intent-to-modify	R	A	N	N	A	N	MMU	d	D,U/S	Valid
Touch load	R	A	MMU	MMU	N	N	MMU	b	TFA	Valid
Replacement copyback	W	A	N	N	A	N	N	d	D,S	Valid
Snoop copyback	W	A	N	N	A	N	N	d	SCB	Valid
Flush copyback	W	A	N	N	A	N	N	d	D,S	Valid
Flush load	W	A	N	N	A	N	N	w	TFA	Valid
Processor DMA invalidate†	W	A	x	x	x	x	A	x	x	x

b= Byte
h= Half word
w = Word
d = Double word
R = Read
W = Write
A = Asserted
N = Negated
x = Don't care

I = Instruction access
D = Data access
S = Supervisor access
U = User access
TFA = Touch, flush, or allocate access
TSO = Table search operation
SCB = Snoop copyback operation
MMU = Determined by MC88110 MMU
† = Transaction initiated by MC88410

This section assumes that processor transactions hit in the secondary cache and are not propagated to the system interface. For information regarding system bus transactions, refer to **Section 5 System Bus Interface.** For information regarding processor bus transactions and the effect of cache coherency considerations on data transfer, refer to **Section 2 Secondary Cache Operation.**

In Table 4-5, note that the $\overline{\text{P\_UPA1}}$–$\overline{\text{P\_UPA0}}$ signals are not shown since they are always determined by the MC88110 memory management unit (MMU).

Note that since the $\overline{\text{MC}}$ signal of the MC88110 is not connected to the MC88410, all processor transactions (including invalidates) cause data to be transferred to the secondary cache. In the case of a primary cache invalidate transaction, the data is marked invalid in the main tag (MTAG) even though data is written to the secondary cache. The $\overline{\text{P\_INV}}$ signal is asserted by the MC88110 to indicate that the MC88410 should perform a system invalidate broadcast, if necessary, to maintain cache coherency.

The $\overline{\text{P\_INV}}$ signal is asserted by the MC88410 to indicate to the MC88110 that it should invalidate its corresponding primary data cache line. The $\overline{\text{P\_INV}}$ signal is asserted for all write transactions, locked read transactions, and allocate load transactions.

4.4.3 Processor Single-Beat Transactions

Accesses that occur directly on the external bus independently of the data cache (regardless of a cache hit) cause single-beat transactions to occur. Transactions in Table 4-5 with $\overline{\text{P\_TBST}}$ negated are single-beat transactions. All single-beat transactions have similar timing characteristics; the differences between the transactions are determined by the transfer attribute signals that are asserted/negated.

Figure 4-8 shows the relative timing of the data transfer signals during a single-beat transaction in the case of a secondary cache hit. Before a single-beat transaction begins, the MC88110 arbitrates for bus mastership, if it is not parked, and becomes the processor bus master.

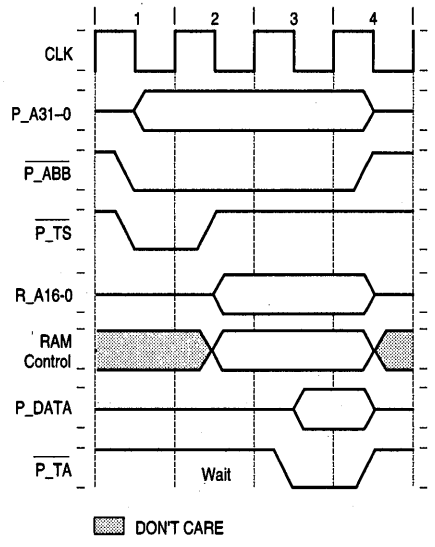


Figure 4-8. Single-Beat Transaction – Fastest Case

In Figure 4-8, the processor drives the address signals with the physical address of the access off the rising edge of clock 1 and at the same time asserts the appropriate attribute and control signals for the type of single-beat transaction being performed. The MC88410 latches the address and control information on the rising edge of clock 2. During clock 2, the MC88410 decodes the access information, determines whether there is a hit or a miss in the secondary cache, and drives the appropriate RAM address and control signals to the MCM62110 array. The MC88410 keeps the $\overline{\text{P\_TA}}$ signal negated for at least one clock cycle in order to allow sufficient time to process the transaction.

To indicate the status of the transaction to the processor, the MC88410 either asserts or negates $\overline{\text{P\_TA}}$. In the case of a cache hit, $\overline{\text{P\_TA}}$ is asserted (clock 3 in Figure 4-8) two clocks after the address is driven on the processor address bus . If there is a cache miss, $\overline{\text{P\_TA}}$ is negated until the clock following system bus data transfer termination.

While $\overline{\text{P\_TA}}$ is negated, the processor waits and continuously drives the address (and data for write transactions) on the processor bus until $\overline{\text{P\_TA}}$ is asserted. During the clock cycle after the assertion of $\overline{\text{P\_TA}}$, the address lines are three-stated and $\overline{\text{P\_ABB}}$ is negated by the MC88110.

4.4.3.1 Processor Single-Beat Read Transaction

Single-beat read transactions include cache-inhibited accesses and MMU descriptor fetches that occur during table search operations. The $\overline{\text{P\_INV}}$ signal is asserted only for locked (**xmem**) reads and allocate load transactions (which are intent-to-modify). During a single-beat read transaction, the MC88110 reads a byte, half word, word, or double word from the memory system.

To perform a single-beat read transaction, the MC88110 first arbitrates for mastership of the processor bus if it is not the current bus master. The MC88110 then drives the address onto the address bus, asserts or negates the appropriate transfer attribute signals, and asserts $\overline{\text{P\_TS}}$ as shown in Figure 4-9.

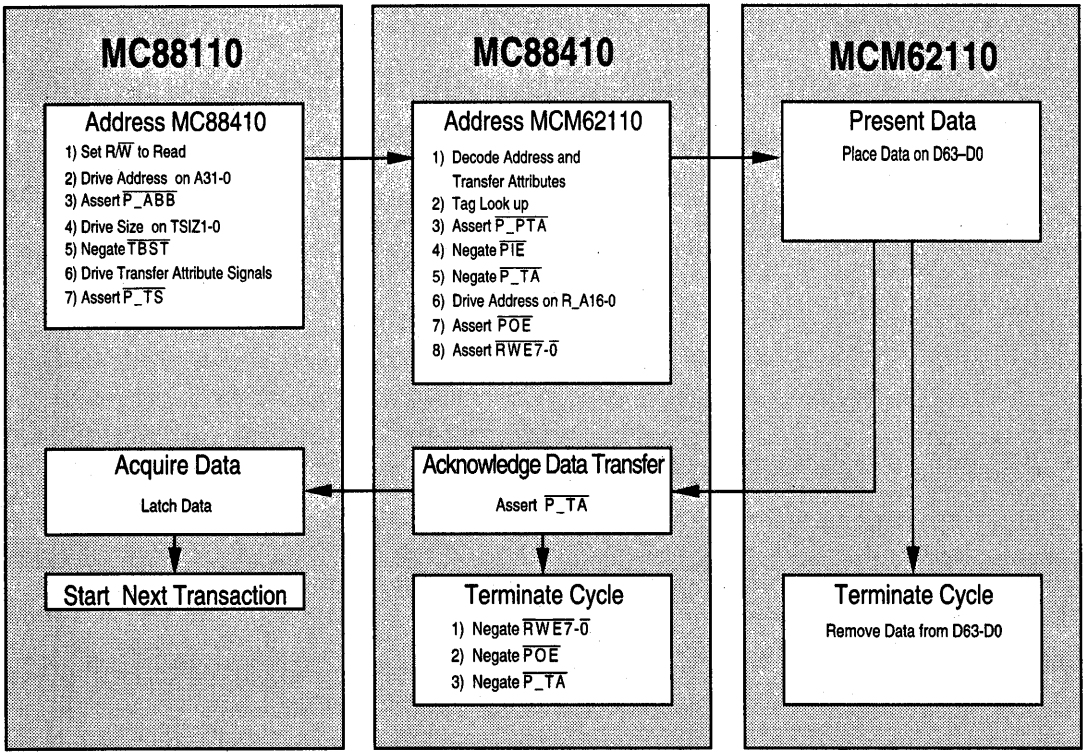


Figure 4-9. Single Beat Read Transaction Flow – Secondary Cache Hit

Once the MC88410 latches the address and drives it to the MCM62110 array, the MCM62110 array supplies the requested data on the appropriate D63–D0 signals within the required setup and hold times while the MC88410 asserts $\overline{\text{P\_TA}}$. If the transaction had missed in the secondary cache, the MC88410 inserts wait states by negating $\overline{\text{P\_TA}}$ until the data is available. For timing diagrams of transactions that miss in the secondary cache, refer to **Section 5 System Bus Interface**.

Figure 4-10 shows the relative timing for single-beat read transactions that hit in the secondary cache. The processor drives the address signals with the physical address of the access off the rising edge of clock 2 and at the same time asserts the appropriate attribute and control signals for the single-beat read transaction. The MC88410 latches the address and control information on the rising edge of clock 3 and asserts $\overline{\text{P\_PTA}}$ (not

shown). The MC88410 always asserts $\overline{\text{P\_PTA}}$ one clock after the processor asserts $\overline{\text{P\_TS}}$. In order to perform a cache tag lookup and to drive the RAM address and control signals to the MCM62110 array, the MC88410 inserts a wait state for one clock by negating $\overline{\text{P\_TA}}$. If a cache hit occurs, the data is transferred to the processor in clock cycle 4 and $\overline{\text{P\_TA}}$ is asserted to indicate the end of the transaction. During a cache miss, $\overline{\text{P\_TA}}$ is negated until the data is available from the system bus.

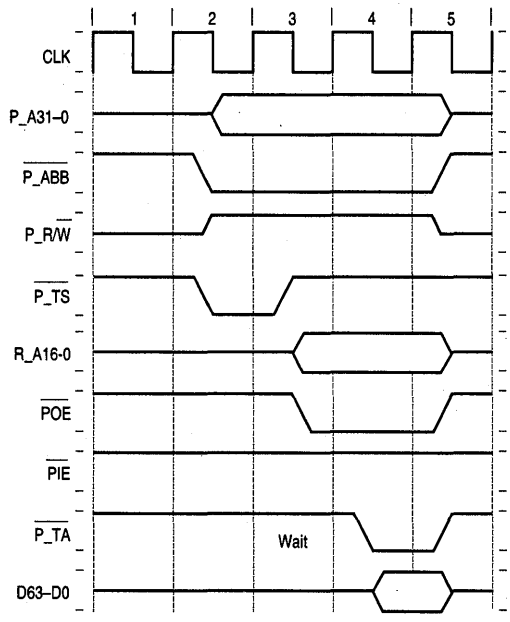


Figure 4-10. Single-Beat Read Hit Timing

4.4.3.2 Processor Single-Beat Write Transaction

Single-beat write transactions consist of cache-inhibited write transactions, write-through transactions, primary cache invalidate, and processor invalidate transactions. In the write-through mode, write transactions update external memory every time the primary cache line is modified. Write transactions that hit the secondary cache are written into it and to the system interface. A primary cache invalidate transaction occurs when there is a system bus write- or read-with-intent-to-modify which affects a line in the primary cache that is marked unmodified in the MC88110. Processor invalidate transactions occur when the MC88110 first writes to a primary cache line. Processor invalidate transactions always hit the secondary cache and leave the MC88410's line in a modified state (a system invalidate precedes the state change if necessary).

During a single-beat write transaction, the MC88110 transfers a byte, half word, word, or double word to the MC88410. To perform a single-beat write transaction, the MC88110 first becomes the processor bus master if it is not already the bus master. The MC88110

then asserts P_TS, drives the address, and asserts or negates the appropriate attribute and control signals. All write transactions cause P_INV to be asserted.

Once the MC88410 drives the address to the MCM62110 array, the MCM62110 latches the data from the appropriate D63-D0 signals in the following clock. In the event of a secondary cache miss, the MC88410 inserts wait states by negating P_TA until the data is latched on the system bus. The MC88110 continues to drive the address and data on the processor bus until P_TA is asserted (or the transaction is otherwise terminated). Figure 4-11 shows the transaction flow for a single-beat write transaction.

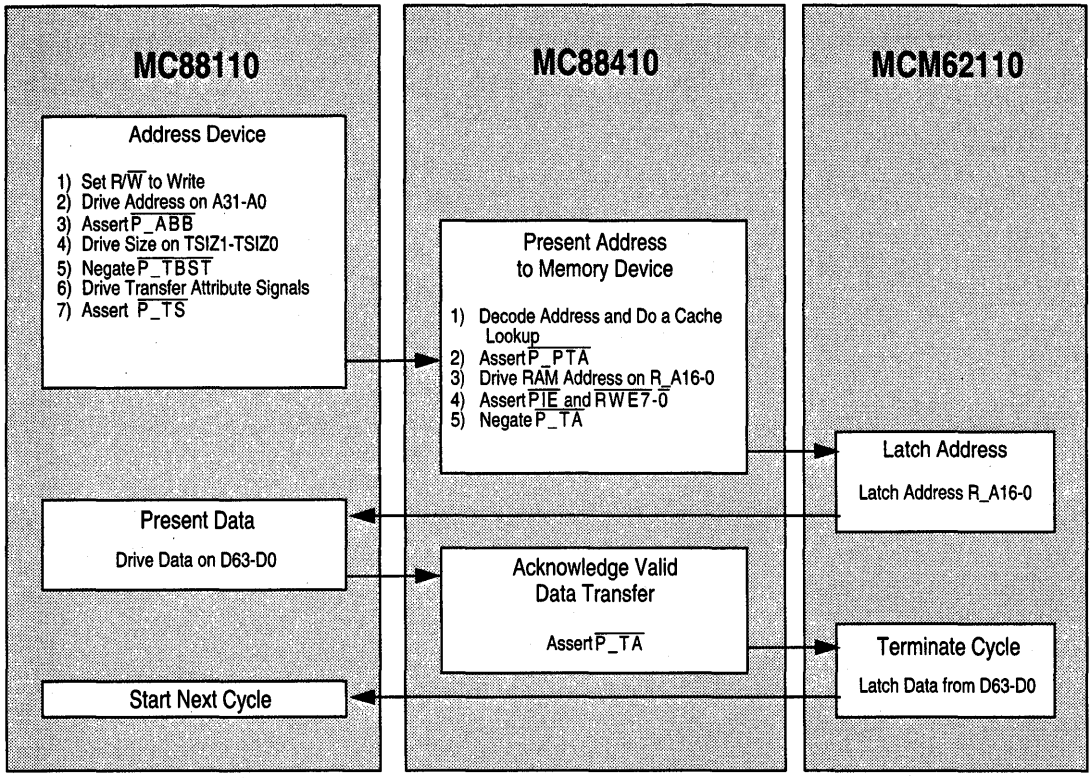


Figure 4-11. Single-Beat Write Transaction Flow

Figure 4-12 shows the relative timing for single-beat write transactions. The processor drives the address signals with the physical address of the access off the rising edge of clock 2 and at the same time asserts the appropriate attribute and control signals for the single-beat write transaction. The MC88410 latches the address and control information on the rising edge of clock 3 and asserts P_PTA (not shown). The MC88410 always asserts P_PTA one clock after the processor asserts P_TS. In order to perform a cache tag lookup and to drive the RAM address and control signals to the MCM62110 array, the MC88410

inserts a wait state for one clock by negating $\overline{\text{P\_TA}}$. When a secondary cache hit occurs, as shown in Figure 4-12, the data is written into the MCM62110 array in clock cycle 4, when $\overline{\text{P\_TA}}$ is asserted by the MC88410 to terminate the transaction. In the case of a secondary cache miss, the memory update policy determines if data is written into the secondary cache (see **Section 2 Secondary Cache Operation**).

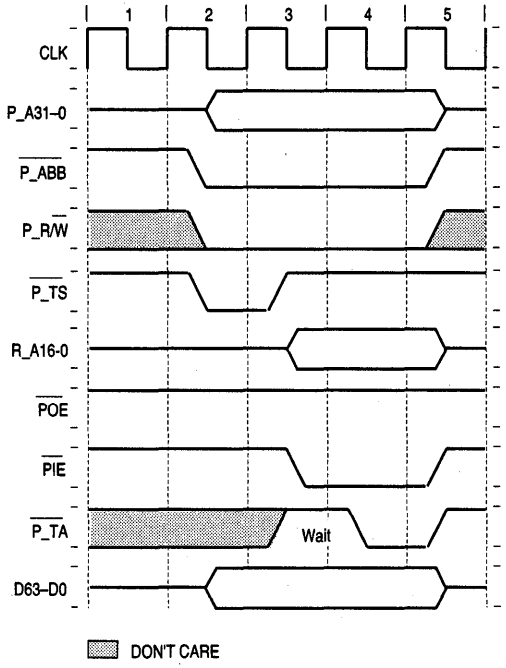


Figure 4-12. Single – Beat Write Hit Timing

4.4.4 Primary Cache Invalidate and DMA Invalidate

The MC88410 coherency hardware provides address snooping on the system interface. If the data to be modified is contained in the primary (MC88110) data cache, the MC88410 performs a primary cache invalidate transaction in order for the MC88110 to invalidate its cache line. The primary cache invalidate transaction is used to maintain vertical cache coherency (see **Section 2 Secondary Cache Operation**). To perform a primary cache invalidate transaction, the MC88410 obtains mastership of the processor bus through normal arbitration (the MC88410 should have higher arbitration priority than the processor).

Once the MC88410 becomes the processor bus master, it drives the address of the data to be invalidated and asserts both the $\overline{\text{P\_GBL}}$ and $\overline{\text{P\_INV}}$ signals. The primary cache invalidate transaction consists of an address transaction only (no data is transferred). It does not require normal transaction termination. This is different from the invalidate transactions of either the MC88110 or the MC88410 system interface, which both rely on

the assertion of the $\overline{TA}$ signal to confirm that their invalidate transactions have completed successfully.

If the address hits on a modified primary cache line, the MC88110 asserts $\overline{SSTAT1}$ (snoop hit to a modified line, connected to $\overline{P\_ARTRY}$), takes control of the bus, and performs a copyback transaction . If the snoop hits on an unmodified line, the MC88110 does not retry the primary cache invalidate transaction. The MC88410 takes the absence of an address retry (after a two-clock snoop delay) to indicate that the primary cache line was invalidated by the snoop.

The MC88410 can also issue a primary cache DMA invalidate transaction due to a direct memory access snoop on the system interface (see **5.7.9 System DMA Invalidate**). This occurs when a DMA controller transfers data to main memory and overwrites data that is cached in the primary cache. The MC88410 performs a primary cache invalidate transaction but asserts the $\overline{P\_TBST}$ signal instead of negating it. This causes the MC88110 to invalidate the primary cache tag entry without copying the data back, even if it was modified.

Figure 4-13 shows the timing of a primary cache invalidate transaction that hits on an unmodified primary cache line. The MC88110 completes a transaction in clock 2. During clock 3, the MC88410 takes mastership of the bus and begins the primary cache invalidate transaction. The $\overline{P\_TS}$ signal is asserted for one clock cycle to begin the transaction. The address and control signals are driven on the processor bus and the $\overline{P\_INV}$ and $\overline{P\_GBL}$ signals are asserted. During clock 4, the MC88110 performs a cache tag lookup to see if the address hits on an unmodified or modified line. In this example, the cache hit is to an unmodified line so no copyback operation is needed. Since $\overline{P\_ARTRY}$ is not asserted, the transaction is terminated in clock 6. If the primary cache invalidate transaction had hit a modified primary cache line, the $\overline{P\_ARTRY}$ signal would have been asserted by the MC88110 and a copyback transaction would have occurred before it invalidated its cache line.

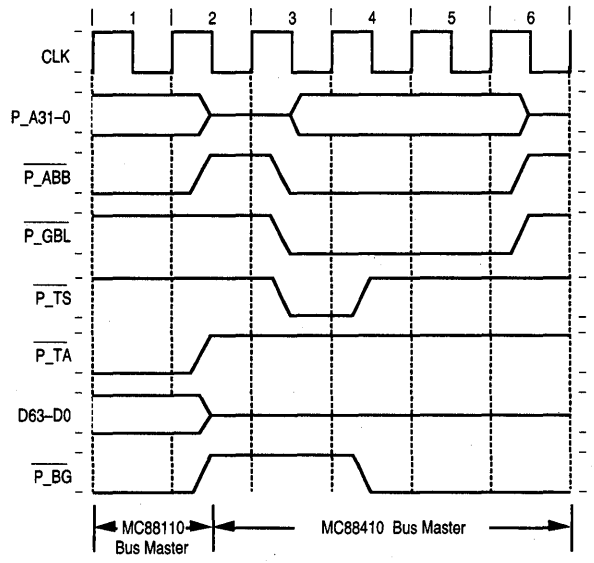


Figure 4-13. Primary Cache Invalidate Timing

4.4.5 Processor Burst Transactions

Transactions in Table 4-5 with P_TBST asserted are burst transactions. Burst transactions perform the transfer of four double words between the processor and the secondary cache and/or the system bus.

Data transfers on the processor interface always start with the double-word address presented by the processor. For read miss line fills, this is the "critical word" address. As a result of critical-word-first ordering, the read miss line fill (or copyback) transaction always begins with the evenly-aligned double word containing the missed word (that is, critical-word-first), followed by the subsequent double word(s) in the cache line, if any. If the double word containing the missed data does not correspond to the first double word in the cache line, the fill operation wraps around and then fills the double word(s) at the beginning of the line. Data is always transferred on the processor bus in critical-word-first order for burst transactions.

To begin a transaction, the processor drives the address of the critical word on the processor bus and asserts P_TBST to indicate a burst transaction. When the MC88410 detects that the transaction is a burst, it internally increments the address of the remaining double words to MCM62110 array. The incremented addresses provided by the processor bus are not used.

When the double-word data is guaranteed to meet the appropriate setup and hold times, the MC88410 asserts P_TA to terminate the beat. At this time, either the address is incremented to be the address for the next beat of the burst or if all four beats have completed successfully, the burst transaction is terminated.

If the data cannot be supplied in the clock cycle after the address is sampled, $\overline{\text{P\_TA}}$ is negated until the data is available. While $\overline{\text{P\_TA}}$ is negated, the processor waits and continuously drives the address on the processor address bus until $\overline{\text{P\_TA}}$ is asserted.

If the transaction terminates with an error, the actions of the processor depend on when the error is detected and the type of transaction being performed.

Figure 4-14 shows the relative timing of the data transfer signals during a burst transaction that hits in the secondary cache. Before a burst transaction begins, the processor arbitrates for the processor bus and becomes the bus master. The processor then drives the address signals with the physical address of the access in clock 1 and at the same time asserts the appropriate attribute and control signals for the type of burst transaction being performed.

In clock 2 the MC88410 continues to negate $\overline{\text{P\_TA}}$, determines whether the transaction hit in the secondary cache, and drives the address to MCM62110 array. In the case of a cache hit, the MCM62110 array latches or drives the data for the first beat of the burst during clock 3. The next three beats of the burst occur during subsequent clock cycles. To indicate the status of each of the four beats of the burst transaction to the processor, the MC88410 either asserts or negates the $\overline{\text{P\_TA}}$ signal.

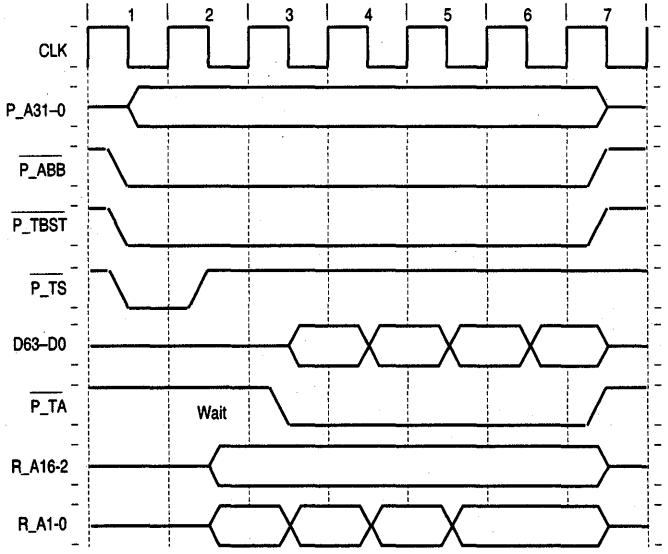


Figure 4-14. Burst Transaction - Fastest Case

4.4.5.1 Processor Burst Read Transaction

Figure 4-15 shows the transaction flow for a burst read transaction. During a burst read transaction, the MC88110 transfers four double words from a secondary cache line to the MC88110.

To perform a burst read transaction, the MC88110 first arbitrates for mastership of the processor bus if it is not parked. The MC88110 then drives the address onto the bus, asserts or negates the appropriate transfer attribute signals, and asserts $\overline{\text{P\_TS}}$ to signal the start of a new transaction. The MC88110 also asserts $\overline{\text{P\_TBST}}$ to signal a burst transaction.

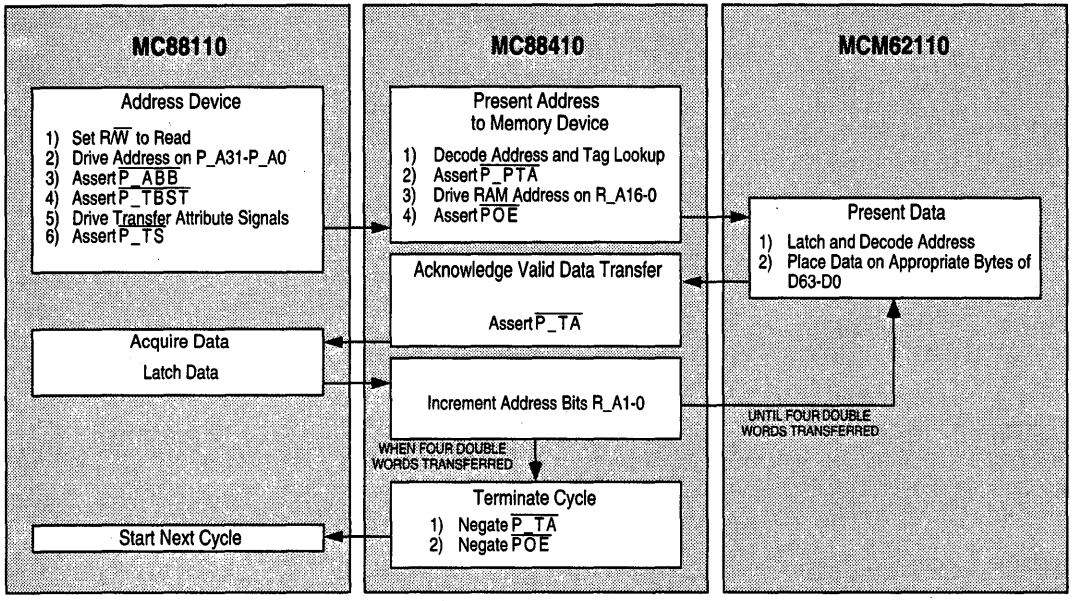


Figure 4-15. Burst Read Transaction Flow

The MC88410 decodes the address and transfer attribute signals and performs a cache tag lookup to determine if there is a secondary cache hit or miss. Then the MC88410 drives the address and control signals to the MCM62110 array and asserts $\overline{\text{P\_PTA}}$ signal. The MC88410 always asserts $\overline{\text{P\_PTA}}$ one clock after the processor asserts $\overline{\text{P\_TS}}$.

Once the MC88410 drives the address to the MCM62110 array, the MCM62110 drives the requested data on the appropriate D63–D0 signals in the following clock if the transaction hits in the secondary cache. In the event of a secondary cache miss, the MC88410 inserts wait states by negating $\overline{\text{P\_TA}}$ until the data is available. For timing diagrams of transactions that miss in the secondary cache, refer to **Section 5 System Bus Interface**.

The timing for a processor burst read transaction that hits in the secondary cache is shown in Figure 4-16. In clock 1, the processor begins the transaction by driving the address on the processor bus, asserting $\overline{\text{P\_ABB}}$, and asserting $\overline{\text{P\_TS}}$ to signal a new transaction. The MC88110 negates P_R/$\overline{\text{W}}$ for the read transaction and asserts $\overline{\text{P\_TBST}}$ signal to indicate a burst transaction. In clock 2, the MC88410 drives the appropriate address and control information to the MCM62110 array. The $\overline{\text{POE}}$ signal is asserted to enable data transfer from MCM62110 array to the processor. To perform a cache tag lookup and to drive the

RAM address and control signals to the MCM62110 array, the MC88410 inserts a wait state in clock 2 by continuing to negate $\overline{\text{P\_TA}}$.

Since the transaction hits in the secondary cache, the MCM62110 array drives the first aligned double word on the data signals (P DATA)in clock 3 and the MC88410 asserts $\overline{\text{P\_TA}}$. At the same time, the MC88410 increments the RAM address to the next double word and asserts $\overline{\text{RWE7}}$–$\overline{\text{RWE0}}$. During each of the following three clock cycles, data for the specified address is placed on the data bus and the address is incremented to reflect the address of the appropriate double word. The address, data, and control signals are three-stated in clock 7, and $\overline{\text{P\_TA}}$ is negated to signal the end of the transaction.

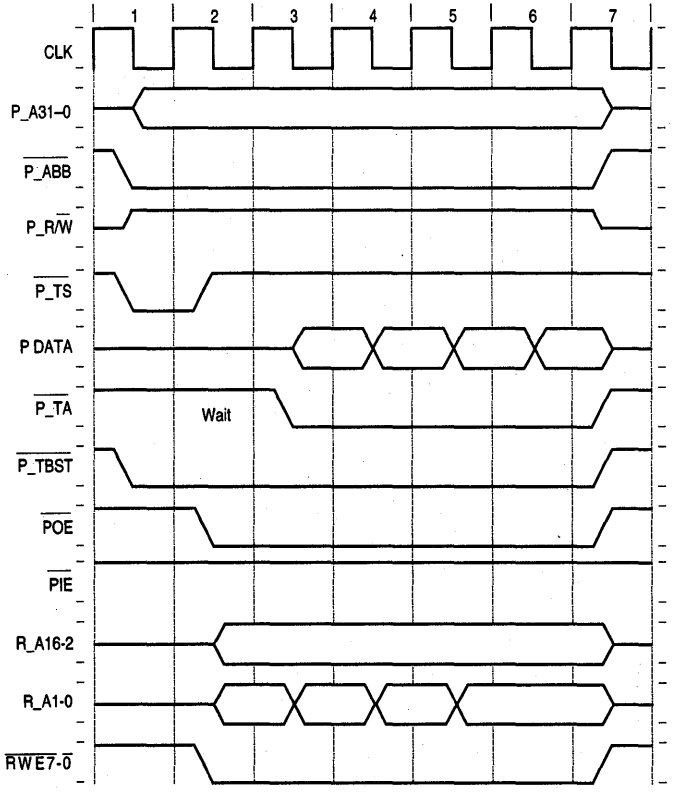


Figure 4-16. Burst Read Hit Timing

4.4.5.2 Processor Burst Write Transaction

During a burst write transaction, the MC88110 transfers four double words from a data cache line to memory.

To perform a burst write transaction, the MC88110 first arbitrates for mastership of the processor bus if the MC88110 is not parked. The MC88110 then drives the address onto the bus, drives the data on the appropriate D63–D0 signals, asserts or negates the appropriate transfer attribute signals, and asserts $\overline{\text{P\_TS}}$ to signal the start of a new transaction. The MC88110 also asserts $\overline{\text{P\_TBST}}$ to signal a burst transaction. The $\overline{\text{P\_INV}}$ signal is asserted for all write transactions.

The MC88410 decodes the address and transfer attribute signals and performs a cache tag lookup to determine if there is a secondary cache hit. Then the MC88410 drives the address and control signals to the MCM62110 array and asserts $\overline{\text{P\_PTA}}$ (not shown). The MC88410 always asserts $\overline{\text{P\_PTA}}$ one clock after the processor asserts $\overline{\text{P\_TS}}$. The MCM62110 array latches the address and the data from the processor bus.

Once the MC88410 drives the address to the MCM62110 array, the MCM62110 latches the data from the appropriate D63–D0 signals in the following clock. In the event of a secondary cache miss, the MC88410 inserts wait states by negating $\overline{\text{P\_TA}}$ until the data is latched on the system bus. For timing diagrams of transactions that miss in the secondary cache, refer to **Section 5 System Bus Interface**. The MC88110 continues to drive the address and data on the processor bus until $\overline{\text{P\_TA}}$ is asserted (or the transaction is terminated). Figure 4-17 shows the transaction flow for a burst write transaction.

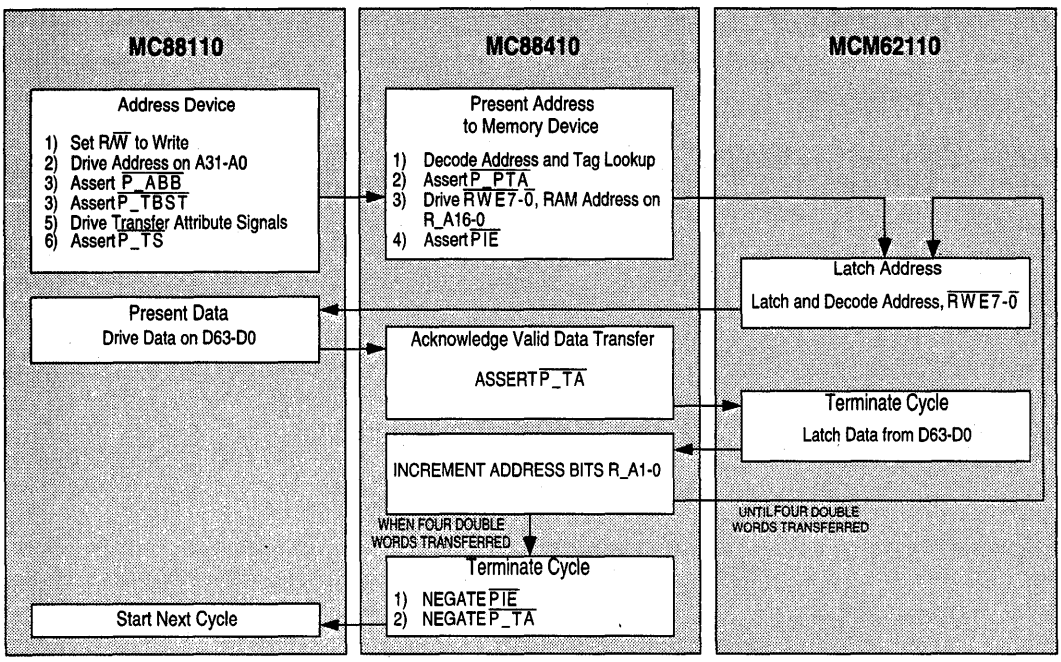


Figure 4-17. Burst Write Transaction Flow

The timing for a processor burst write transaction that hits in the secondary cache is shown in Figure 4-18. In clock 1, the processor begins the transaction by driving the address on the processor bus, asserting $\overline{\text{P\_ABB}}$, and asserting $\overline{\text{P\_TS}}$ to signal a new transaction. The MC88110 asserts P_R/$\overline{\text{W}}$ for the write transaction and asserts $\overline{\text{P\_TBST}}$ signal to indicate a burst transaction. The MC88110 also drives the data on the appropriate data signals (P DATA) in clock 1. In clock 2, the MC88410 drives the appropriate address and control information to the MCM62110 array. The $\overline{\text{PIE}}$ signal is asserted to enable data transfer from the processor to the MCM62110 array. To perform a cache tag lookup and to drive the RAM address and control signals to the MCM62110 array, the MC88410 inserts a wait state in clock 2 by continuing to negate $\overline{\text{P\_TA}}$.

Because the transaction hits in the secondary cache, the processor transfers the first aligned double word on the data bus in clock cycle 3 and the MC88410 acknowledges this by asserting $\overline{\text{P\_TA}}$. At the same time, the MC88410 increments the RAM address to the next double word. During each of the following three clock cycles, data for the specified address is transferred on the appropriate D63–D0 signals and the address is incremented to reflect the address of the appropriate double word. The address, data, and control signals are three-stated in clock 7, and $\overline{\text{P\_TA}}$ is negated.

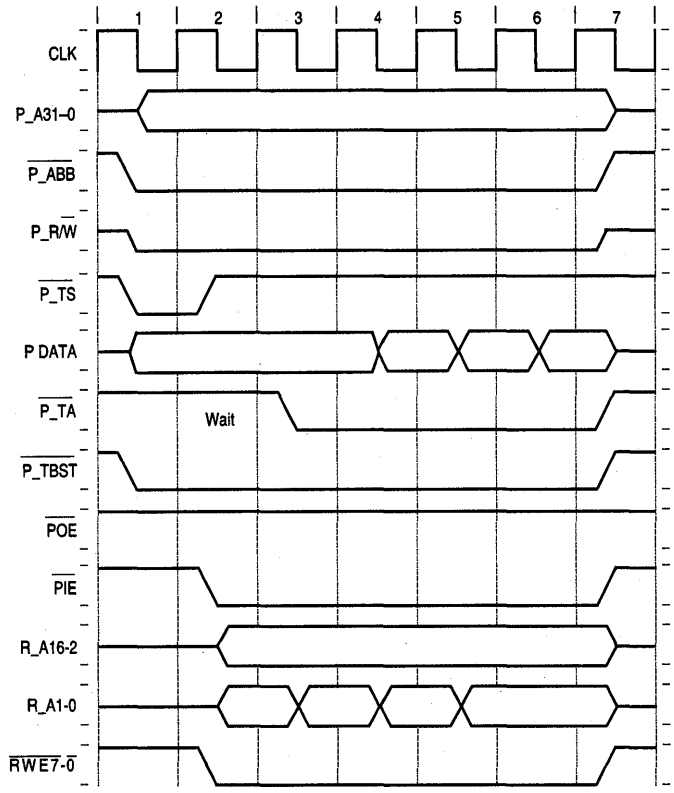


Figure 4-18. Burst Write Hit Timing

4.5 PROCESSOR TRANSACTION TERMINATION

The following paragraphs describe the methods used for terminating transactions on the processor bus. Transactions may be terminated normally, indicating that the transfer completed successfully, or terminated with an error or a retry indication.

The state of the $\overline{\text{P\_TA}}$, $\overline{\text{P\_TEA}}$, and $\overline{\text{P\_TRTRY}}$ signals determine the termination for each transaction on the processor bus. Table 4-6 depicts the encodings of $\overline{\text{P\_PTA}}$, $\overline{\text{P\_TA}}$, $\overline{\text{P\_TEA}}$, and $\overline{\text{P\_TRTRY}}$ and the corresponding types of transaction termination. The $\overline{\text{P\_PTA}}$ signal indicates that $\overline{\text{P\_TA}}$ may be asserted in the following clock to terminate the transaction and is used for decoupled access of the primary data cache. If the MC88410 $\overline{\text{P\_TEA}}$ signal is connected to the $\overline{\text{TEA}}$ signal of the MC88110, a transfer error termination is only signaled by devices on the system bus and is propagated on the processor bus to the processor. For information about transfer error termination, refer to **Section 5 System Bus Interface.**

Table 4-6. Transaction Termination Encodings

P_PTA	P_TA	P_TEA	P_TRTRY	Termination
A	A	N	N	Normal
x	x	A	x	Error
x	x	N	A	Transfer retry

A = Asserted
N = Negated
x = Don't care

4.5.1 Normal Transaction Termination with P_TA

The assertion of P_TA by the MC88410, while P_TRTRY and P_TEA are negated, indicates a normal termination to the processor. For a read transaction, the data is valid on the data bus and may be latched by the processor. For a write transaction, the data has been written to the memory system.

For single-beat transactions, the MC88110 completes the transaction after P_TA is asserted. To end the transaction, the MC88110 releases mastership of the processor bus by negating P_ABB unless it is parked and a new transaction is ready to begin. For burst transactions, each beat of the burst must be terminated by P_TA before the transaction is completed. Figure 4-19 shows a single-beat transaction that is completed with a normal transaction termination.

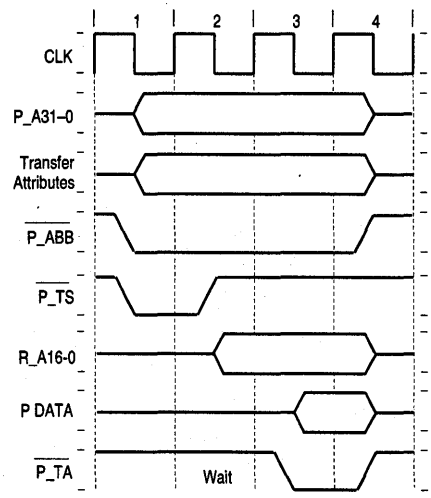


Figure 4-19. Normal Termination with P_TA

In Figure 4-19, the MC88110 starts a new transaction by asserting P_TS and P_ABB in clock 1. During clock 1, the MC88410 decodes the address and transfer attribute signals and performs a cache tag lookup to determine if there is a secondary cache hit. Assuming a

cache hit, the MC88410 drives the RAM address and control information in clock 2. To perform a cache tag lookup and to drive the RAM address and control signals to the MCM62110 array, the MC88410 inserts a wait state in clock 2 by continuing to negate $\overline{\text{P\_TA}}$. On the rising edge of clock 4, the MC88110 detects that $\overline{\text{P\_TA}}$ is asserted while the $\overline{\text{P\_TEA}}$ and $\overline{\text{P\_TRTRY}}$ signals are both negated, so it completes the transaction. For a burst transaction, each of the four double words must be terminated by asserting $\overline{\text{P\_TA}}$.

4.5.2 Termination for Decoupled Cache Accesses

The MC88110 can process instructions that access the primary data cache while executing external bus transactions using a feature called cache decoupling.

When the processor is operating with decoupled cache and bus accesses, the MC88110 $\overline{\text{PTA}}$ signal must be used to explicitly indicate when on-chip data cache accesses must be suspended in order to grant the bus access to the data cache. The MC88410 uses the $\overline{\text{P\_PTA}}$ signal to inform MC88110 that the initial assertion of $\overline{\text{TA}}$ may follow on the next rising edge. If decoupled cache accesses are not desired, the $\overline{\text{PTA}}$ signal can be connected to ground and $\overline{\text{P\_PTA}}$ can be left unconnected.

The window of time between the assertion of $\overline{\text{P\_TS}}$ by the processor and $\overline{\text{P\_PTA}}$ by the MC88410, allows load and store hits to the data cache to occur without interrupting bus activity. Once $\overline{\text{P\_PTA}}$ is asserted, $\overline{\text{P\_TA}}$ may follow in the next clock, so on-chip data accesses are prevented from accessing the primary data cache. The MC88110 begins sampling $\overline{\text{P\_PTA}}$ simultaneously with the assertion of $\overline{\text{P\_TS}}$. The MC88410 always asserts $\overline{\text{P\_PTA}}$ one clock cycle after the assertion of $\overline{\text{P\_TS}}$ regardless of secondary cache hit status. Once $\overline{\text{P\_PTA}}$ is recognized as asserted by the processor, it is ignored for the remainder of the transaction. For more information about the use of decoupled cache/bus accesses, see **Section 6 Instruction and Data Caches** in the *MC88110 Second Generation RISC Microprocessor User's Manual.*

Figure 4-20 shows a timing diagram for a single-beat transaction that uses $\overline{\text{P\_PTA}}$. The transaction begins in clock 1 with $\overline{\text{P\_PTA}}$ negated. The MC88110 drives the address and transfer attribute signals on the processor bus. At the same time, $\overline{\text{P\_TS}}$ is asserted for one clock cycle. During clock 1, the MC88410 performs a cache lookup to determine whether there is a secondary cache hit or a cache miss. In clock 2, the MC88410 drives the RAM address and control signals to the MCM62110 array and asserts $\overline{\text{P\_PTA}}$. During clock 3, the data is transferred and $\overline{\text{P\_TA}}$ is asserted to indicate the end of the transaction.

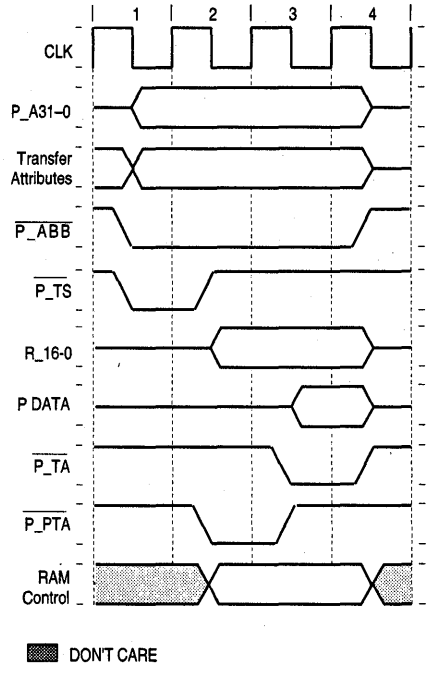


Figure 4-20 Decoupled Cache Access Timing

4.5.3 Transfer Retry Termination

Transfer retry terminations are usually initiated on the system interface of the MC88410. The assertion of $\overline{\text{P\_TRTRY}}$ indicates that the processor should terminate its transaction and allow the MC88410 to gain bus mastership. For example, Figure 4-21 shows a case where the MC88110 initiates a transaction at the same time that the MC88410 initiates a primary cache invalidate transaction. The MC88410 negates $\overline{\text{P\_BG}}$ in clock 1 and asserts $\overline{\text{P\_TRTRY}}$ during clock 2 in order to retry the processor and grant itself the bus (using on-chip arbitration). The MC88410 begins its primary cache invalidate transaction in clock 4 and completes it in clock 7. The processor can reinitiate its transaction in clock 8. Note that if the processor bus is not occupied when the MC88410 initiates a primary cache invalidate transaction, it will not assert $\overline{\text{P\_TRTRY}}$.

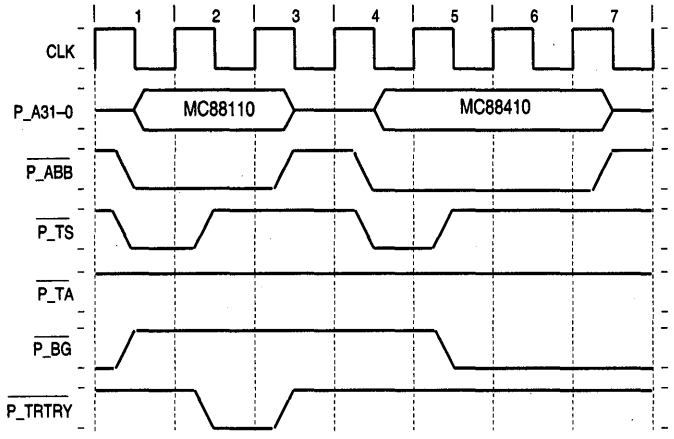


Figure 4-21. Retry of a Processor Transaction

A processor read transaction that misses in the secondary cache can cause the MC88410 to perform a secondary cache line fill transaction on the system bus. If a secondary cache line is filled due to a miss, the MC88410 may need to replace modified data. If the line to be replaced is included in the primary data cache, a primary cache invalidate transaction precedes all transactions to ensure that the processor has copied back its data before the MC88410 initiates a replacement copyback operation. If the MC88410 is involved in a processor bus transaction, the MC88410 asserts $\overline{\text{P\_TRTRY}}$ to become the processor bus master as shown in Figure 4-22.

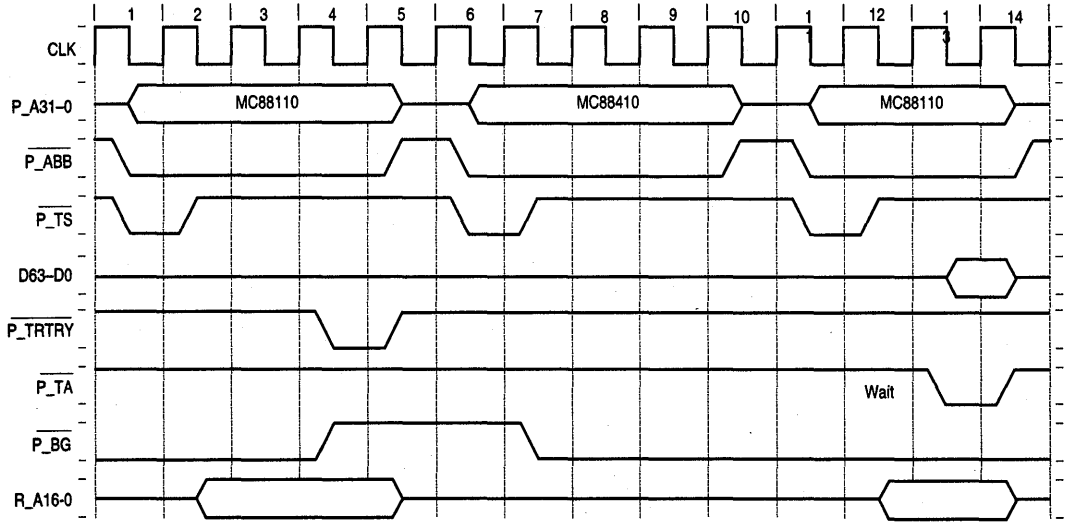


Figure 4-22. Retry Caused By Replacement Copyback

In clock 1, the processor begins a new transaction. In this example, the MC88410 determines that it needs to replace a secondary cache line that is included in the primary cache. The MC88110 performs a transaction in clocks 1 through 4. In clock 4, the MC88410 negates the $\overline{\text{P\_BG}}$ signal to the MC88110 (assuming on-chip arbitration) and asserts $\overline{\text{P\_TRTRY}}$. The processor immediately relinquishes the bus by negating $\overline{\text{P\_ABB}}$ and terminating the transaction.

In clock 6, the MC88410 assumes mastership of the bus and begins its primary cache invalidate. The MC88410 drives the address of the line to be replaced on the processor bus and asserts $\overline{\text{P\_TS}}$ for one clock cycle. If the broadcast address hits on a modified line, the MC88110 asserts $\overline{\text{P\_ARTRY}}$ to regain bus mastership and perform a snoop copyback operation. If the snoop hits on an unmodified primary cache line, the $\overline{\text{P\_ARTRY}}$ signal is not asserted. The MC88410 takes the absence of an address retry (after a two-clock snoop delay) to indicate that the primary cache line was invalidated by the snoop.

In Figure 4-22, the MC88410 has not detected $\overline{\text{P\_ARTRY}}$ asserted by clock 9 indicating that the primary cache line was invalidated. In clock 10, the MC88410 completes the primary cache invalidate transaction and relinquishes the processor bus.

In clock 11, the MC88110 regains mastership of the bus and begins a new transaction. Note that this example assumes a single-MC88410 configuration since $\overline{\text{P\_BG}}$ is asserted by the MC88410 in clock 7, one clock cycle after it asserts $\overline{\text{P\_TS}}$. Therefore, the bus arbitration handshake is not evident in clocks 10 and 11.

SECTION 5
SYSTEM BUS INTERFACE

This section provides a functional description of the MC88410 system bus, the signals that control the system bus, and the system bus cycles for data transfer operations. It also includes the descriptions of system bus timing, system bus arbitration, transaction types, split-bus transactions, double-word ordering, data streaming, termination, snoop timing, and collisions.

NOTE

The terms assert and negate are used extensively in this manual to avoid confusion between active-high and active-low signals. **Assert** or **assertion** indicates that a signal is active or true, regardless of whether the signal is active high or active low. **Negate** or **negation** indicates that the signal is inactive or false.

The MC88410/MCM62110 secondary cache contains both data and instruction data types. The term **data** is used in this manual to refer to both data and instructions unless specifically noted otherwise.

The timing for the external signals shown in this section is only accurate to within a half-clock cycle and is included for reference only. The input and output signals of the MC88410 are synchronous in that all setup and hold times are specified in reference to the clock signal. The MC88410 outputs are driven from a clock edge and a maximum delay is specified. In addition, minimum hold times are specified in relation to the clock. The minimum setup and hold times must be met to guarantee proper device operation. The timing for some signals on the MC88410 system bus interface has been shifted compared to the processor bus interface such that they are sampled on the rising edge of the clock instead of the falling edge. Therefore a system board design that is compatible with the MC88410 and the MC88110 must maintain valid data across both edges. For detailed timing information, refer to the *MC88410 Electrical Specifications*.

5.1 SYSTEM BUS INTERFACE OVERVIEW

The system bus interface is similar in operation to the processor bus interface. However, extensions to the processor bus protocol include the choice of full-speed or half-speed system bus clock frequency, the choice of double-word ordering of burst transactions, and extensions to the bus snooping protocol.

The MC88410 system bus interface includes many features that maximize the rate of data transfers between the processor and other devices in the system. All data transfers are synchronous and occur in either single-beat transactions or burst transactions. Burst transactions are either critical-word-first or zero-word-first. The data streaming feature provides the data to the processor as it is received from the bus while simultaneously bypassing or writing to the secondary cache.

Although one or more of the devices on the MC88410 system bus can have the capability of driving the system address and data buses, there can be only one device controlling each bus at any one time. This device is the address or data bus master. Bus arbitration is the protocol by which a device becomes a bus master. The MC88410 defines an arbitration protocol in which an external arbiter controls system bus arbitration and the MC88410 requests mastership of the system bus from the arbiter in order to perform an external access.

The MC88410 system bus has separate address and data buses whose mastership can be split from each other to enable split-bus transactions. Although the MC88410 does not include the data bus, it does control the system data bus arbitration. Through its control of the MCM62110 fast static RAMs the MC88410 also controls data output on the system data bus. Split-bus transactions allow different devices to control the address bus and data bus at the same time. This potentially increases system performance by allowing multiple bus transactions to be in progress simultaneously by multiple bus masters. Bus pipelining occurs when the address phase of a transaction can overlap the data phase of other transactions. The complexity of the pipeline levels is dependent on external circuitry.

The MC88410 must arbitrate for mastership of both the address and data bus separately. If the MC88410 is the only possible bus master on both buses, both buses can be continuously granted to the MC88410 by external logic and no arbitration is required. To avoid the overhead of arbitration, it may be desirable to "park" the MC88410 on the system address bus. The MC88410 is parked when bus grant is asserted and the MC88410 is not performing a bus transaction. For systems with multiple system bus masters but no split-bus transactions, the data bus can be continuously granted to the MC88410 by external logic, requiring only address bus arbitration.

System bus transactions may be terminated normally or terminated with an error or retry indication. The address retry terminates the transaction of the current address bus master. The transfer retry terminates the transaction of the current data bus master. Transfer errors and transfer retry indications are propagated to the processor interface when processor transactions are involved.

The MC88410 uses a bus snooping protocol to monitor bus transactions performed by other system bus masters and to intervene in the access, when required, in order to maintain cache coherency. The MC88410 services system bus snoop transactions from the secondary cache without interaction with the processor unless the snoop operation hits in the primary data cache.

5.2 SYSTEM BUS COMPATIBILITY

A memory system designed to be compatible with both the MC88110 system bus and the MC88410 system bus must take into account the signal timing differences. Table 5-1 summarizes the hardware timing differences between the MC88410 and MC88110 system bus for both the full-speed and half-speed mode. These differences are explained in detail in the remainder of this section.

Table 5-1. MC88110/MC88410 Timing Differences

Signal	MC88110	MC88410 Full-Speed	MC88410 Half-Speed
$\overline{TA}$ (input)	$\overline{TA}$ asserted with data	$\overline{S\_TA}$ asserted one CLK before data	$\overline{S\_TA}$ asserted with data
$\overline{TRTRY}$ (input)	Sampled one CLK after $\overline{DBG}$ is asserted	Sampled with $\overline{S\_DBG}$	Sampled one HCLK after $\overline{S\_DBG}$ is asserted
Data (input)	9 ns setup, -3 ns hold	2.5 ns setup, 2 ns hold (to MCM62110 array)	2.5 ns setup, 2 ns hold (to MCM62110 array)
Data (output)	4 ns minimum, 15 ns maximum propagation delay	Zero (0) ns minimum, 8 ns maximum propagation delay	Zero (0) ns minimum, 8 ns maximim propagation delay
$\overline{BR}$ (output)	$\overline{BR}$ asserted one CLK after $\overline{SSTAT1}$ on snoop hits	$\overline{S\_BR}$ asserted one CLK after $\overline{S\_SSTAT1}$ on snoop hits	$\overline{S\_BR}$ asserted with $\overline{S\_SSTAT1}$ on snoop hits

System signal differences between the MC88110 and MC88410 are shown in Table 5-2. The $\overline{TSHD}$ and $\overline{S\_SSTAT2}$ signals are MC88410 additions to the system bus protocol. The absence of the $\overline{WT}$ signal on the MC88410 system bus prevents signaling the write-through memory policy to a third level of cache memory.

Table 5-2. MC88110/MC88410 System Signal Differences

Signal	MC88110	MC88410
$\overline{WT}$	Indicates write-through memory update policy	Signal not available. Cannot signal write-through to next level of cache
PSTAT2–PSTAT0*	Provide visibility of the CPU status	Signals not available
$\overline{TSHD}$	Signal not available. Shared status of data is transmitted during address bus tenure using the $\overline{SHD}$ signal	Allows data to be marked as shared during data bus tenure
$\overline{S\_SSTAT2}$	Signal not available	Indicates this secondary cache will perform a snoop copyback transaction
F1–F0	Signals not available. Flushing/invalidation of cache performed through control registers	Asserted to flush or invalidate secondary cache
$\overline{FBSY}$	Signal not available	Indicates MC88410 is performing a flush operation
CS	Signal not available	Dynamically selects the MC88410
FD2	Signal not available	Used for tag monitoring
FD1/CWM*	Signal not available	Tag monitoring/word order for burst transactions
FD0/LINSIZ*	Signal not available	Tag monitoring/secondary cache line size
SD3/CSIZ1*	Signal not available	Tag monitoring/secondary cache size
SD2/CSIZ0*	Signal not available	Tag monitoring/secondary cache size
SD1/ARBEN*	Signal not available	Tag monitoring/select internal arbitration
SD0/CSP*	Signal not available	Tag monitoring/chip select polarity
HCLK	Signal not available	Half-speed clock input

Note: Signals marked with * (asterisk) are used to configure the MC88110 or MC88410 during reset. After reset these signals can be used during debug and for statistical analysis of the processor or secondary cache.

5.3 HALF-SPEED SYSTEM BUS TIMING

The MC88410 provides a half-speed mode for systems sensitive to component cost, board routing, and design issues related to the high clock rate of the MC88410 and MC88110 processor cluster. Half-speed mode allows the system interface to operate at half the clock speed of the processor bus while retaining the full-speed operation of the MC88410 processor interface.

The MC88410 requires both the CLK and the HCLK signals to operate in the half-speed mode. The MC88410 operates in half-speed mode when a half-speed clock is driven to HCLK. The MC88410 uses the HCLK signal as a qualifier for inputs and outputs on the system interface. The CLK and HCLK signals must be in phase with each other and the rising edge of HCLK must coincide with a rising edge of CLK within the electrical specification timing. Figure 5-1 shows a block diagram of a simple MC88110/MC88410 system implementing a half-speed system interface.

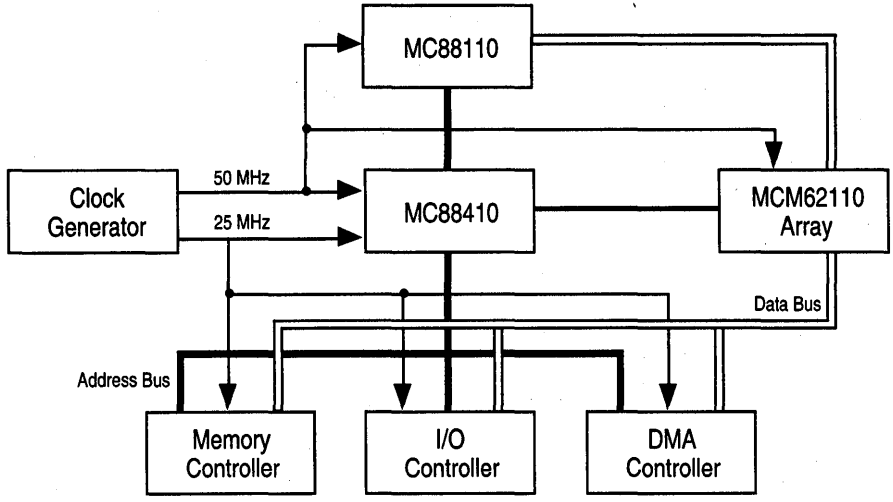


Figure 5-1. Half-Speed System Interface

All input and output timing specifications for the full-speed mode are applied to the half-speed mode as well. Setup and hold times are still referenced to CLK but the inputs are only sampled on the rising edge of HCLK. The full-speed mode differs from the half-speed mode in the relative timing of the $\overline{\text{S\_TA}}$ signal. In the half-speed mode the $\overline{\text{S\_TA}}$ signal must be asserted with the data (the same timing as the $\overline{\text{P\_TA}}$ signal). In the full-speed mode, the memory system must assert $\overline{\text{S\_TA}}$ one clock before the data is latched in order to allow the MC88410 time to increment the address to the secondary cache for burst transactions. The snooping protocol is the same for both modes except that $\overline{\text{S\_BR}}$ is asserted with $\overline{\text{S\_SSTATI}}$ for the half-speed mode instead of a clock later for the full-speed mode. The system bus timing diagrams reference the full-speed clock and half-speed clock when relevant. The term "system bus clock cycle" refers to CLK in the full-speed mode and HCLK in the half-speed mode.

5

5.4 SYSTEM BUS ARBITRATION

Arbitration for bus mastership in a multi-master system is performed by external arbitration logic and the system bus arbitration signals of the MC88410. Unlike the processor bus interface, which only provides for address bus arbitration, the system bus interface provides for independent address and data bus arbitration.

5.4.1 System Bus Arbitration Signals

Table 5-3 lists the system bus arbitration signals for the MC88410. Note that $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_DBB}}$ are I/O signals. These signals are outputs while the MC88410 has mastership of each of the buses and inputs at all other times.

Table 5-3. System Bus Arbitration Signals

Signal	Function	Type
$\overline{\text{S\_BR}}$	System bus request	Output
$\overline{\text{S\_BG}}$	System bus grant	Input
$\overline{\text{S\_ABB}}$	System address bus busy	I/O
$\overline{\text{S\_DBG}}$	System data bus grant	Input
$\overline{\text{S\_DBB}}$	System data bus busy	I/O

5.4.2 System Address Bus Arbitration

When the MC88410 needs to perform an external bus access and is not parked ($\overline{\text{S\_BG}}$ is negated), it asserts $\overline{\text{S\_BR}}$, and continues to assert $\overline{\text{S\_BR}}$ until it has been granted mastership of the address bus and the bus is available or until the bus is no longer needed. The external arbiter grants mastership of the bus to the potential master by asserting the $\overline{\text{S\_BG}}$ signal. Because the $\overline{\text{S\_ABB}}$ signal is asserted by the current master to indicate address bus mastership, the potential master determines that the bus is available when the $\overline{\text{S\_ABB}}$ signal is negated. A qualified system bus grant is defined as $\overline{\text{S\_BG}}$ asserted and $\overline{\text{S\_ABB}}$ negated (as an input). The potential master does not assume system address bus mastership until it receives a qualified system bus grant.

When a parked MC88410 ($\overline{\text{S\_BG}}$ continuously asserted) needs to perform a system bus access, it qualifies its bus grant with $\overline{\text{S\_ABB}}$. If $\overline{\text{S\_ABB}}$ is negated, then the MC88410 has a qualified bus grant and it can assume address bus mastership.

When the MC88410 receives a qualified bus grant, it assumes system address bus mastership by asserting the $\overline{\text{S\_ABB}}$ signal and negates the $\overline{\text{S\_BR}}$ output signal (unless the transaction is the first half of a locked operation). At the same time, the MC88410 drives the address for the requested access onto the system address bus and asserts the $\overline{\text{S\_TS}}$ signal to indicate the start of a new transaction.

In the clock cycle that $\overline{\text{S\_TS}}$ is asserted, the MC88410 begins sampling $\overline{\text{S\_DBG}}$ and $\overline{\text{S\_TA}}$. If the arbitration and external memory is fast enough to respond with $\overline{\text{S\_TA}}$ asserted, data could be sampled on the following clock.

When designing external bus arbitration logic, it is important to note that the MC88410 may assert $\overline{\text{S\_BR}}$ but not use the bus after it receives the qualified bus grant. One example of this is in a system that uses snooping. If bus master A asserts $\overline{\text{S\_BR}}$ in order to perform a replacement copyback transaction, it is possible for another device to invalidate that line before bus master A is granted the bus. Then, once bus master A is granted the bus, it no longer needs to perform the copyback transaction, and it does not assert $\overline{\text{S\_ABB}}$ for this case.

When operating the MC88410 in the half-speed mode, external arbitration logic should also take into account the fact that the MC88410 asserts $\overline{\text{S\_BR}}$ at the rising edge of the HCLK signal and samples $\overline{\text{S\_BG}}$ on the next rising edge of HCLK. The MC88410 qualifies its outputs with HCLK in the half-speed mode.

5.4.3 System Data Bus Arbitration

Although the MC88410 does not contain the data signals, the MC88410 is responsible for data bus arbitration and, through its control of the MCM62110s, the data bus. In addition to signaling the start of a new transaction, the assertion of the $\overline{\text{S\_TS}}$ output signal implies a system data bus request. The arbitration for the data bus is very similar to the arbitration for the address bus. The $\overline{\text{S\_TS}}$ signal serves the same function for the data bus as the $\overline{\text{S\_BR}}$ signal does for the address bus; however, $\overline{\text{S\_TS}}$ is asserted for only one clock cycle. As with the address bus, the MC88410 only assumes system data bus mastership when it has been granted the data bus and the data bus is available.

The external arbiter grants data bus mastership by asserting the $\overline{\text{S\_DBG}}$ signal. The potential data bus master determines that the bus is available when the $\overline{\text{S\_DBB}}$ signal is negated. A qualified system data bus grant is defined as $\overline{\text{S\_DBG}}$ asserted and $\overline{\text{S\_DBB}}$ negated (as an input).

When the MC88410 receives a qualified system data bus grant, the MC88410 asserts $\overline{\text{S\_DBB}}$ and data transfers may begin on the next rising clock edge. A design alternative for systems without a split address and data bus is to ground the $\overline{\text{S\_DBG}}$ signals for all bus masters, as the data bus can be controlled by $\overline{\text{S\_DBB}}$ alone.

Note that the data bus arbitration handshake must occur for all transactions. Therefore, even for the system invalidate transaction, in which no data is transferred, the $\overline{\text{S\_DBG}}$ signal must be asserted to the MC88410 for the transaction to terminate properly.

5.4.4 System Bus Arbitration Timing Examples

Figures 5-2 and 5-3 show the relative timing of the bus arbitration signals for some simple cases of full- and half-speed system bus arbitration. Note that there are separate signals shown for $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_DBB}}$ as inputs and as outputs (even though there is only one $\overline{\text{S\_ABB}}$ and one $\overline{\text{S\_DBB}}$ signal on the MC88410). This clarifies when these signals are monitored as inputs, when they are driven as outputs, and when they are ignored. In systems with multiple MC88410s, the multiple $\overline{\text{S\_ABB}}$ signals can be connected together, as can the multiple $\overline{\text{S\_DBB}}$ signals. The combined $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_DBB}}$ signals should be connected to pull-up resistors to keep the signal negated when no devices are driving the signals. For all timing diagrams that follow Figure 5-3, the combined $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_DBB}}$ signals are shown with the assumption that pull-up resistors are being used.

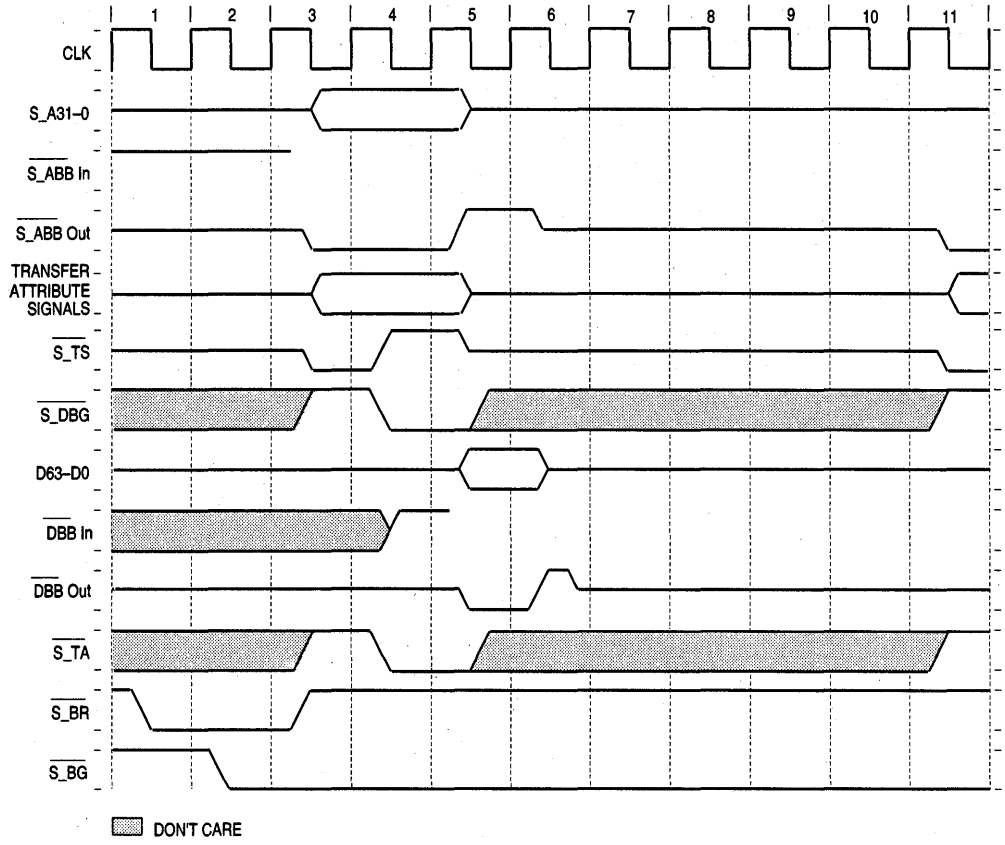


Figure 5-2. Full-Speed System Bus Arbitration Timing Example

Figure 5-2 shows an example of a single-beat (read or write) split bus transaction. In clock 1 of Figure 5-2, the MC88410 asserts $\overline{\text{S\_BR}}$ and monitors $\overline{\text{S\_BG}}$ and $\overline{\text{S\_ABB}}$. Note that all MC88410 output signals except those used for arbitration are three-stated during clocks 1 and 2 because the MC88410 is not the current bus master. However, it is likely that these signals are driven by other bus masters in the system during that time. Since it receives a qualified bus grant on the rising edge of clock 3, the MC88410 asserts $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_TS}}$, negates $\overline{\text{S\_BR}}$, and drives the appropriate values onto the address bus and transfer attribute signals. On the rising edge of clock 5, the MC88410 receives a qualified data bus grant and $\overline{\text{S\_TA}}$. During clock 5 the MC88410 asserts $\overline{\text{S\_DBB}}$ and the data is driven onto the data bus.

The MC88410 must arbitrate for the data bus for each transaction. This protocol enforces at least one clock cycle for data bus turnaround.

In Figure 5-2, many of the input signals are ignored a majority of the time. The $\overline{\text{S\_DBG}}$ signal is monitored only between when $\overline{\text{S\_TS}}$ is asserted and when the MC88410 assumes mastership of the data bus, which in this case is two clock cycles for each memory

transaction. The $\overline{\text{S\_TA}}$ signal is monitored when the MC88410 asserts $\overline{\text{S\_TS}}$ and during data bus mastership.

In the full-speed mode, the relative position of $\overline{\text{S\_TA}}$ is shifted so that $\overline{\text{S\_TA}}$ must be asserted one clock cycle before the system data is latched (a clock earlier than the processor bus signal $\overline{\text{P\_TA}}$), in this case, coincident with $\overline{\text{S\_DBG}}$. The function of the $\overline{\text{S\_TA}}$ signal is described in detail in **5.6 System Bus Transaction Termination**. In the half-speed mode, $\overline{\text{S\_TA}}$ must be asserted coincident with the data (the same timing as $\overline{\text{P\_TA}}$).

Note that in Figure 5-2, when the MC88410 is no longer using the address and data buses, it negates $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_DBB}}$ before three-stating the signals. As mentioned previously, these signals should be connected to pull-up resistors. The MC88410 negates the signal before three-stating it so that the signals meet the setup time for the next clock edge.

For the fastest case of back-to-back MC88110 transactions (and allowing for cache tag lookup), clock 11 is the earliest that the MC88410 requests the system bus again. In this example, the MC88410 remains parked on the system bus. For the fastest back-to-back MC88410 transaction (a replacement copyback followed by a line fill), the MC88410 asserts $\overline{\text{S\_TS}}$ (or $\overline{\text{S\_BR}}$ if it is not parked) in the clock following termination of the previous transaction.

Figure 5-3 shows the same example of system bus arbitration timing using half-speed mode. Note that the MC88410 asserts $\overline{\text{S\_BR}}$ on the rising edge of HCLK and samples $\overline{\text{S\_BG}}$ on the next rising edge of HCLK. In the half-speed mode $\overline{\text{S\_TA}}$ is sampled with the data. Half-speed mode arbitration timing is otherwise identical to the full-speed mode timing in the previous example.

In HCLK clock cycle 1 of Figure 5-3, the MC88410 asserts $\overline{\text{S\_BR}}$ and monitors $\overline{\text{S\_BG}}$ and $\overline{\text{S\_ABB}}$. Note that all MC88410 output signals except those used for arbitration are three-stated during HCLK clock cycles 1 and 2 because the MC88410 is not the current bus master. However, it is likely that these signals are driven by other bus masters in the system during that time. Since it receives a qualified bus grant on the rising edge of HCLK clock 3, the MC88410 asserts $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_TS}}$, negates $\overline{\text{S\_BR}}$, and drives the appropriate values onto the address bus and transfer attribute signals. On the rising edge of clock 5, the MC88410 receives a qualified data bus grant. During clock 5 the MC88410 asserts $\overline{\text{S\_DBB}}$ and the data is driven onto the data bus. The memory system asserts $\overline{\text{S\_TA}}$ with the data and it is detected by the MC88410 on the rising edge of clock 6.

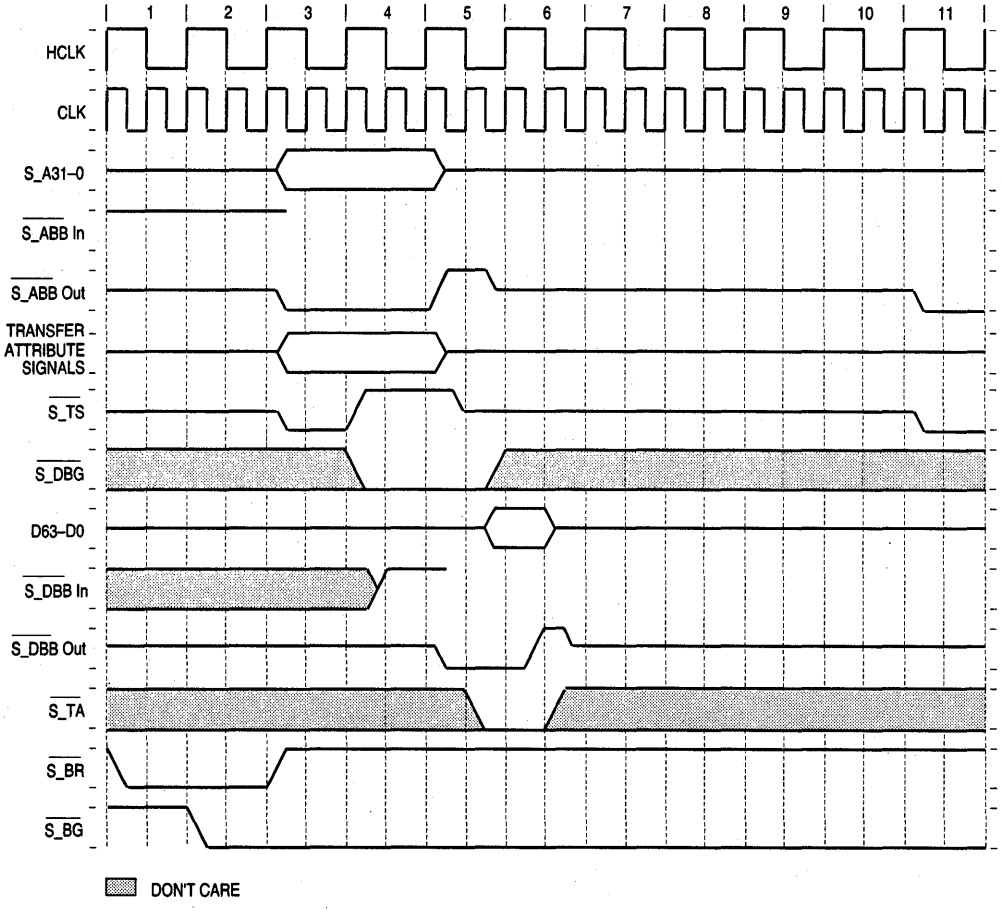


Figure 5-3. Half-Speed System Bus Arbitration Timing Example

5

Figure 5-4 shows an example of bus arbitration in full-speed mode in which the data bus is not immediately available for the MC88410/MCM62110. In clock 1 the MC88410 asserts $\overline{\text{S\_BR}}$ and monitors $\overline{\text{S\_BG}}$ and $\overline{\text{S\_ABB}}$. Since it receives a qualified bus grant on the rising edge of clock 3, the MC88410 asserts $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_TS}}$, negates $\overline{\text{S\_BR}}$, and drives the appropriate values onto the address bus and transfer attribute signals. In clock 4 $\overline{\text{S\_DBG}}$ is negated rather than asserted. Therefore, the MC88410 does not assume data bus mastership ($\overline{\text{S\_DBG}}$ asserted and $\overline{\text{S\_DBB}}$ negated as an input). On the rising edge of clock 6, the MC88410 receives a qualified data bus grant and $\overline{\text{S\_TA}}$. During clock 6 the MC88410 asserts $\overline{\text{S\_DBB}}$, the data is driven onto the data bus, and the transaction completes.

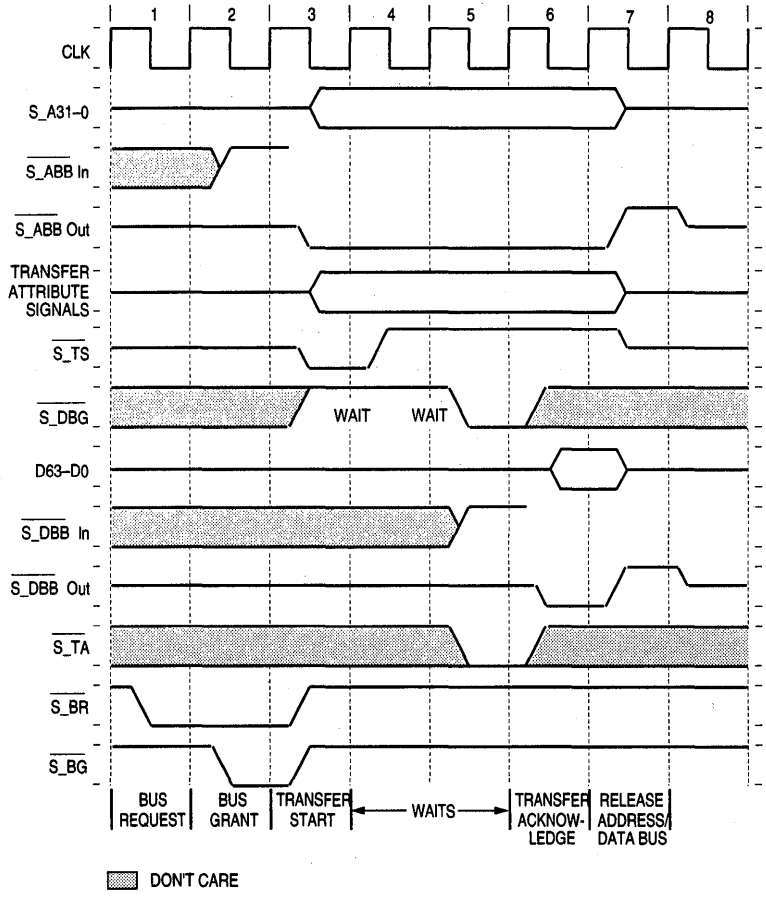


Figure 5-4. Full-Speed Data Bus Arbitration Timing Example

5

5.4.5 System Bus Parking

To avoid the overhead of arbitration, it may be desirable to "park" the device on the system address bus. The MC88410 is parked when $\overline{S\_BG}$ remains asserted regardless of whether the MC88410 is requesting bus mastership. If $\overline{S\_BG}$ is asserted when the system bus is requested internally, the MC88410 completes the arbitration sequence without any overhead and can begin the transaction without asserting $\overline{S\_BR}$. Thus, bus parking provides a performance advantage in that bus accesses begin without any delay for the arbitration protocol.

Figure 5-5 shows an example of the MC88410/MC88110 arbitration protocol using bus parking. Initially, an alternate master is master of the system bus and performs a data transaction. At the end of this transaction, the arbitration logic parks the MC88410 on the address bus by asserting the MC88410 $\overline{S\_BG}$ input. Clock cycles 4 and 5 show no device using the bus, but the MC88410 is parked on the address bus.

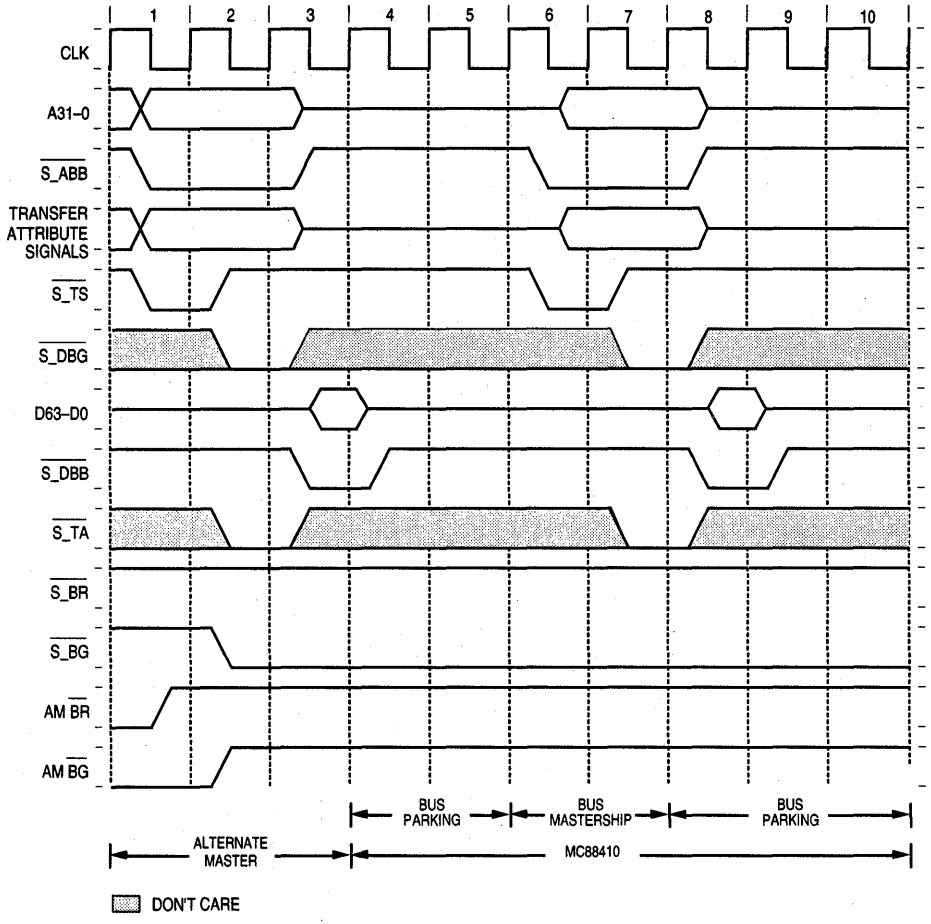


Figure 5-5. Bus Parking: Full-Speed Mode

5

In clock 6, the MC88410 initiates a transaction by driving the address and control information and asserting $\overline{\text{S\_TS}}$ and $\overline{\text{S\_ABB}}$. The $\overline{\text{S\_ABB}}$ signal is asserted to indicate that the address bus is in use (slow masters may assert $\overline{\text{S\_ABB}}$ without driving a valid address). Both $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_DBB}}$ always negate after the transaction is complete. The MC88410 always negates $\overline{\text{S\_ABB}}$ for one clock between back-to-back transactions.

At the end of the transaction shown in Figure 5-5, $\overline{\text{S\_BG}}$ for the MC88410 remains asserted, so the MC88410 remains parked on the address bus.

The external arbiter should use caution when negating $\overline{\text{S\_BG}}$ to a parked MC88410, because the parked MC88410 could assert $\overline{\text{S\_ABB}}$ and start a transfer in the same clock cycle that $\overline{\text{S\_BG}}$ is negated. The examples in this section assume that the $\overline{\text{S\_ABB}}$ signal of multiple masters are connected together.

5.4.6 System Bus Pipelining Protocol

The MC88410 has the capability to split the address and data buses so that they operate independently from one another. For example, in a multiprocessor configuration, the system address bus master is the MC88410 driving the address and the data bus master is the MC88410 that drove the address of the current data transfer. The separate control for this arbitration is controlled by the $\overline{\text{S\_AACK}}$ signal. The assertion of $\overline{\text{S\_AACK}}$ by a memory system indicates that the current address has been latched and that the address bus master can relinquish mastership of the address bus. The minimum address cycle time for any given transaction is two clocks on the system address bus.

The address bus master begins sampling the $\overline{\text{S\_AACK}}$ input signal during the clock after $\overline{\text{S\_TS}}$ is asserted. When the master detects that $\overline{\text{S\_AACK}}$ is asserted, it releases the address bus by three-stating it and negating $\overline{\text{S\_ABB}}$ so that another master can acquire the bus. The $\overline{\text{S\_AACK}}$ signal is ignored during any clock that results in the termination of the transaction (for example, $\overline{\text{S\_TEA}}$, $\overline{\text{S\_ARTRY}}$, $\overline{\text{S\_TRTRY}}$, or the last $\overline{\text{S\_TA}}$).

5.4.6.1 Multi-Master Single-Level Bus Arbitration

Figure 5-6 shows the relative timing for a split-bus transaction. The MC88410 drives an address onto the address bus and then detects the assertion of $\overline{\text{S\_AACK}}$. It then releases the address bus by three-stating it and negating $\overline{\text{S\_ABB}}$, but continues to transfer data on the data bus. The data transfer proceeds and terminates as in other normal transactions.

As shown in Figure 5-6, MC88410-A begins a transaction and drives an address on the address bus. MC88410-A begins the data transfer on the data bus and detects $\overline{\text{S\_AACK}}$ asserted on the rising edge of clock 3. Therefore, MC88410-A releases the address bus, which allows MC88410-B to begin to drive a new address onto the address bus before MC88410-A has completed the data transfer. A responding device can latch the new address from MC88410-B and begin the data access before the transaction of MC88410-A has completed. The ability to decouple the address and data bus increases the efficiency of the system by allowing the new address to overlap a previous transaction.

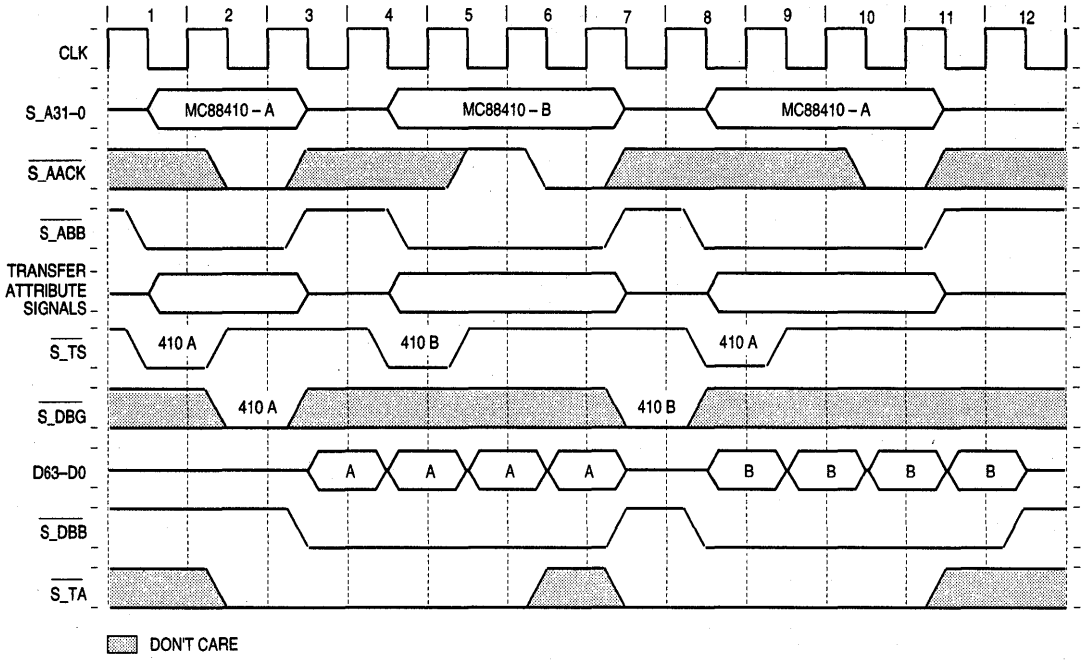


Figure 5-6. Full-Speed Split Bus (One-Level) Arbitration

Note that in this case, $\overline{\text{S\_AACK}}$ is not asserted to MC88410-B before MC88410-A has completed its data transfer, characterizing this transaction as a one-level split-bus transaction. One advantage in implementing a one-level split bus is that the $\overline{\text{S\_DBG}}$ signals to all bus masters can be connected to ground, which simplifies the data bus arbitration circuitry. After MC88410-A completes its data transfer in clock cycle 6, $\overline{\text{S\_DBB}}$ is negated and sampled by MC88410-B. The memory system asserts $\overline{\text{S\_AACK}}$ in clock 6 and the address bus is released by MC88410-B. MC88410-A then initiates another transaction on the address bus in clock 8.

5

5.4.6.2 Multi-Level System Bus Arbitration

A bus master can complete its address bus tenure before the data bus transaction of the previous bus master terminates, allowing multiple addresses to be outstanding on the bus. However, for each MC88410, only one outstanding transaction exists at any time. For example, it is possible to have four outstanding transactions at one time for a system with four MC88410s, which corresponds to a three-level split-bus system.

Multi-level split-bus systems require that the memory system generate the correct $\overline{\text{S\_DBG}}$ (and data) to the correct bus master. Figure 5-7 illustrates the relative timing for a multi-level split-bus transaction example. Note that in this case, MC88410-B gains address bus mastership and relinquishes it before the data is returned for the transaction of MC88410-A.

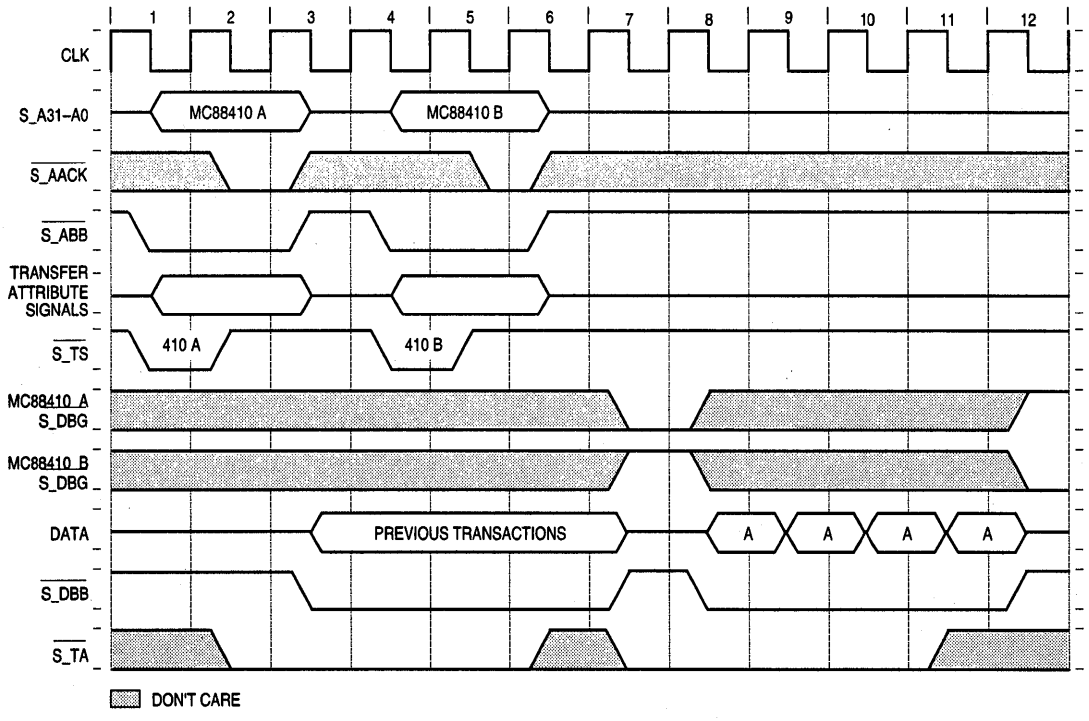


Figure 5-7. Full-Speed Multi-Level System Bus Arbitration

5.5 DATA TRANSFER MECHANISM

The following paragraphs describe the signals used in the transfer of data between the MC88410 and external devices on the system bus. The data transfer protocol and examples of the relative signal relationships for the different types of transactions are also described.

5

5.5.1 Data Transfer Mechanism Signal Overview

The signals that implement the data transfer mechanism for the MC88410 are classified as data transfer signals, transfer attribute signals, and transfer control signals. The transfer attribute signals are summarized in Table 5-4.

Table 5-4. System Bus Transfer Attribute Signal Summary

Function	Signal	Asserted	Negated
Read/Write	S_R/$\overline{W}$	Read	Write
Lock	$\overline{S\_LK}$	Transaction is one of two atomic transactions. Reflects the state of the $\overline{P\_LK}$ signal.	Transaction is not part of an atomic sequence. Reflects the state of the $\overline{P\_LK}$ signal.
Cache inhibit	$\overline{S\_CI}$	The transaction in progress is not cached in the primary cache and secondary cache.	The transaction in progress may be cached in the primary and secondary cache.
User page attributes	$\overline{S\_UPA1}$–$\overline{S\_UPA0}$	Reflect the state of the $\overline{P\_UPA1}$–$\overline{P\_UPA0}$ signals.	Reflect the state of the $\overline{P\_UPA1}$–$\overline{P\_UPA0}$ signals.
Transfer burst	$\overline{S\_TBST}$	Burst transaction	Single-beat transaction
Transfer size*	S_TSIZ1–S_TSIZ0	See Table 3-6.	See Table 3-6.
Transfer code	S_TC3–S_TC0	See Table 3-7.	See Table 3-7.
Invalidate	$\overline{S\_INV}$	This signal is broadcast to snooping MC88410s to invalidate the cache line	Invalidation not required
Memory cycle	$\overline{S\_MC}$	Data is transferred from MC88410 to an external device.	No data transfer to occur
Global	$\overline{S\_GBL}$	Data being transferred is global data	Data being transferred is local data

Note: The * (asterisk) indicates that signal should be ignored for burst cycles.

5.5.2 RAM Interface

Although the MC88410 does not contain the data bus, through its control of the MCM62110 secondary cache RAM array it does control data transfer. The RAM interface consists of the R_A16–R_A0, $\overline{RWE7}$–$\overline{RWE0}$, $\overline{PIE}$, $\overline{POE}$, $\overline{SIE}$, and $\overline{SOE}$ signals. The R_A16–R_A0, $\overline{RWE7}$–$\overline{RWE0}$, $\overline{SIE}$, and $\overline{SOE}$ signals are involved in system bus transactions. For more information regarding the RAM interface signals, refer to **3.3 RAM Interface Signals**.

The R_A16–R_A0 signals provide the secondary cache with the address of the transaction. For processor transactions, R_A16–R_A0 are asserted in the clock after the address (P_A31–P_A0) is driven on the processor bus. For system bus burst transactions, the MC88410 automatically increments the address to R_A16–R_A0 in the clock before data is valid. For this reason, $\overline{S\_TA}$ must be asserted one clock before the data is valid for all full-speed transactions.

The $\overline{RWE7}$–$\overline{RWE0}$ signals allow individual bytes of the 64-bit word to be written into the MCM62110 array. The $\overline{RWE7}$–$\overline{RWE0}$ signals are asserted or negated during system or processor data bus mastership depending upon the transaction. The $\overline{SOE}$ signal causes data from the MCM62110 array to be driven onto the system data bus and has identical timing to $\overline{RWE7}$–$\overline{RWE0}$. The $\overline{SIE}$ signal enables the MCM62110 array to latch data from the

system bus, is asserted in the same system bus clock cycle as $\overline{\text{S\_BR}}$, and remains asserted until the transaction terminates.

5.5.3 Data Transfer Transaction Summary

The system bus interface initiates transactions in response to system bus snooping and processor bus transactions. Data is transferred on the system bus in either single-beat transactions or burst transactions.

Transactions with $\overline{\text{S\_TBST}}$ negated are single-beat system bus transactions. Single-beat processor transactions ($\overline{\text{P\_TBST}}$ negated) can result in burst system bus transactions. Transactions with $\overline{\text{S\_TBST}}$ asserted are system bus burst transactions.

Table 5-5 summarizes the state of transfer attribute and control signals for system bus transactions. The actions of the MC88410 for any processor or system bus transaction depend on whether the access is cacheable and the state of the secondary cache line. The state of the secondary cache line is determined by coherency considerations. For information regarding processor transactions and the effect of cache coherency considerations on data transfer, refer to **Section 2 Secondary Cache Operation.**

All single-beat transactions except for system invalidate are the result of a single-beat processor transaction. System bus burst transactions result from both single-beat and burst processor transactions, as well as bus snooping. All write transactions must assert $\overline{\text{S\_INV}}$.

Table 5-5. System Bus Transaction Attribute and Control Signals

Transaction	S_R/$\overline{W}$	$\overline{S\_TBST}$	$\overline{S\_CI}$	$\overline{S\_INV}$	$\overline{S\_MC}$	$\overline{S\_LK}$	$\overline{S\_GBL}$	S_TSIZ1-S_TSIZ0	S_TC3-S_TC0
Single-Beat									
Cache- inhibited read	R	N	A	A	A	N	MMU	b,h,w,d	I/D,U/S
Locked read: Cache- inhibited load-store	R	N	A	N	A	A	MMU	b,w	D,U/S
Locked read: store-load	R	N	A	N	A	A	MMU	b,w	D,U/S
Cache- inhibited write	W	N	A	A	A	N	MMU	b,h,w,d	D,U/S
Write-through	W	N	N	A	A	N	MMU	b,h,w,d	D,U/S
Locked write: Cache- inhibited load-store	W	N	A	A	A	A	MMU	b,w	D,U/S
Locked write: Cached load-store in write-through mode	W	N	N	A	A	A	MMU	b,w	D,U/S
Locked write: store-load	W	N	A	A	A	A	MMU	b,w	D,U/S
System invalidate	W	N	N	A	N	N	A	b,h,w,d	D,U/S
Burst									
Secondary cache line fill	R	A	N	N	A	N	MMU	d	I/D,U/S
Secondary cache read-with-intent-to-modify	R	A	N	A	A	N	MMU	d	D,U/S
Replacement copyback	W	A	N	A	A	N	N	d	RCB
Snoop copyback	W	A	N	A	A	N	N	d	SCB
Flush copyback	W	A	N	A	A	N	N	d	D,S
System DMA invalidate	W	A	x	x	N	x	A	x	x

b= Byte
h= Half word
w = Word
d = Double word
R = Read
A = Asserted
N = Negated
x= Don't care

I = Instruction access
D = Data access
S = Supervisor access
U = User access
W = Write
SCB= Snoop copyback transaction
RCB= Replacement copyback transaction
MMU= Determined by MC88110 MMU

5.5.4 System Single-Beat Transactions

Single-beat transactions are read or write transactions that occur with disabled caches, cache-inhibited accesses, invalidation transactions, locked (**xmem**) transactions, and write transactions that occur in write-through mode. The only single-beat transaction initiated by the MC88410 is a system bus invalidate transaction. All other single-beat system bus transactions occur in response to processor bus single-beat data transactions. Transactions that are single-beat processor transactions can result in burst system bus transactions.

All single-beat transactions have similar timing characteristics; the differences between the transactions are determined by the transfer attribute signals of Table 5-5 that are asserted/negated. Note that all transaction types except for "invalidate" cause the $\overline{\text{S\_MC}}$ signal to be asserted. The $\overline{\text{S\_MC}}$ signal is asserted when data must be transferred between the MC88410/MCM62110 and an external device. If the processor transaction is cache-inhibited ($\overline{\text{P\_CI}}$ asserted), $\overline{\text{P\_TBST}}$ is ignored by the MC88410 and a single-beat transaction occurs on the system bus. The $\overline{\text{S\_INV}}$ signal is asserted to notify snooping MC88410s to invalidate their corresponding cache line if necessary and is asserted for all system bus write transactions. The state of the $\overline{\text{P\_CI}}$, $\overline{\text{S\_CI}}$, and $\overline{\text{P\_WT}}$ signals are determined by the MC88110 memory management units (MMU).

5.5.4.1 System Single-Beat Read Transactions

During single-beat read transactions on the system bus, the MC88410/MCM62110 reads a byte, half word, word, or double word from an external device. Single-beat read transactions on the system bus are caused by the following cache-inhibited processor transactions: single-beat read, touch load, and a locked read. An allocate load transaction that is cache-inhibited appears as a single-beat cache-inhibited read transaction.

To perform a single-beat read transaction, the processor asserts $\overline{\text{P\_TS}}$, negates $\overline{\text{P\_TBST}}$, drives the address onto the processor address bus and asserts or negates the appropriate transfer attribute signals as described in Figure 5-8. The MC88410 decodes the address and looks it up in the cache tags. Assuming a secondary cache miss, the MC88410 enables the MCM62110 array to latch data by asserting the appropriate cache control signals, arbitrates for the system address bus, drives the address onto the system address bus, and asserts or negates the transfer attribute signals appropriately. Finally, the MC88410 asserts $\overline{\text{S\_TS}}$ to indicate that the system address is valid.

At the beginning of each transaction, $\overline{\text{S\_TS}}$ is asserted for one clock. The external arbiter should interpret the assertion of $\overline{\text{S\_TS}}$ as a data bus request. In the full-speed mode the memory system must assert $\overline{\text{S\_TA}}$ one clock cycle before the data. If the memory system cannot supply the data within the appropriate setup and hold times, it should insert wait states by negating $\overline{\text{S\_TA}}$ until the data is available.

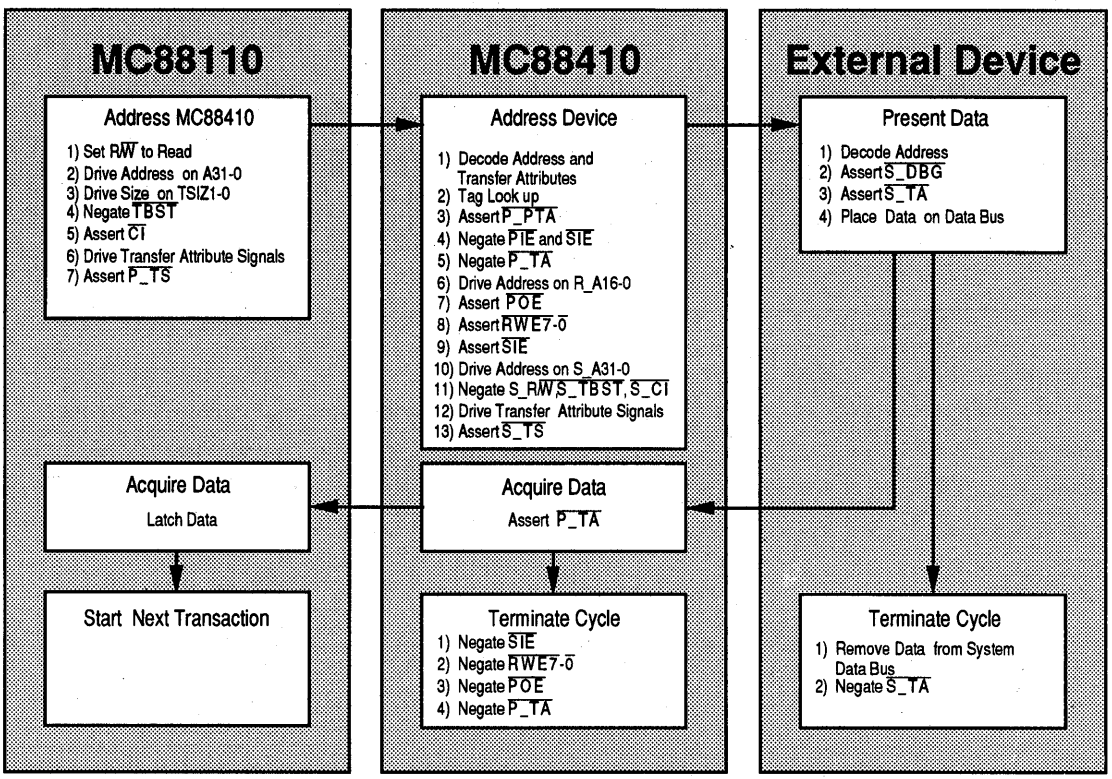


Figure 5-8. Full-Speed Single Beat Read Transaction Flow

5

Once the MC88410 becomes the data bus master, the memory system should supply the requested data on the appropriate D63–D0 signals within the required setup and hold times with respect to the rising edge of the clock. Once S_TA is asserted, the MC88410 completes the transaction and asserts P_TA, terminating the processor transaction.

5.5.4.2 System Single-Beat Read Transaction Timing

Figure 5-9 shows the relative timing of the data transfer signals during a single-beat read transaction. In this example the transaction is a cache-inhibited, single-beat read transaction from the processor (P_CI and S_CI asserted, RWE7–RWE0 negated). Arbitration signals, assuming a synchronous external arbiter, and the RAM interface signals are shown for reference.

As shown in Figure 5-9, the MC88110 drives the address signals with the physical address of the access off the rising edge of clock 1 and at the same time asserts the appropriate attribute and control signals for the type of single-beat transaction being performed (see Table 5-5). In this example P_CI is asserted for a cache-inhibited read. Since the transaction is cache-inhibited, RWE7–RWE0 remain negated throughout the transaction. The MC88410 samples the processor address on the next rising clock edge (clock 2).

The MC88110 also asserts the P_TS signal off the rising edge of clock 1 for one clock. The MC88410 interprets P_TS as indicating that a transfer has begun and that the driven address is now valid. Since the processor is always granted the data bus, P_TS is not interpreted as a data bus request. Both processor address and data bus arbitration is controlled by P_ABB alone.

The MC88410 asserts P_PTA in the clock following the assertion of P_TS. At the same time it also asserts POE and drives the address to the secondary cache RAM (R_A16–R_A0).

In clock 3 the MC88410 has completed its tag lookup of the address. In the next clock the MC88410 asserts S_BR to request the system bus. Note that it takes two clock cycles from the recognition of P_TS for the MC88410 to look up the cache tags and execute the system bus request. At the same time the MC88410 asserts SIE to allow data to be streamed through the MCM62110 array to the processor.

When the MC88410 asserts S_BR in clock 4 it monitors S_BG and S_ABB. In this example the external arbiter asserts S_BG in clock 5. Since the MC88410 recognizes a qualified bus grant on the rising edge of clock 6, the MC88410 asserts S_ABB and S_TS, negates S_BR, and drives the appropriate values onto the system address bus and transfer attribute signals. The assertion of S_TS also acts as a data bus request.

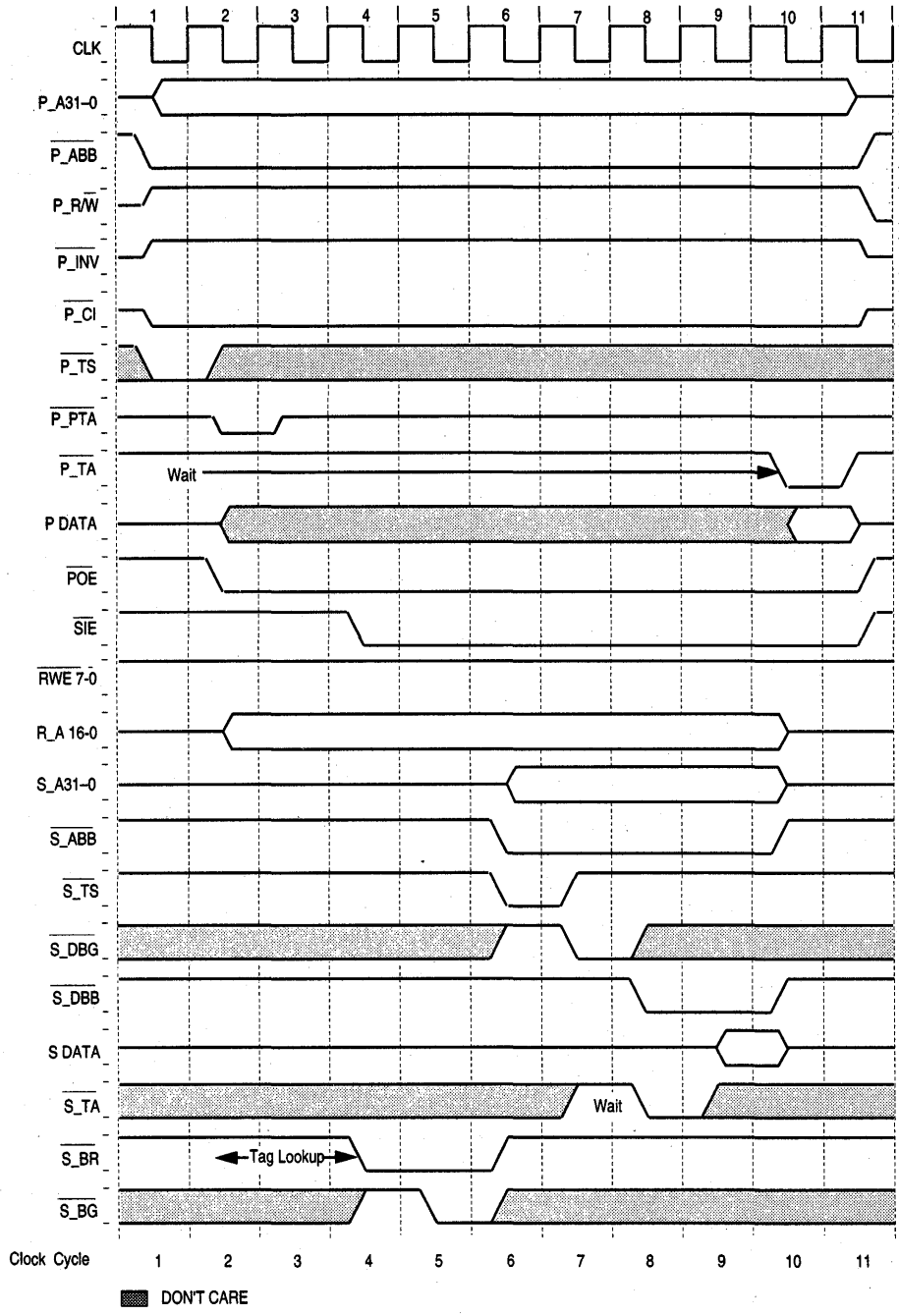


Figure 5-9. Full-Speed Single-Beat Cache-Inhibited Read

In clock 7 the external arbiter asserts $\overline{\text{S\_DBG}}$, which is qualified by $\overline{\text{S\_DBB}}$ negated, allowing the MC88410 to assume data bus mastership. To indicate the status of the transaction to the MC88410, the memory system then either asserts or negates the $\overline{\text{S\_TA}}$ signal. When the data is guaranteed to meet the appropriate setup and hold times with respect to the rising edge of the clock, the memory system should assert $\overline{\text{S\_TA}}$ to terminate the transaction. In the full-speed mode, $\overline{\text{S\_TA}}$ must be asserted one clock before the data (clock 8 in Figure 5-9). In the half-speed mode, $\overline{\text{S\_TA}}$ is asserted concurrent with the data. If the data cannot be supplied in time during the clock cycle after the address is sampled, $\overline{\text{S\_TA}}$ must be explicitly negated until the appropriate setup and hold times are met. In this example the memory system cannot provide the data until one clock after $\overline{\text{S\_TA}}$ is asserted in clock 8, incurring a one clock wait. The MC88410 continuously drives the address on the system address bus until $\overline{\text{S\_TA}}$ is asserted. The memory system can insert as many wait cycles as necessary until the appropriate data setup and hold times are met.

In clock 8 the MC88410 asserts $\overline{\text{S\_DBB}}$ to indicate data bus mastership. In clock 9 the data is valid and is latched from the system data bus. In the next clock the MC88410 releases the data bus by negating $\overline{\text{S\_DBB}}$, stops driving the address onto the system bus, and negates the RAM interface signals. With the assertion of $\overline{\text{P\_TA}}$ in clock 10, the processor completes its transaction.

If the transaction terminates with an error or retry, the memory system should assert the $\overline{\text{S\_TEA}}$ signal for a bus error, the $\overline{\text{S\_TRTRY}}$ signal for a transfer retry, or the $\overline{\text{S\_ARTRY}}$ signal for an address retry. For more information about $\overline{\text{S\_TA}}$ and other termination signals, refer to **5.6 System Bus Transaction Termination.**

5.5.4.3 System Single-Beat Write Transactions

During single-beat write transactions, the MC88110 transfers a byte, half word, word, or double word to an external device. Single-beat write transactions on the system bus result from MC88410 system invalidate transactions, write-through transactions, or cache-inhibited write transactions.

Figure 5-10 describes the flow of a processor single-beat write transaction that propagates to the system bus. The processor initiates a single-beat write transaction after becoming the processor address bus master. The processor then drives the address onto the processor address bus and asserts or negates the appropriate attribute and control signals (see Table 5-5). All write transactions from the MC88110 and MC88410 cause the invalidate ($\overline{\text{P\_INV}}$ and $\overline{\text{S\_INV}}$, respectively) signals to be asserted so that snooping devices can invalidate their cached versions of the data. The MC88410 enables the secondary cache for input from the processor (if it is not cache-inhibited) and arbitrates for the system bus. The system bus transaction concludes with the assertion of $\overline{\text{S\_TA}}$ by the memory system.

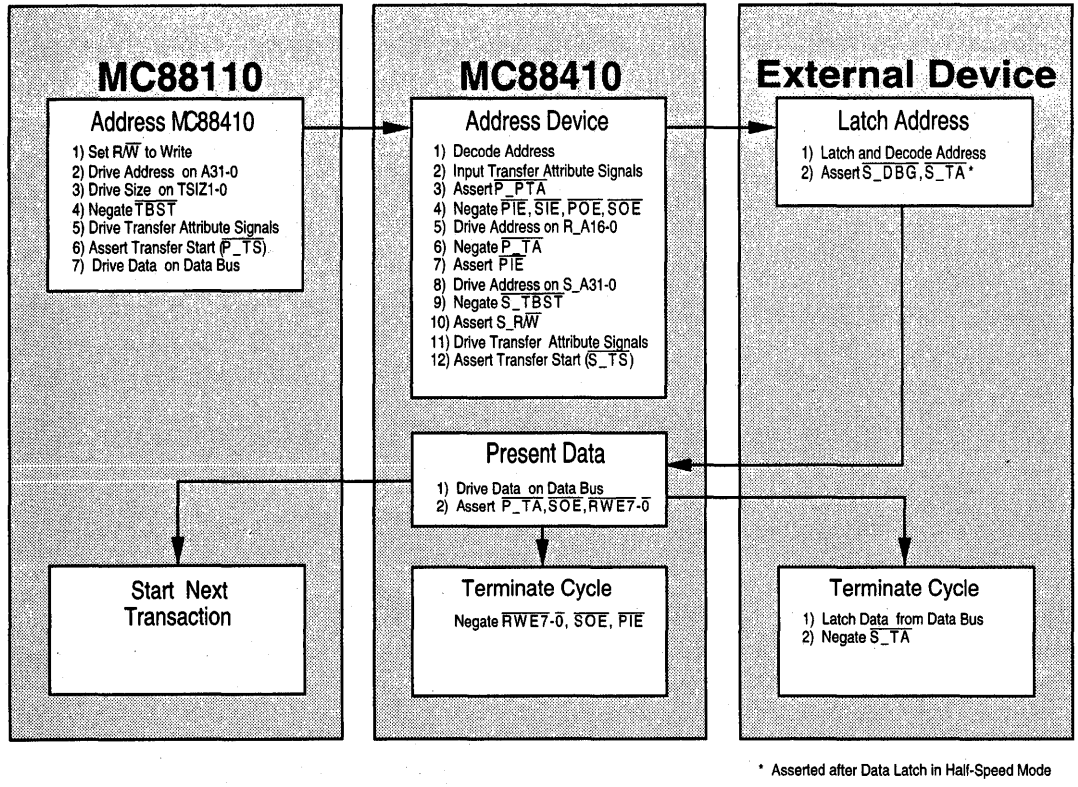


Figure 5-10. Full-Speed Single-Beat Write Transaction Flow

5

5.5.4.4 System Single-Beat Write Transaction Timing

Figure 5-11 shows the relative timing of the data transfer signals during a single-beat write transaction. In this example the transaction is a single-beat write-through transaction from the processor that hits in the secondary cache. If this had been a write-through write that missed in the secondary cache, data would be written to the system bus interface but not into the secondary cache ($\overline{RWE7}$–$\overline{RWE0}$ negated). Arbitration signals assuming a synchronous external arbiter and the RAM interface signals are shown for reference.

As shown in Figure 5-11, the MC88110 drives the address signals with the address of the access during the rising edge of clock 1 and at the same time asserts the appropriate attribute and control signals for the type of single-beat transaction being performed (see Table 5-5). In this example the $\overline{P\_INV}$ and $\overline{P\_WT}$ signals are asserted for a write-through transaction. The MC88410 samples the address on the next rising clock edge (clock 2).

The MC88110 also asserts the $\overline{\text{P\_TS}}$ signal off the rising edge of clock 1 for one clock cycle. The MC88410 interprets $\overline{\text{P\_TS}}$ as indicating that a transfer has begun and that the driven address is now valid. Since the processor is always granted the data bus, $\overline{\text{P\_TS}}$ is not interpreted as a data bus request. Both processor address and data bus hand-off is controlled by $\overline{\text{P\_ABB}}$ alone.

The MC88410 generates $\overline{\text{P\_PTA}}$ in the clock following the assertion of $\overline{\text{P\_TS}}$. At the same time it also asserts $\overline{\text{PIE}}$ and drives the address to the secondary cache RAM (R_A16–R_A0). While $\overline{\text{PIE}}$ is asserted, the MC88110 continues to write data into the secondary cache and drive the address until $\overline{\text{P\_TA}}$ is asserted by the MC88410.

In clock 3 the MC88410 has completed its tag lookup of the address. In the next clock the MC88410 asserts $\overline{\text{S\_BR}}$ to request the system bus. Note that it takes two clock cycles from the recognition of $\overline{\text{P\_TS}}$ for the MC88410 to look up the cache tags and perform the system bus request.

When the MC88410 asserts $\overline{\text{S\_BR}}$ in clock 4, it monitors $\overline{\text{S\_BG}}$ and $\overline{\text{S\_ABB}}$. In this example the external arbiter asserts $\overline{\text{S\_BG}}$ in clock 5. Since the MC88410 recognizes a qualified bus grant on the rising edge of clock 6, the MC88410 asserts $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_TS}}$, negates $\overline{\text{S\_BR}}$, and drives the appropriate values onto the system address bus and transfer attribute signals. The assertion of $\overline{\text{S\_TS}}$ also acts as a system data bus request.

In clock 7 the external arbiter asserts $\overline{\text{S\_DBG}}$, which is qualified by $\overline{\text{S\_DBB}}$ negated. The MC88410 continues to drive data onto the system data bus until $\overline{\text{S\_TA}}$ is asserted. One clock cycle before the memory system can latch the data (in the full speed mode), it should assert $\overline{\text{S\_TA}}$ to terminate the transaction. In the fastest case for the full-speed mode, $\overline{\text{S\_TA}}$ is asserted in the clock that the address is sampled and one clock before the data.

In this example, $\overline{\text{S\_TA}}$ is asserted in the clock following $\overline{\text{S\_TS}}$ (clock 7 in Figure 5-11). In the half-speed mode, $\overline{\text{S\_TA}}$ causes data to be latched in the same clock. If the data cannot be latched in time during the clock after the address is sampled, $\overline{\text{S\_TA}}$ must be explicitly negated until the appropriate setup and hold times are met. While $\overline{\text{S\_TA}}$ is negated, the MC88410 continuously drives the address and data on the system bus until $\overline{\text{S\_TA}}$ is asserted (or $\overline{\text{S\_AACK}}$ is asserted to terminate address bus mastership). The memory system can insert as many wait cycles as necessary until it can latch the data.

The MC88410 asserts $\overline{\text{RWE7}}$–$\overline{\text{RWE0}}$ in clock 7 to enable the writing of data into each byte of secondary cache. The $\overline{\text{SOE}}$ signal is asserted in clock 8, when it becomes the data bus master, to enable the data to be driven to the system data bus.

In clock 8 the MC88410 asserts $\overline{\text{S\_DBB}}$ and data is driven on the system data bus. In the next clock the MC88410 releases the data bus by negating $\overline{\text{S\_DBB}}$, stops driving the address onto the system bus, and negates the RAM interface signals. The MC88410 asserts $\overline{\text{P\_TA}}$ in clock 8, so the processor stops driving data and completes its transaction.

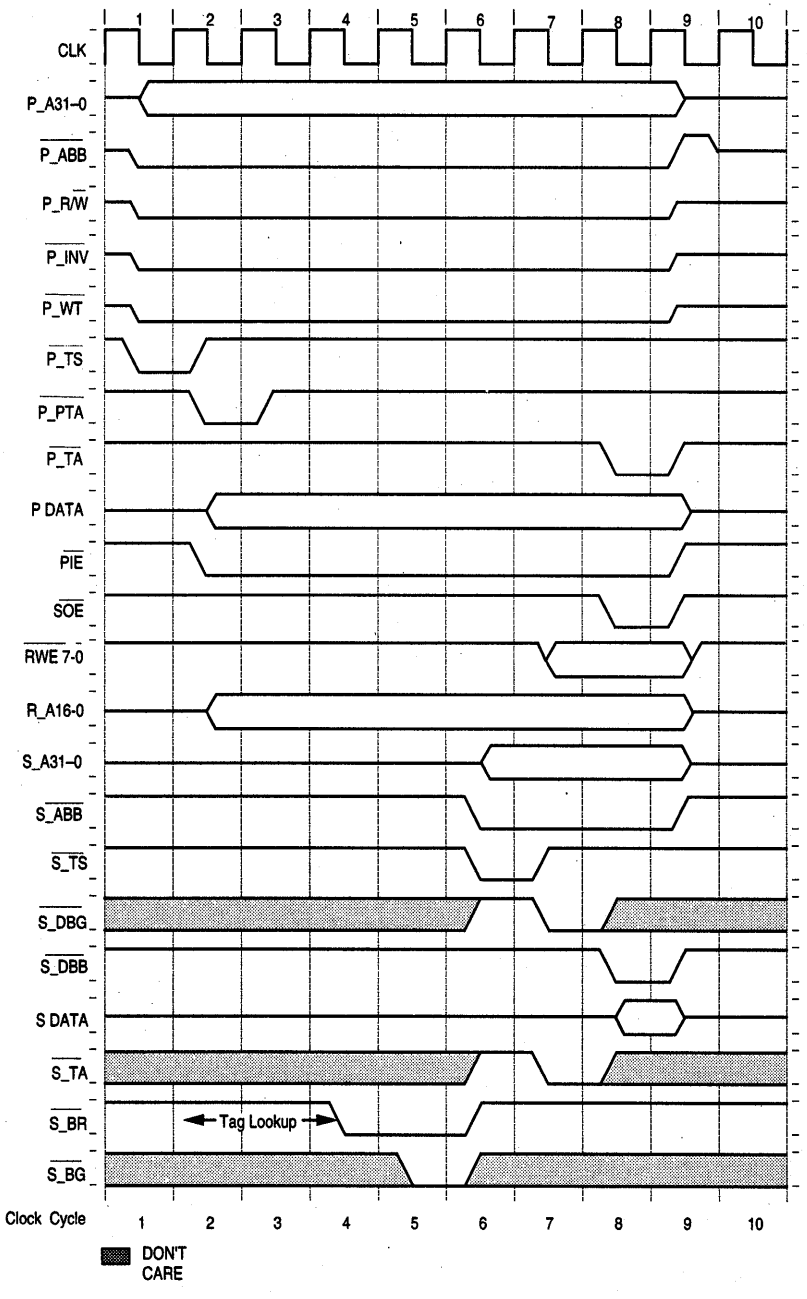


Figure 5-11. Full-Speed Single-Beat Write-Through Timing

5.5.4.5 System Invalidate Transaction

Invalidate transactions are single-beat transactions used by the MC88410 to maintain cache coherency among multiple caches. Invalidate transactions broadcast to snooping

bus masters that a cache line will be modified; thus, snooping bus masters should invalidate their cached versions of the line. See **Section 2 Secondary Cache Operation** for more information about snooping and cache coherency.

An invalidate transaction is an address-only transaction; although valid data is driven on the data bus, no data is transferred. Invalidate transactions use the protocol defined for single-beat write transactions. The only difference between an invalidate transaction and a normal single-beat write transaction is that for an invalidate transaction, $\overline{\text{S\_MC}}$ is negated since no data must be transferred. For both invalidate and normal single-beat write transactions, the S_R/$\overline{\text{W}}$ signal is asserted, signaling a write, and $\overline{\text{S\_INV}}$ is asserted to notify snooping processors to invalidate their cached versions of the line.

Even though no data is transferred during an invalidate transaction, the MC88410 must still request and be granted the data bus. Unless a transaction is terminated with an address retry indication (see **5.6.3 Address Retry Transaction Termination**), the transaction cannot be completed until the external arbiter asserts $\overline{\text{S\_DBG}}$ and the memory system asserts $\overline{\text{S\_TA}}$. The $\overline{\text{S\_TEA}}$ and $\overline{\text{S\_TRTRY}}$ signals are not recognized until the MC88410 has received a qualified data bus grant.

Figure 5-12 shows the timing diagram for a system bus invalidate transaction assuming that the MC88410 has been granted the address bus.

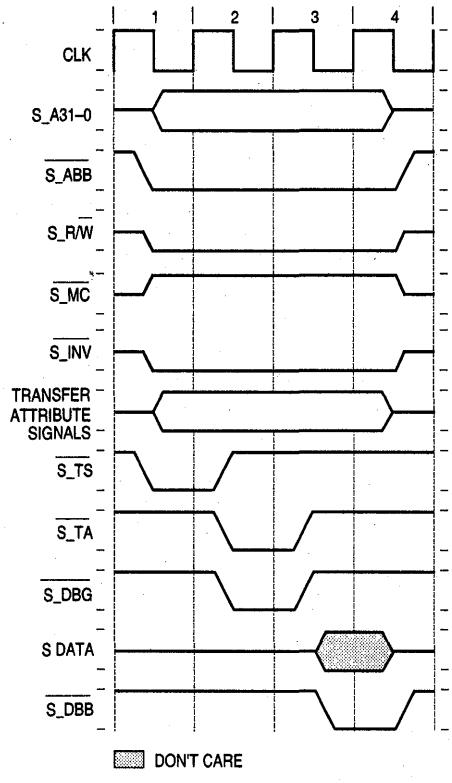


Figure 5-12. Full-Speed System Invalidate Transaction

5.5.4.6 Locked Transactions

The **xmem** instruction is a multiprocessor synchronization instruction that uses a single-beat read transaction and a single-beat write transaction to exchange the contents of a processor general register with the contents of a memory location. The **xmem** instruction is normally used to implement semaphores or resource locks in multiprocessor or multitasking systems.

The MC88110 **xmem** instruction is effectively a locked combination of a processor load and store instruction. The MC88110 implements the **xmem** instruction in one of two ways. In load-store ordering (the MC88110 default case), the **xmem** instruction causes a single-beat read followed by a single-beat write transaction. In store-load ordering the **xmem** instruction causes the MC88410 to perform a single-beat write followed by a single-beat read transaction.

The MC88410 response to locked transactions is dependent upon the order. For store-load ordering, the MC88410 treats the locked transaction as a pair of cache-inhibited accesses that results in a single-beat write and read on the system bus. For load-store ordering, the MC88410 treats the locked transaction as a pair of cacheable data requests depending on the $\overline{\text{P\_CI}}$ signal. Load-store locked transactions that are not cache-inhibited

and miss in the secondary cache causes a system bus read-with-intent-to-modify transaction. For the cache-inhibited load-store locked transaction, both the locked store and the locked load result in single-beat system bus transactions. The MC88410 system bus interface allows both transactions to be forced into a single bus tenure as follows.

During the execution of the locked transaction, the $\overline{\text{S\_LK}}$ signal is asserted for both the read and write portions of the transaction. The lock signal is asserted to indicate that the bus arbitration circuitry should not allow another bus master to alter the data being accessed by the **xmem** instruction between the read and the write. The external arbiter can ensure this by not granting mastership of the bus to other potential bus masters between the read and write portions of the locked transaction.

The $\overline{\text{S\_BR}}$ signal operates slightly differently for locked transactions than for all other transactions in that $\overline{\text{S\_BR}}$ remains asserted while $\overline{\text{S\_TS}}$ is asserted for the first transaction in the locked transaction. In all other cases, including the second transaction in locked transaction, the $\overline{\text{S\_BR}}$ signal is negated when $\overline{\text{S\_TS}}$ is asserted. The external arbiter can use this feature to easily lock the bus between the two transactions by not negating $\overline{\text{S\_BG}}$ (once it is asserted) until the MC88410 negates $\overline{\text{S\_BR}}$, thus parking the MC88410. Another advantage in keeping $\overline{\text{S\_BG}}$ asserted throughout the two transactions is that the transfer attribute signals remain valid.

If $\overline{\text{S\_BG}}$ is negated, or if $\overline{\text{S\_AACK}}$ is asserted during a locked tenure, the address bus tenure ends, and the locked pair of accesses continue execution as two independent accesses until completion. In this case the address bus mastership is released ($\overline{\text{S\_LK}}$ is negated between the two accesses) and the system bus mastership may be granted to another bus master between the two accesses. The MC88410 will negate $\overline{\text{S\_BR}}$ if $\overline{\text{S\_AACK}}$ is asserted. If address bus is retried ($\overline{\text{S\_ARTRY}}$ asserted) during the first transaction of the locked transaction, then the locked transation is reinitiated from the begining.

Between the load and store transactions of a cacheable locked transaction, the MC88410 maintains a lock collision buffer that retries any snooped address that attempts to access the same cache line address as the locked transaction. For more information about lock collisions, see **5.8.4 Lock Collision.**

5

5.5.4.7 Locked Transaction Timing

Figure 5-13 shows the timing for a full-speed, cache-inhibited, load-store **xmem** transaction where the arbiter parks the MC88410 so that the bus is locked between transactions. The system bus arbitration signals and RAM interface signals are shown for reference.

In clock 1, the processor initiates the single-beat read when it asserts $\overline{\text{P\_TS}}$ and drives the address, transfer attribute signals, and the $\overline{\text{P\_LK}}$ signal which identifies it as an **xmem** transaction. In clock 2, the MC88410 responds by asserting $\overline{\text{P\_PTA}}$ (not shown), driving the address to the MCM62110 array (R_A16–R_A0), and asserting the $\overline{\text{POE}}$ signal. During clock 3 the MC88410 compares the address to the internal cache tags.

Initially, the MC88410 is not parked on the system bus and in clock 4, the MC88410 requests the system bus and enables the RAM input from the system bus by asserting the

$\overline{\text{SIE}}$ signal. In clock 5, the MC88410 drives the address and transfer attribute signals onto the system bus and asserts $\overline{\text{S\_TS}}$ and $\overline{\text{S\_ABB}}$.

In this full-speed example, the external arbiter grants the data bus in clock 6 and the external memory system asserts $\overline{\text{S\_TA}}$. In clock 7, the MCM62110 array latches the data and the MC88410 completes the transaction on the system bus. Notice that the arbiter has parked the MC88410 in this example by continuing to assert $\overline{\text{S\_BG}}$. The MC88410 responds by continuing to assert it $\overline{\text{S\_BR}}$, $\overline{\text{S\_ABB}}$, $\overline{\text{S\_LK}}$, and other transfer attribute signals.

In clock 8, the MC88410 asserts $\overline{\text{P\_TA}}$ and the processor latches the data, completing its read transaction. The processor continues to assert its transfer attribute signals and $\overline{\text{P\_ABB}}$.

In clock 12, the processor again asserts $\overline{\text{P\_TS}}$ to initiate the single-beat write half of the **xmem**. The MC88410 responds by asserting the $\overline{\text{PIE}}$ signal in the next clock and again drives the address to the secondary cache. On the rising edge of clock 16, the MC88410 recognizes its bus grant and negates $\overline{\text{S\_BR}}$. In clock 16, the MC88410 asserts $\overline{\text{S\_TS}}$ and drives the address onto the system bus. The MC88410 receives a data bus grant in clock 17, asserts the $\overline{\text{SOE}}$ signal and recognizes $\overline{\text{S\_TA}}$. In clock 18, the MC88410 drives the data onto the system bus and asserts $\overline{\text{P\_TA}}$. Finally, in clock 19 the processor recognizes $\overline{\text{P\_TA}}$ and the transaction completes.

Figure 5-14 shows the system bus timing for a full-speed load-store **xmem** transaction where the MC88410 in not parked between transactions. In this case the system bus may be granted to another bus master between transactions. For system bus timing of a lock collision between **xmem** transactions when the MC88410 is not parked, see Figure 5-43.

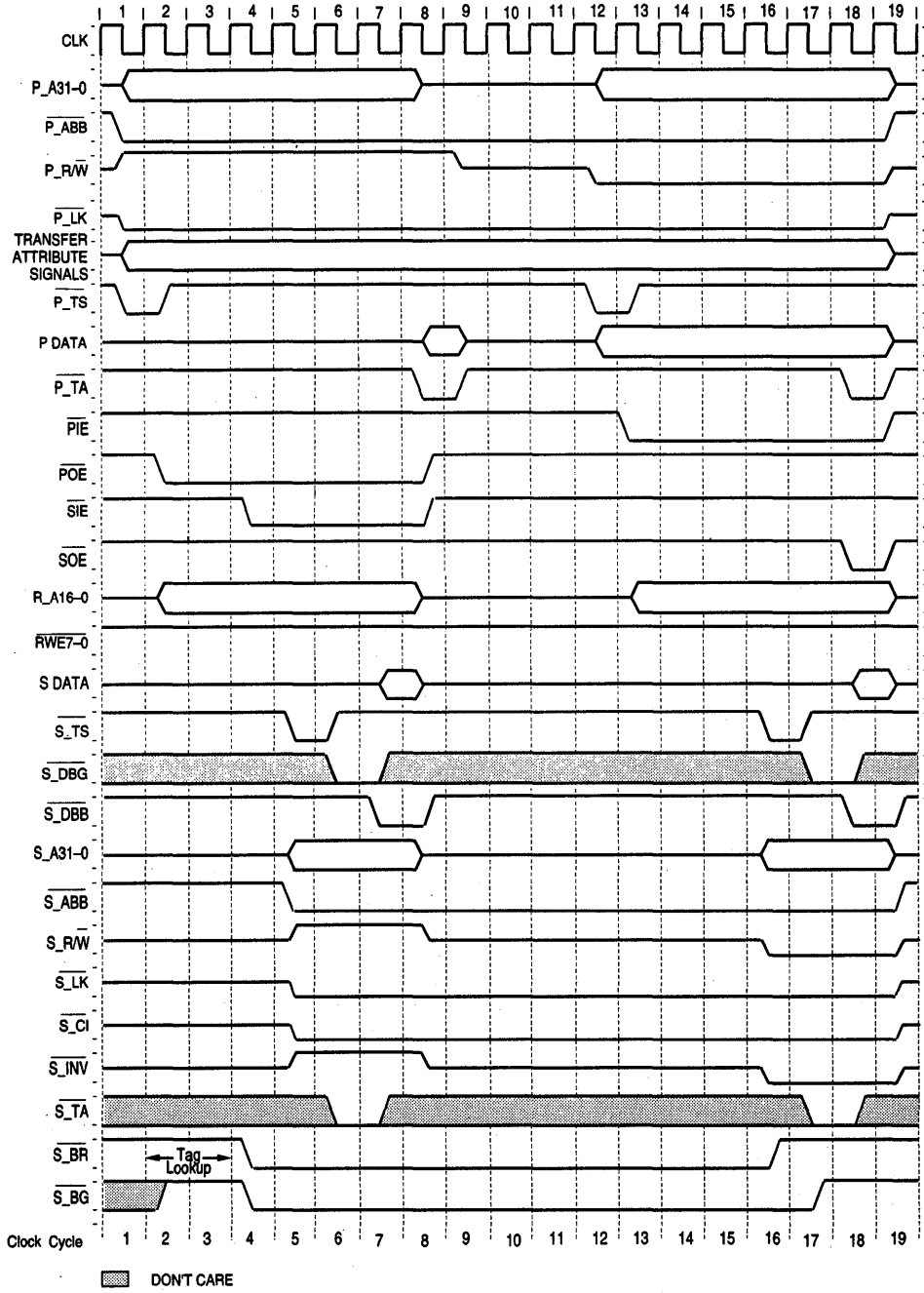


Figure 5-13. Cache-Inhibited Load-Store Locked Transaction

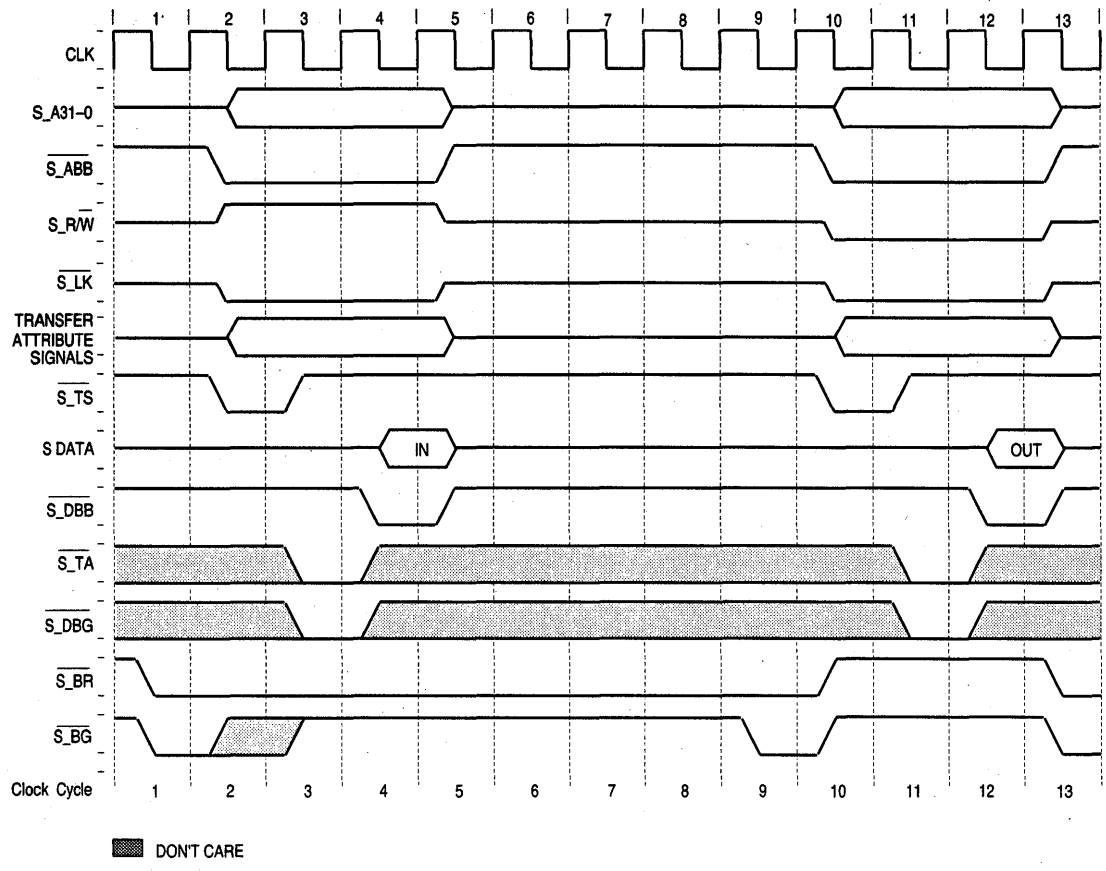


Figure 5-14. Locked Transaction Timing — Unparked Case

5

5.5.5 System Burst Transactions

Burst transactions are read or write transactions that occur as a result of single-beat transactions that miss in the secondary cache, burst processor transactions that miss in the secondary cache, and copyback transactions. Transactions in Table 5-5 with $\overline{\text{S\_TBST}}$ asserted are system bus burst transactions. Note that if a processor transaction is cache-inhibited ($\overline{\text{P\_CI}}$ is asserted) $\overline{\text{P\_TBST}}$ is ignored and a single-beat transaction occurs on the system bus. All burst transactions assert $\overline{\text{S\_MC}}$ and have similar timing characteristics. The differences between the transactions are determined by the transfer attribute signals shown in Table 5-5.

Burst transactions on the system bus interface differ from those on the processor bus interface by allowing a choice of critical-word-first or zero-word-first burst ordering. Also, the system interface transfers either 32 bytes or 64 bytes depending on the cache line size configuration. For information regarding the effect of cache line size and burst ordering on processor transactions, refer to **Section 2 Secondary Cache Operation.**

The following paragraphs describe burst transaction types and transaction timing assuming a 32-byte secondary cache line size and critical-word-first burst ordering. Transaction timing for other cache line sizes and burst ordering are described in **5.5.5.5 Burst Order and Streaming Timing Examples.**

5.5.5.1 Burst Read Transaction Types

During a burst read transaction, the MC88410 fills a secondary cache line by reading four (32-byte line size) or eight (64-byte line size) double words from memory depending on cache line size.

5.5.5.1.1 Secondary Cache Line Fill

A system bus burst read operation that results from a miss in the secondary cache is referred to as a secondary cache line fill. Secondary cache line fill transactions result from a read miss in the primary instruction or data cache and a read miss in the secondary cache. Processor single-beat read transactions which can result in a secondary cache line fill are the table search, allocate load, and load-store **xmem** transactions. Secondary cache line fills that result from a primary instruction or data cache read miss stream data to the processor while writing data to the secondary cache. The MC88410 does not assert $\overline{\text{S\_INV}}$ during secondary cache line fills.

5.5.5.1.2 Secondary Cache Read-with-Intent-to-Modify

Processor burst read operations resulting from a write miss to the primary data cache are intent-to-modify. If this read misses in the secondary cache, the system bus burst read is intent-to-modify and $\overline{\text{S\_INV}}$ is asserted to alert snooping devices to invalidate their copy of the data. Secondary cache read-with-intent-to-modify transactions are caused by touch load and data cache read-with-intent-to-modify processor transactions that miss in the secondary cache.

5.5.5.2 Burst Read Transaction Timing

The following paragraphs describe three examples of burst read timing: a full-speed secondary cache line fill, a full-speed secondary cache line fill with wait states, and a half-speed secondary cache line fill.

5.5.5.2.1 Full-Speed Secondary Cache Line Fill

Figure 5-15 shows the relative timing of the data transfer signals during a full-speed secondary cache line fill transaction. In this example the data is read into the secondary cache and streamed to the processor.

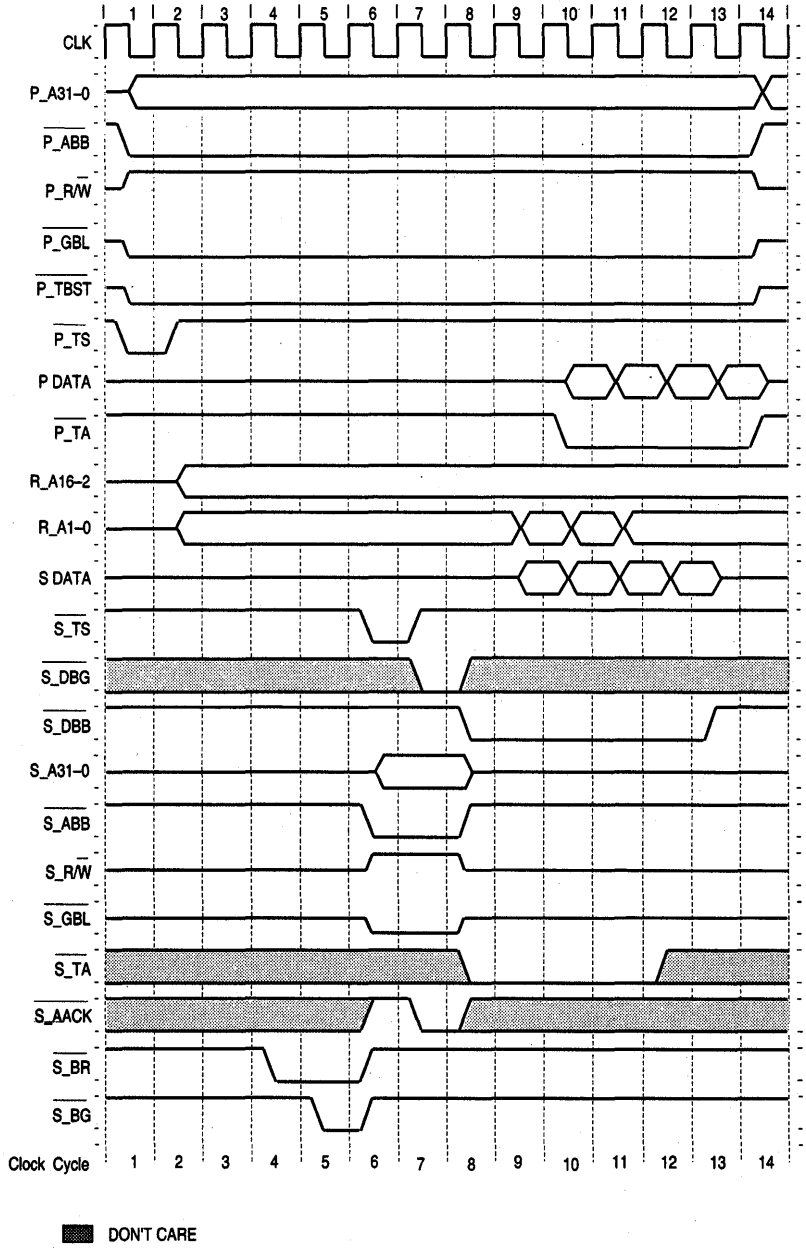


Figure 5-15. Full-Speed Line Fill Transaction Timing

Before the burst transaction begins, the MC88110 becomes the processor bus master. In clock 1 the processor asserts $\overline{\text{P\_TS}}$, $\overline{\text{P\_ABB}}$, $\overline{\text{P\_TBST}}$, and the appropriate attribute and control signals and drives the full 32-bit address onto the processor bus. The MC88410 samples the address on the next rising clock edge (clock 2). Two clocks later (clock 4) the MC88410 has determined that the read misses in the secondary cache and it requests the system bus. In this example the external arbiter asserts $\overline{\text{S\_BG}}$ in the next clock.

The MC88410 recognizes a qualified bus grant on the rising edge of clock 6. The MC88410 asserts $\overline{\text{S\_ABB}}$, $\overline{\text{S\_TS}}$, and the appropriate transfer attribute signals, negates $\overline{\text{S\_BR}}$, and drives the full 32-bit address of the requested data onto the system address bus. Assuming the assertion of $\overline{\text{S\_AACK}}$ on the rising edge of clock 8, the system address bus tenure terminates and the address and control signals are three-stated by the MC88410 in clock 8. The assertion of $\overline{\text{S\_TS}}$ also acts as a data bus request. In clock 8 the external arbiter asserts $\overline{\text{S\_DBG}}$, which is qualified by $\overline{\text{S\_DBB}}$ negated, and the MC88410 becomes the data bus master.

To indicate the status of each of the four beats of the transaction to the MC88410, the memory system then either asserts or negates the $\overline{\text{S\_TA}}$ signal. When the data is guaranteed to meet the appropriate setup and hold times with respect to the rising edge of the clock, the memory system should assert $\overline{\text{S\_TA}}$ to terminate the beat. A clock later, the address is incremented by MC88410 to the address of the next beat of the burst transaction, or, if all four beats have successfully completed, the burst transaction is terminated.

In the full-speed mode, $\overline{\text{S\_TA}}$ must be asserted one clock before the data (clock 8 in Figure 5-15) to allow the external RAM address to be pre-incremented to prevent wait states between the data bursts. In the half-speed mode, $\overline{\text{S\_TA}}$ is asserted concurrent with the data. In both cases $\overline{\text{S\_TA}}$ is asserted for the same number of clocks. If the data cannot be supplied in time during the clock cycle after the address is sampled, $\overline{\text{S\_TA}}$ must be explicitly negated until the appropriate setup and hold times are met.

The fastest case burst transaction occurs when no wait cycles are inserted by the memory system. In this example (as shown in Figure 5-15), $\overline{\text{S\_TA}}$ is asserted in clock 8 and the memory system places the first aligned double word on the data bus during clock 9 and it is latched by the secondary cache. During each of the following three clock cycles, the address is incremented by the MC88410 to reflect the address of the appropriate double word. The memory system continues to supply the secondary cache with the appropriate double words on the data bus. To signal the end of the transaction on the system bus after four data beats have been transferred, $\overline{\text{S\_TA}}$ is negated in clock 12.

On the processor bus, the MC88410 asserts $\overline{\text{P\_TA}}$ in clock 10 and the first data beat is latched by the processor. Note that $\overline{\text{P\_TA}}$ is asserted concurrent with the data and the processor receives the data one clock after it is latched by the secondary cache. For the next three clocks the MC88410 increments the address and asserts $\overline{\text{P\_TA}}$. The processor transaction completes in clock 14.

5.5.5.2.2 Full-Speed Secondary Cache Line Fill with Wait States

An example of a full-speed read miss burst transaction with wait cycles is shown in Figure 5-16. During clock 6, the MC88410 drives the full 32-bit address of the requested data onto the system address bus . Address bus tenure is terminated by the assertion of $\overline{\text{S\_AACK}}$ in clock 7. The external arbiter grants the MC88410 the data bus in clock 8. In this case, the memory system cannot provide the data until clock 13 so it negates $\overline{\text{S\_TA}}$ until clock 12. Note that the MC88410 keeps $\overline{\text{P\_TA}}$ negated, thus making the processor wait, until after it receives $\overline{\text{S\_TA}}$ asserted. In clock 12 the memory system asserts $\overline{\text{S\_TA}}$ and provides one beat of data on the system data bus which is latched by the secondary cache in clock 13.

During clock 13 the MC88410 increments the RAM address based on $\overline{\text{S\_TA}}$ being asserted in clock 12. Also during clock 13 the memory system negates $\overline{\text{S\_TA}}$ to insert a wait state. On the rising edge of clock 14 the MC88410 recognizes $\overline{\text{S\_TA}}$ negated.

Also in clock 14 the MC88410 asserts $\overline{\text{P\_TA}}$ and the first beat of data is latched by the processor on the rising edge of clock 15. The MC88410 negates $\overline{\text{P\_TA}}$ in clock 15 to wait the processor. During clock 14 the memory system asserts $\overline{\text{S\_TA}}$ to indicate that the second beat of data will be valid in clock 15.

In clock 15 the MC88410 increments the address based on $\overline{\text{S\_TA}}$ being asserted in clock 14. In clock 16 the memory system continues to assert $\overline{\text{S\_TA}}$ and drives the third beat of data to the secondary cache. During clock 16 the MC88410 asserts $\overline{\text{P\_TA}}$ and the second data beat is driven to the processor. The MC88410 increments the RAM address during clock 16 and the memory system drives the last beat of data to the secondary cache in clock 17. The system bus transaction is terminated in clock 18.

During clock 18 the MC88410 asserts $\overline{\text{P\_TA}}$ and the fourth data beat is driven to processor, completing its read transaction.

5.5.5.2.3 Half-Speed Secondary Cache Line Fill

Figure 5-17 shows a secondary cache line fill with the MC88410 in the half-speed mode. The MC88410 asserts $\overline{\text{S\_BR}}$ in system bus clock 3 ($\overline{\text{HCLK}}$ in this case) and samples $\overline{\text{S\_BG}}$ on the rising edge of system bus clock 4. When the MC88410 receives a qualified bus grant in system bus clock cycle 3, $\overline{\text{S\_TS}}$ is asserted for one system bus clock. The arbitration protocol is the same for the half-speed mode as for the full-speed mode. Transfer attribute signals are asserted for the duration of system address bus mastership.

The system bus data transaction begins in system bus clock 7 and the processor transaction begins during processor bus cycle 15. Note that the memory system must assert $\overline{\text{S\_TA}}$ in the same system bus clock as the data is sampled. The MC88410 negates $\overline{\text{P\_TA}}$ between each data transfer to match the system bus data transfer. If the processor transaction had hit in the secondary cache, data would have been provided without wait states.

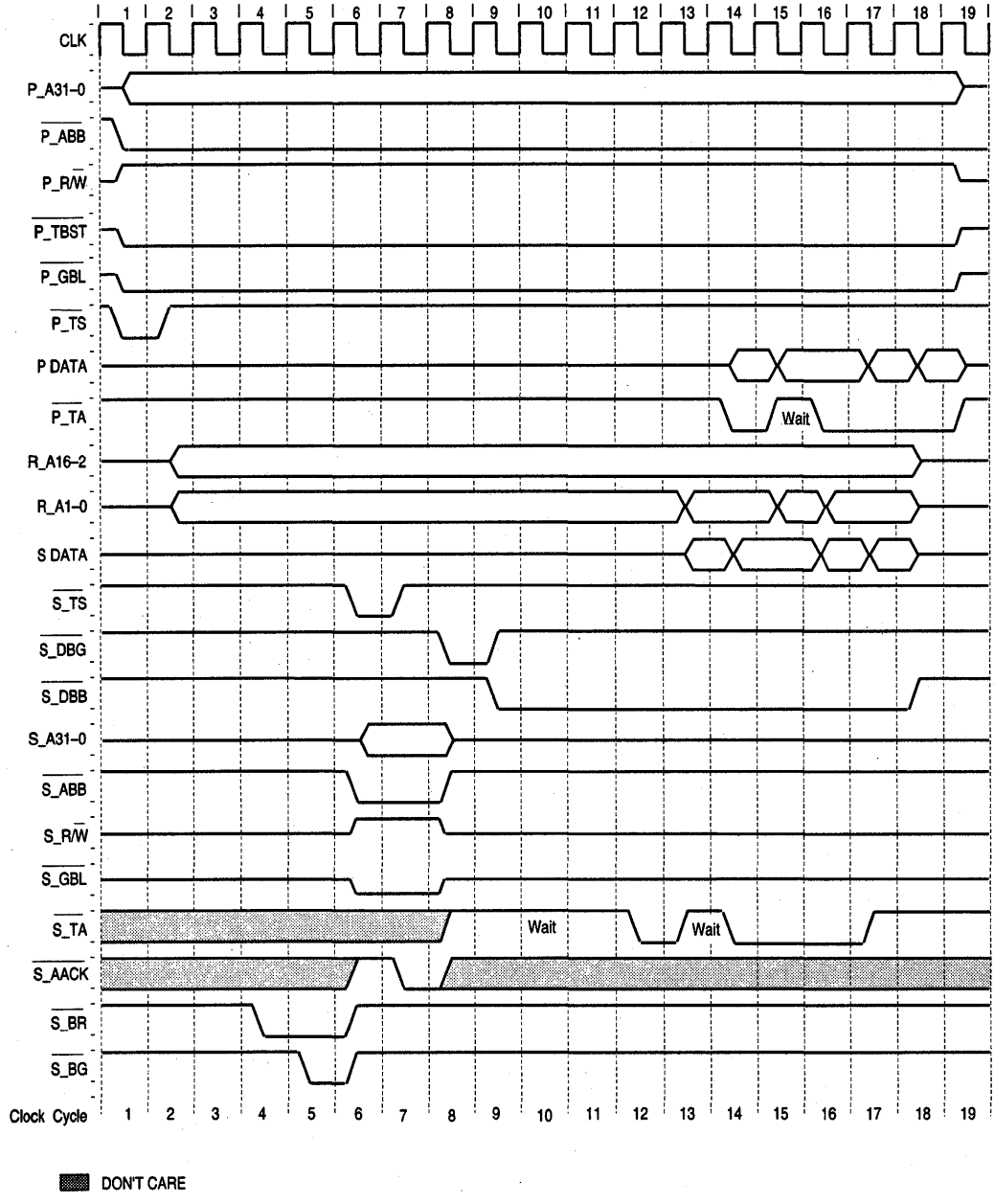


Figure 5-16. Full-Speed Burst Read Transaction Timing with Wait Cycles

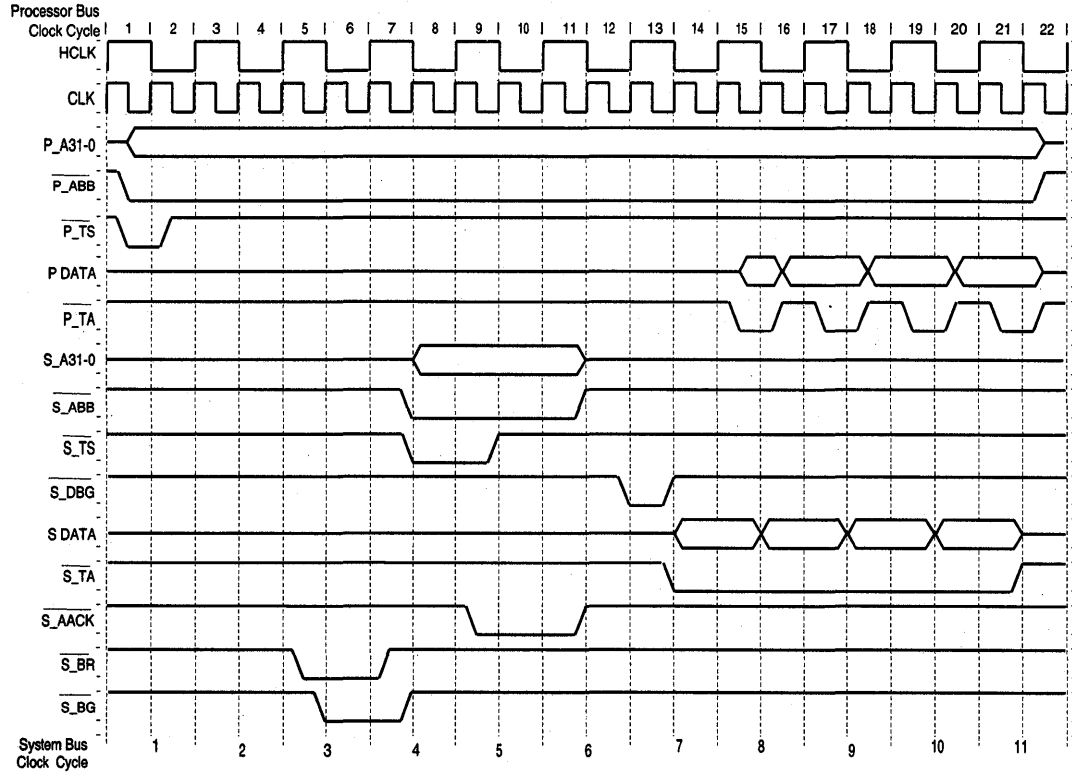


Figure 5-17. Half-Speed Streaming Line Fill

5.5.5.3 Burst Write Transaction Types

5

During a burst write transaction, the MC88410 transfers four or eight double words from a secondary cache line to main memory.

There are three types of burst write transactions: replacement copyback, snoop copyback, and flush copyback. A copyback transaction is the process of writing a modified cache line out to memory so that memory is updated. The timing and transfer attributes are the same for copyback transactions. The particular type of copyback transaction can be determined by decoding the $\overline{S\_TC3}$–$\overline{S\_TC0}$ signals. Note that the timing for the $\overline{S\_TC3}$–$\overline{S\_TC0}$ signals coincides with the timing for the system bus address signals.

5.5.5.3.1 Replacement Copyback Operation

When a read miss in the secondary cache requires a secondary cache line fill to occur, the line to be filled may contain modified data. In this case the MC88410 writes the modified data to the system bus in a four or eight double-word burst before filling the cache line. This copyback transaction is referred to as a replacement copyback. Replacement copyback transactions always start with zero-word-first ordering.

5.5.5.3.2 Snoop Copyback Operation

The MC88410 uses a bus snooping protocol to maintain cache coherency in systems where more than one bus master is allowed to access shared memory. When a snooping MC88410 has a secondary cache hit during a global write or global read-with-intent-to-modify transaction, the snooping MC88410 determines if the data is modified in the secondary or primary cache. If the line is modified, the line must be copied back to main memory before the device performing the global access can complete its transaction. This copyback transaction is referred to as a snoop copyback. The snoop copyback transaction can start with critical-word-first or zero-word-first ordering. Snoop copyback transaction timing is described in detail in **5.7.6 Secondary Cache Copyback Timing.**

5.5.5.3.3 Flush Copyback Operation

The MC88410 contains a flush mechanism that causes a copyback of all of the modified secondary cache lines in the cache or a specified page of the cache. The burst write transaction by which each cache line is transferred to memory is called a flush copyback transaction. Flush copyback transactions use zero-word-first ordering. Secondary cache flushing is described in detail in **Section 2 Secondary Cache Operation.**

5.5.5.4 Burst Write Transaction Timing

Figure 5-18 shows the relative timing of a full-speed burst write transaction caused by a copyback transaction. In this example, a read miss in the primary (MC88110) data cache causes a burst read transaction on the processor bus to fill the primary cache line. The read transaction misses in the secondary cache; however, the line to be filled in the secondary cache contains valid modified data. A replacement copyback transaction precedes the allocation of the secondary cache line. If the line to be replaced had been included in the primary data cache, a primary cache invalidate transaction would have preceded the copyback transaction to allow the processor to flush its data. This example represents the fastest case back-to-back MC88410 system bus transaction.

Before the burst transaction begins, the MC88110 becomes the processor bus master. In clock 1 the processor asserts $\overline{\text{P\_TS}}$, $\overline{\text{P\_ABB}}$, and the appropriate attribute and control signals and drives the full 32-bit address onto the processor bus. The MC88410 samples the address on the next rising clock edge (clock 2). During clock cycles 2 and 3, the MC88410 determines that the replacement copyback transaction is required and drives the address to the MCM62110 array. In clock 4, the MC88410 requests the system bus, which is granted by the external arbiter in clock 5.

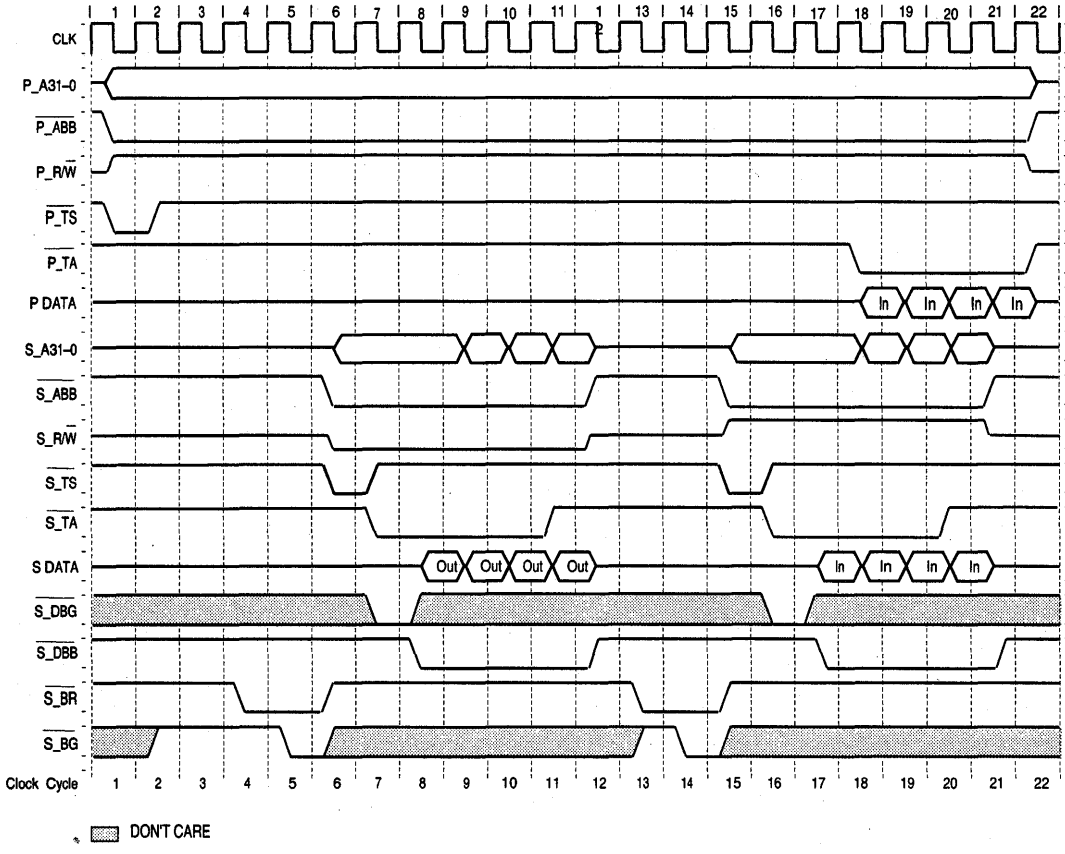


Figure 5-18. Full-Speed Read Miss Causing Replacement Copyback (Fastest Back-to-Back MC88410 Transaction)

5

During clock 6, the MC88410 initiates the burst write transaction by asserting the $\overline{\text{S\_TS}}$ and $\overline{\text{S\_ABB}}$ signals, driving the address on the system bus, and negating S_R/$\overline{\text{W}}$. If the MC88410 had been parked on the system bus, $\overline{\text{S\_TS}}$ would have been asserted in clock 4. To indicate the status of each of the four beats of the transaction to the MC88410, the memory system then either asserts or negates the $\overline{\text{S\_TA}}$ signal. When the data can be latched by the slave device, it should assert $\overline{\text{S\_TA}}$ to terminate the beat. In the next clock, the address is incremented by MC88410 to the next beat of the burst transaction, or, if all data beats have successfully completed, the burst transaction is terminated. If $\overline{\text{S\_TA}}$ is not asserted, the MC88410 continues to drive data until the transaction is terminated.

In the full-speed mode, $\overline{\text{S\_TA}}$ must be asserted one clock before the data is latched (clock 7 in Figure 5-18) to allow the external RAM address to be pre-incremented to prevent wait states between the data bursts. In the half-speed mode, $\overline{\text{S\_TA}}$ is asserted concurrent with the data. In both cases $\overline{\text{S\_TA}}$ is asserted for the same number of clocks. If the data cannot be supplied in time during the clock cycle after the address is sampled, $\overline{\text{S\_TA}}$ must be

explicitly negated until the appropriate setup and hold times are met. In this example, the memory system responds by asserting $\overline{\text{S\_TA}}$ in clock 7. The MC88410 drives the data during clock 8 and asserts $\overline{\text{S\_DBB}}$. The MC88410 increments the address and drives the data in each subsequent clock until the replacement copyback transaction is complete.

In the clock cycle after the last data beat of the burst write transaction (clock 13), the MC88410 asserts $\overline{\text{S\_BR}}$ to request the system bus again for the burst read transaction to fill the cache line. If the MC88410 had been parked on the system bus, $\overline{\text{S\_TS}}$ would have been asserted in clock 13. In this example, the MC88410 recognizes a qualified bus grant during the rising edge of clock 15 and asserts $\overline{\text{S\_TS}}$ to initiate the transaction. At the same time it drives the address of the data to be read, asserts $\overline{\text{S\_ABB}}$ and negates S_R/$\overline{\text{W}}$.

The memory system asserts $\overline{\text{S\_TA}}$ in clock 16 and places the first double word on the data bus during clock 17, which is latched by the secondary cache. During each of the following three clock cycles, the address is incremented by the MC88410 to reflect the address of the appropriate double word. The memory system continues to supply the secondary cache with the appropriate double words on the data bus. After four data beats have been transferred, $\overline{\text{S\_TA}}$ is negated in clock 20. The system bus address, data, and control signals are three-stated by the MC88410 in clock 21.

On the processor bus, the MC88410 asserts $\overline{\text{P\_TA}}$ in clock 18 and the first data beat is latched by the processor. Note that $\overline{\text{P\_TA}}$ is asserted concurrent with the data and the processor receives the data one clock after it is latched by the secondary cache. For the next three clocks the MC88410 increments the address and asserts $\overline{\text{P\_TA}}$. The processor transaction completes in clock 22.

Figure 5-19 shows the relative timing of a burst write transaction with wait states inserted during the last data beat. This transaction is identical to Figure 5-18 except that it assumes that $\overline{\text{S\_AACK}}$ is asserted in clock 7 and the memory system negates $\overline{\text{S\_TA}}$ in clock 10 for two clocks to insert wait states. The MC88410 continues to drive the last beat of data until it receives the last $\overline{\text{S\_TA}}$ during clock 13. The MC88410 ends the copyback transaction in the following clock. The burst read transaction follows in the same manner as Figure 5-18.

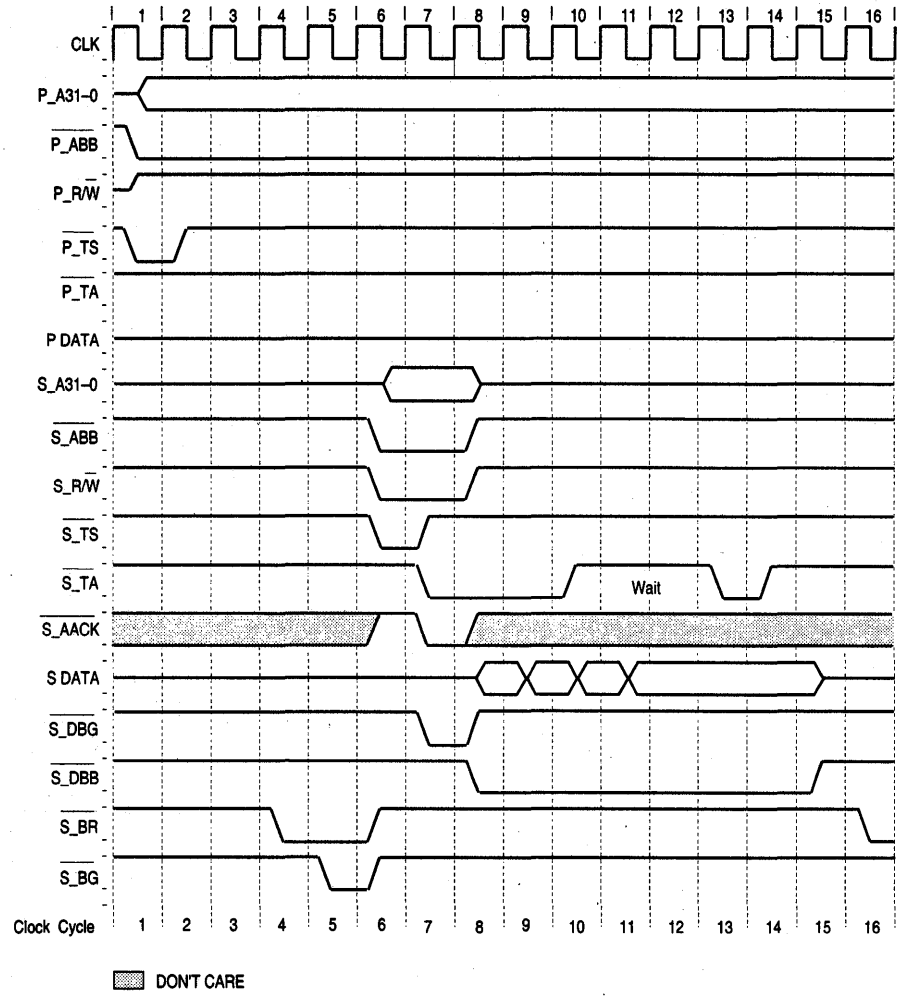


Figure 5-19. Full-Speed Burst Write with Wait Cycles

5.5.5.5 Burst Order and Streaming Timing Examples

The MC88410 increments the address to provide all addresses (four or eight) during burst transactions. The order of burst addressing on the system interface is programmable at reset between zero-word-first and critical-word-first. This ordering applies to both secondary cache line fills and secondary cache snoop copyback transactions. Replacement copyback and flush copyback transactions always start with word zero.

The MC88410 uses data streaming to reduce the penalty seen by the processor on secondary cache misses. Streaming means that as data is being written into the secondary cache from memory, it is also passed onto the processor bus to satisfy the original request. This concept is straightforward in the previous timing diagrams dealing with configurations using a 32-byte secondary cache line size and critical-word-first burst

ordering. In this case, the four data beats from memory pass through the secondary cache to the processor bus. Upon transfer of the last word to the processor bus, the operation is completed. The transfer is more complex when dealing with 64-byte secondary cache line sizes and/or zero-word-first burst ordering on the system bus.

Figure 5-20 shows a read miss with a 32-byte secondary cache line size and zero-word-first ordering. Data beat 2 is the processor's requested double word. The data transfer is initiated in the same way as in Figure 5-15. However, in clocks 9 and 10, $\overline{\text{P\_TA}}$ remains negated while data beats 0 and 1 are written to the secondary cache.

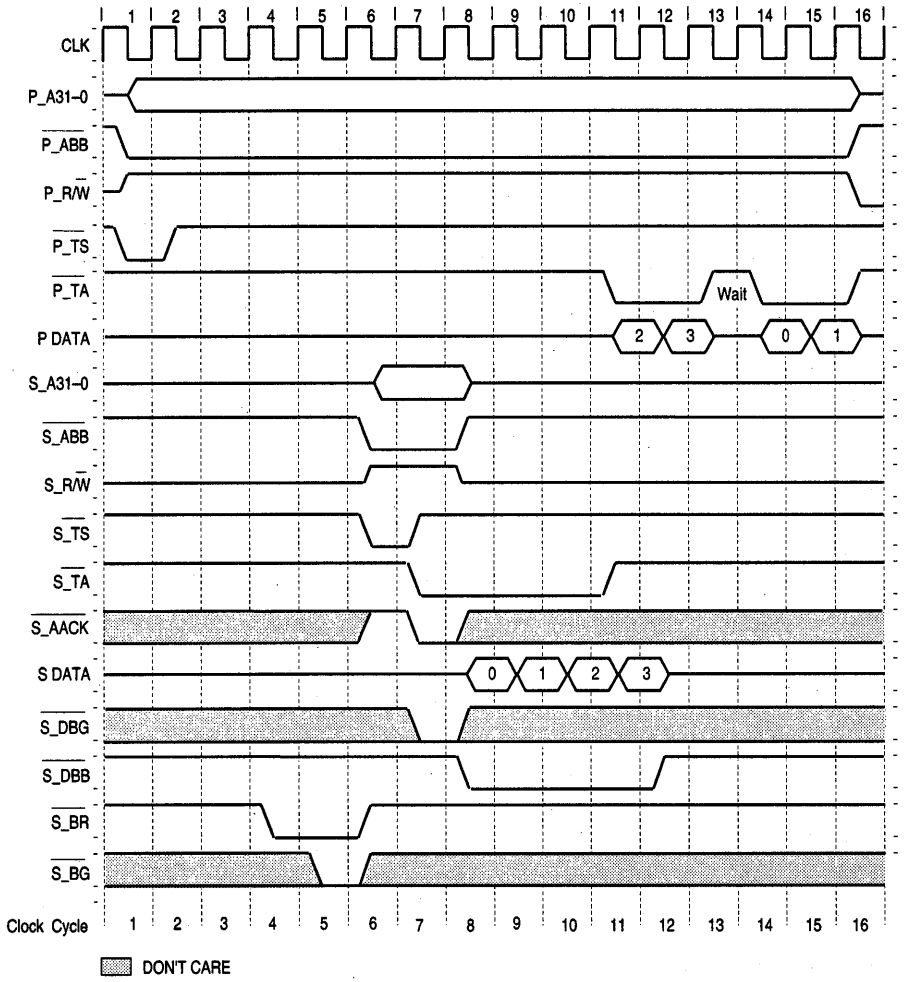


Figure 5-20. Streaming — 32 Byte Cache Line Size with Zero-Word-First

In clocks 11 and 12, $\overline{\text{P\_TA}}$ is asserted and data beats 2 and 3 are streamed to the processor. During clock 13, $\overline{\text{P\_TA}}$ is negated to insert a wait state to the processor to allow the

MC88410 to wrap around to the address of beat 0. During clocks 14 and 15, data beats 0 and 1 are written to the processor from the secondary cache. With the transfer of data beat 1 the transaction completes.

Figure 5-21 shows a secondary cache read miss with a 64-byte cache line size and critical-word-first ordering. Again, the requested double word from the processor is data beat 2. This transaction is similar to Figure 5-21 except that eight data beats are written to the secondary cache. In clocks 9 and 10, data beats 2 and 3 are streamed to the processor and in clocks 15 and 16 data beats 0 and 1 are streamed to the processor.

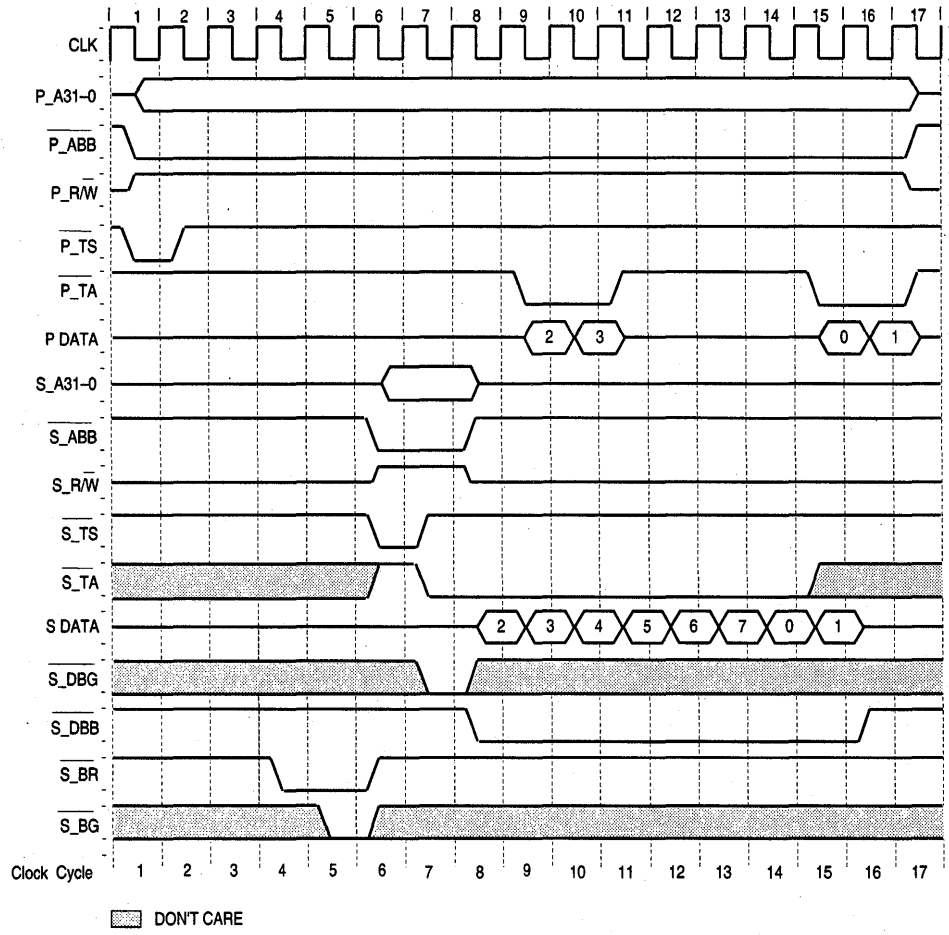


Figure 5-21. Streaming — 64 Byte Cache Line Size with Critical-Word-First

Figure 5-22 shows a full-speed read miss with a 64-byte secondary cache line size with zero-word-first burst order. Data beat 2 is the requested double word from the processor and it is streamed from system bus in clock 11. In the next clock, data beat 3 follows. After data beat 7 is written to the secondary cache (clock 16), data beats 0 and 1 are

written from the secondary cache to the processor and the transaction completes in clock 19.

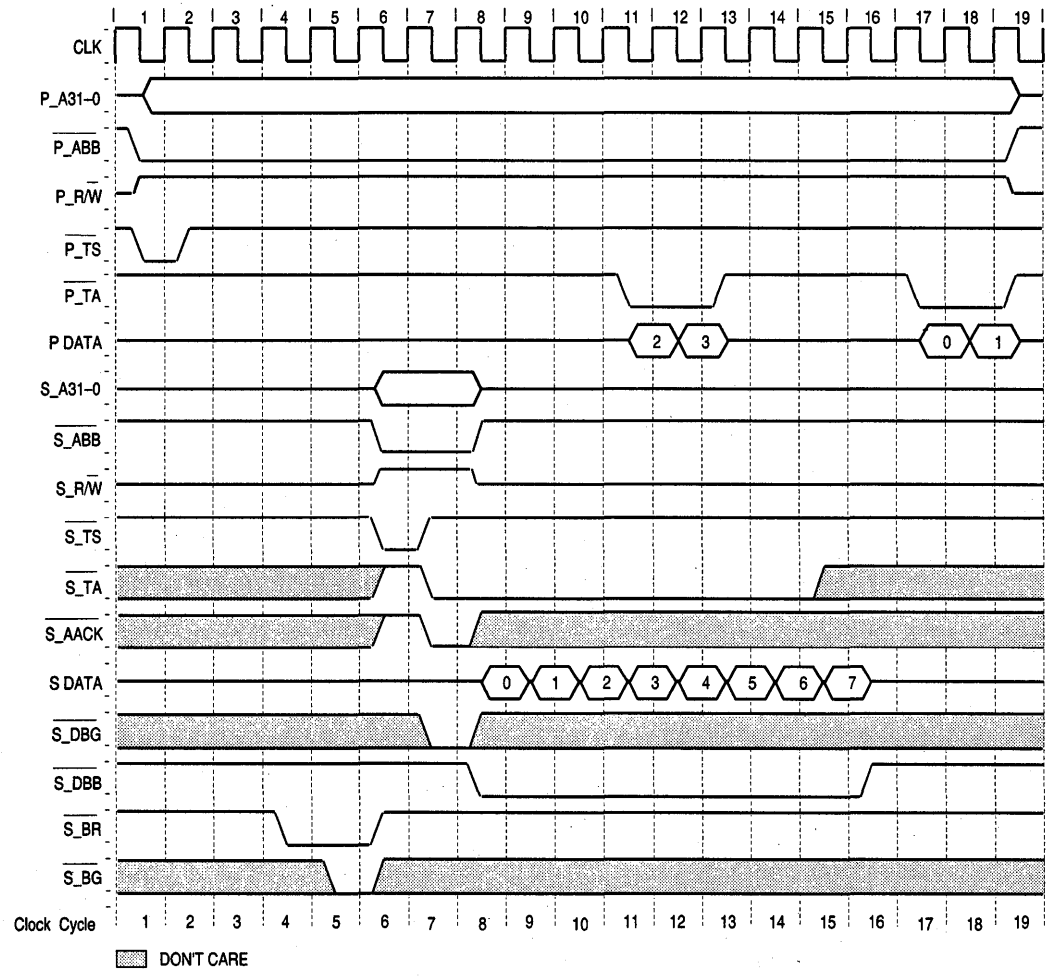


Figure 5-22. Streaming – 64 Byte Cache Line Size with Zero-Word-First

System errors detected during streaming must be passed on to the processor. In configurations using a 64-byte secondary cache line size it would be possible to stream an entire primary cache line to the processor before the secondary cache has completed its allocation. To allow the MC88410 to signal the processor for errors detected during the secondary cache line fill, the processor will not receive its last $\overline{\text{P\_TA}}$ until the secondary cache line fill has completed. For more information, see **5.6.4 Transfer Error Termination.**

5.6 SYSTEM BUS TRANSACTION TERMINATION

The following paragraphs describe the different methods for terminating transactions on the MC88410 bus. Transactions may be terminated normally, indicating that the transfer was completed successfully or terminated with an error or a retry indication. Two types of retry terminations are possible: transfer retry and address retry. The address retry terminates the transaction of the current address bus master. The transfer retry terminates the transaction of the current data bus master.

The state of several input signals to the MC88410 determines the termination for each transaction on the MC88410 bus. These signals are $\overline{\text{S\_DBB}}$, $\overline{\text{S\_TA}}$, $\overline{\text{S\_TEA}}$, $\overline{\text{S\_TRTRY}}$, and $\overline{\text{S\_ARTRY}}$. Table 5-6 depicts the encodings of $\overline{\text{S\_DBB}}$, $\overline{\text{S\_TA}}$, $\overline{\text{S\_TEA}}$, $\overline{\text{S\_TRTRY}}$, $\overline{\text{S\_ARTRY}}$, and the corresponding types of transaction termination. The assertion of $\overline{\text{S\_DBB}}$ indicates that the MC88410 is the current data bus master. The address retry assumes it is qualified by $\overline{\text{S\_TA}}$ or $\overline{\text{S\_DBB}}$ and $\overline{\text{S\_TRTRY}}$.

Table 5-6. Transaction Termination Encodings

$\overline{\text{S\_DBB}}$	$\overline{\text{S\_TA}}$	$\overline{\text{S\_TEA}}$	$\overline{\text{S\_TRTRY}}$	$\overline{\text{S\_ARTRY}}$	Termination
A	A	N	N	N	Normal
A	x	A	x	x	Error
A	x	N	A	x	Transfer retry
A	A	N	A	A	Address retry

A = Asserted
N = Negated
x = Don't Care

Some system bus transaction termination signals are sampled earlier by the MC88410 than the MC88110. Unlike the MC88110, the MC88410 recognizes $\overline{\text{S\_TA}}$ and $\overline{\text{S\_TRTRY}}$ in the clock that $\overline{\text{S\_DBG}}$ is asserted (with $\overline{\text{S\_DBB}}$ negated), which is one clock before the MC88410 takes data bus mastership by asserting $\overline{\text{S\_DBB}}$. Table 5-7 summarizes when the signals are recognized. A qualified data bus grant is defined as $\overline{\text{S\_DBG}}$ asserted and $\overline{\text{S\_DBB}}$ negated.

Table 5-7. Transaction Termination Signal Sampling

Signal	Timing
$\overline{\text{S\_DBG}}$	Sampled in the same clock as $\overline{\text{S\_TS}}$ and after
$\overline{\text{S\_TA}}$	Sampled during a qualified data bus grant, which may be in same clock as $\overline{\text{S\_TS}}$ and while $\overline{\text{S\_DBB}}$ is asserted
$\overline{\text{S\_TRTRY}}$	Sampled during a qualified data bus grant and while $\overline{\text{S\_DBB}}$ is asserted, but not during the clock that $\overline{\text{S\_TS}}$ is asserted
$\overline{\text{S\_TEA}}$	Sampled while $\overline{\text{S\_DBB}}$ is asserted
$\overline{\text{S\_ARTRY}}$	Sampled during and one clock after $\overline{\text{S\_AACK}}$ asserted, or during 1st data beat if $\overline{\text{S\_AACK}}$ has not been asserted.

Figure 5-23 shows two examples to illustrate termination timing. Clock cycle 1 shows the fastest case of data bus arbitration with a qualified data bus grant and $\overline{\text{S\_TA}}$ asserted. Note that $\overline{\text{S\_TA}}$ is sampled during clock 1 but $\overline{\text{S\_TRTRY}}$ is not. In clock 2 the MC88410 transfers the data beat and detects an address retry even though $\overline{\text{S\_AACK}}$ has not been asserted. The MC88410 relinquishes the address bus during clock 3.

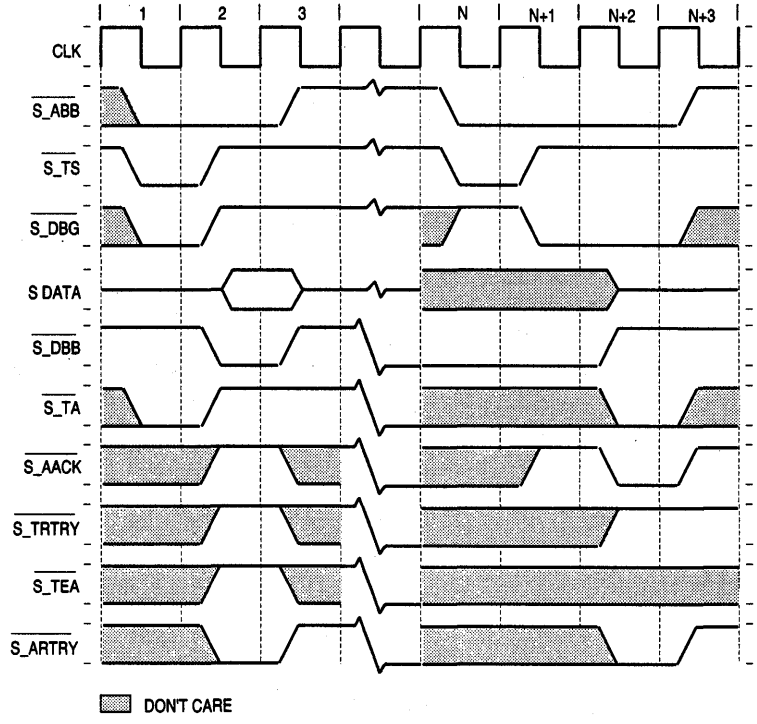


Figure 5-23. Transaction Termination Signal Timing

In clock N, the MC88410 asserts $\overline{S\_TS}$ to initiate a transaction and request the data bus. However, another bus master controls the data bus and $\overline{S\_DBG}$ is negated by the external arbiter. On the rising edge of clock N+2 the MC88410 detects $\overline{S\_DBG}$ asserted and samples $\overline{S\_AACK}$ but does not have a qualified data bus grant since $\overline{S\_DBB}$ is being asserted by another bus master. During clock N+2 $\overline{S\_DBB}$ is negated, giving the MC88410 a qualified data bus grant on the rising edge of clock N+3, which causes it to sample $\overline{S\_TRTRY}$. At the same time the assertion of $\overline{S\_AACK}$ is detected and therefore so is the assertion of $\overline{S\_ARTRY}$. The MC88410 relinquishes the address bus during clock N+3.

The following paragraphs describe normal termination, transfer retry termination, and termination with address retry or error indication and explain their relative timing.

5.6.1 Normal Transaction Termination with $\overline{S\_TA}$

The assertion of $\overline{S\_TA}$ while $\overline{S\_TRTRY}$ and $\overline{S\_TEA}$ are negated signals a normal termination to the MC88410. The assertion of either $\overline{S\_TRTRY}$ or $\overline{S\_TEA}$ overrides $\overline{S\_TA}$ and signals either a transfer retry or an error. In the full-speed mode, normal termination indicates that the memory system is ready to supply or latch the data in the following clock. For a read transaction, the data is valid on the data bus and may be latched by the MC88410 in the following clock. For a write transaction, the memory system will be able to accept the data on the following clock. In the half-speed mode, normal termination indicates to the MC88410 that the current data transfer has completed successfully. For a read transaction, the data is valid on the data bus and may be latched by the MC88410. For a write transaction, the data has been accepted by the memory system.

The $\overline{S\_TA}$ signal is monitored only when the MC88410 has taken mastership of the data bus. The MC88410 can take mastership of the bus during the clock that $\overline{S\_TS}$ is asserted if the memory system and arbiter can respond early enough. The memory system either asserts or negates the $\overline{S\_TA}$ signal to indicate the status of the transaction to the MC88410. When the data is guaranteed to meet the appropriate setup and hold times with respect to the rising edge of the clock, the memory system should assert $\overline{S\_TA}$ to terminate the transaction. In the full-speed mode, $\overline{S\_TA}$ must be asserted one clock before the data is supplied or latched to allow the MC88410 to increment the address to the secondary cache for burst transactions. If the data cannot be supplied or latched in time during the clock cycle after the address is sampled, $\overline{S\_TA}$ must be explicitly negated until the appropriate setup and hold times are met.

In the half-speed mode $\overline{S\_TA}$ should be asserted concurrent with the data. If the data cannot be supplied or latched during the clock cycle, $\overline{S\_TA}$ must be explicitly negated until the appropriate setup and hold times are met.

For single-beat transactions, the MC88410 ends the transaction after $\overline{S\_TA}$ is asserted. To end the transaction, the MC88410 releases the data bus and negates $\overline{S\_DBB}$. If it is also the current address bus master, it releases mastership of the address bus and negates $\overline{S\_ABB}$. For burst transactions, each beat of the burst must be terminated by $\overline{S\_TA}$ before the transaction is completed. Figure 5-24 shows both single-beat and burst transactions that are completed by normal transaction termination. This example assumes that MC88410 remains parked on the bus between transactions ($\overline{S\_BG}$ remains asserted).

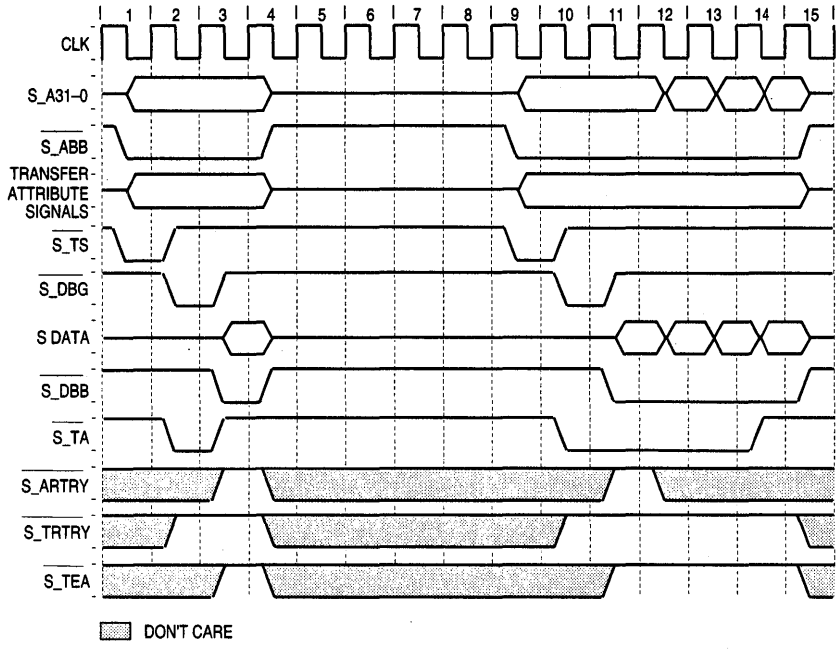


Figure 5-24. Full-Speed Normal Transaction Terminations with S_TA

In the first clock cycle in Figure 5-24, the MC88410 starts a new transaction by asserting S_TS and S_ABB and driving the address. The assertion of S_TS is interpreted as a data bus request. In clock 2, the MC88410 is granted the data bus, and becomes the data bus master by asserting S_DBB in the next clock. In clock 2 the MC88410 also detects that S_TA is asserted, while S_TRTRY is negated, so it completes the transaction and relinquishes data bus mastership. Note that S_TRTRY is sampled throughout the data bus tenure (qualified data bus grant through the last data beat transferred) while S_TEA is sampled only while the data bus is busy (S_DBB asserted). In this example, S_AACK is not used, so S_ARTRY is sampled during the first data beat.

Assuming that S_BG remains asserted, the MC88410 can maintain mastership of the address bus and begin a burst transaction without re-arbitration in clock 9. It becomes the data bus master in clock 10 and detects that S_TA is asserted. This signals the end of the first double-word transfer of the burst. After three more clocks of S_TA asserted successfully (each signaling the end of another double-word transfer), the transaction is complete. Wait states may be added when the MC88410 is the data bus master by not asserting S_TA. There is no limit to the number of wait states that may be inserted for any beat of a transaction.

5.6.2 Transfer Retry Termination

The assertion of S_TRTRY and the negation of S_TEA during the data bus tenure of an MC88410 transaction causes a transfer retry termination of the transaction. If the MC88410 is the current address bus master, but not the data bus master, it does not

recognize an assertion of $\overline{\text{S\_TRTRY}}$. Also, the assertion of $\overline{\text{S\_TEA}}$ has a higher priority than $\overline{\text{S\_TRTRY}}$, so the processor detects an error termination if both signals are asserted during a transaction.

The response of the MC88410 to $\overline{\text{S\_TRTRY}}$ depends on when in the data bus tenure $\overline{\text{S\_TRTRY}}$ is asserted. The response is divided into three possibilities: very early $\overline{\text{S\_TRTRY}}$, early $\overline{\text{S\_TRTRY}}$, and late $\overline{\text{S\_TRTRY}}$. Very early and early retries are received on or before the first data beat and late retries occur after the first data beat. Note that in the full-speed mode the response is coincident with the data, not $\overline{\text{S\_TA}}$.

5.6.2.1 Very Early Assertion of $\overline{\text{S\_TRTRY}}$

If $\overline{\text{S\_TRTRY}}$ is asserted at the same time as $\overline{\text{S\_DBG}}$, the transaction does not transfer the data. When the MC88410 receives a qualified $\overline{\text{S\_TRTRY}}$ ($\overline{\text{S\_DBB}}$ negated) with $\overline{\text{S\_DBG}}$, it retries the entire transaction including the arbitration, address, and data phase without asserting $\overline{\text{S\_DBB}}$. This operation is different from the MC88110 protocol which only recognizes $\overline{\text{TRTRY}}$ on the clock following $\overline{\text{S\_DBG}}$. Conceptually it is the same as if the MC88410 were never granted the bus. The MC88410 will re-intitiate all transactions (including copybacks) if they are terminated with a very early retry. The processor is not made aware of very early retries. Clock cycles 1 to 4 of Figure 5-25 illustrate very early assertion of $\overline{\text{S\_TRTRY}}$.

5.6.2.2 Early Assertion of $\overline{\text{S\_TRTRY}}$

If $\overline{\text{S\_TRTRY}}$ is asserted before or during the first data beat (but after $\overline{\text{S\_BG}}$ asserted), the MC88410 releases the system address and data bus and re-arbitrates for the transaction by asserting $\overline{\text{S\_BR}}$ in the following clock. Unlike very early assertion of $\overline{\text{S\_TRTRY}}$, $\overline{\text{S\_DBB}}$ is asserted with a qualified data bus grant. The processor is not made aware of early retries. Clock cycles N to N+4 in Figure 5-25 illustrate the early assertion of $\overline{\text{S\_TRTRY}}$.

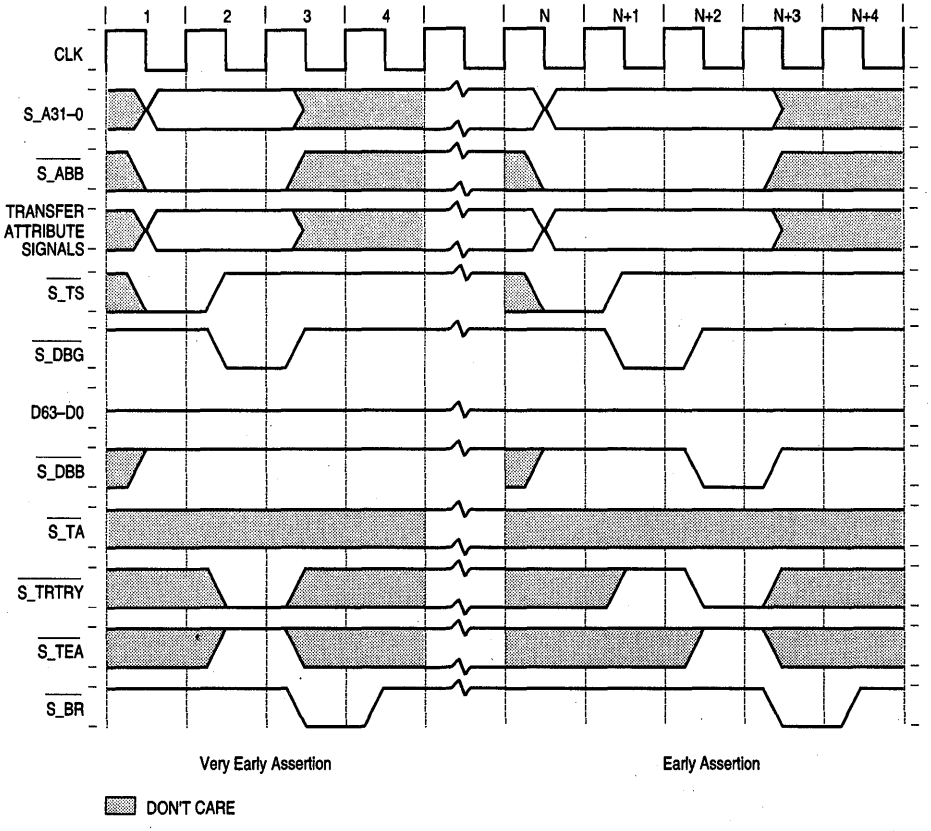


Figure 5-25. Very Early and Early Assertion of S_TRTRY

5.6.2.3 Late Assertion of S_TRTRY

Assertion of S_TRTRY after the first data beat is referred to as a late retry. A late retry causes the MC88410 to abort the transaction. If the processor is involved in the data transfer, then it is retried with the assertion of P_TRTRY. The processor receives an early retry if P_TRTRY is asserted before the first P_TA is asserted or while the first P_TA is asserted. The processor must reinitiate the transaction. For a transfer retry that occurs on the second, third, or fourth beat of a burst to the processor, the processor immediately ends the transaction. If the transaction was a processor burst read, the burst is not reinitiated later. Note that the MC88410 can receive a late S_TRTRY while the processor receives an early P_TRTRY, if S_TRTRY is asserted after the first beat of data to the MC88410 but before it is latched by the processor.

Figure 5-26 shows an example of a zero-word-first burst read in the full-speed mode that is streamed to the processor. The critical word is data beat 2. In this example, the MC88410 detects a late S_TRTRY after the first data beat is latched from the system data bus by the secondary cache. The MC88410 asserts P_TRTRY to retry the processor. Since

the critical word is not latched by the secondary cache, the first $\overline{\text{P\_TA}}$ is not asserted to the processor. The MC88110 interprets this as an early retry and initiates the transaction again in clock 14.

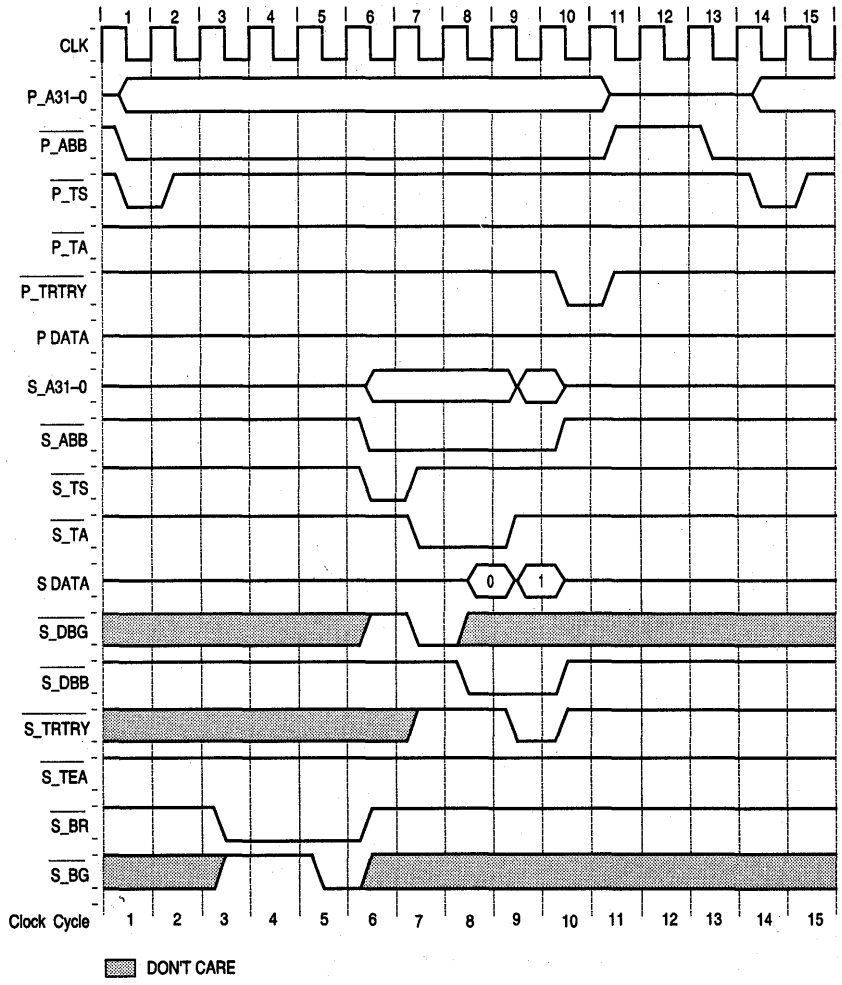


Figure 5-26. Late Assertion of $\overline{\text{S\_TRTRY}}$ with Propagation to Processor

If a snoop copyback transaction is terminated by a late retry, the MC88410 does not reinitiate the copyback. The secondary cache line remains in a modified state and the MC88410 continues snooping. The cache line must be snooped again to reinitiate the copyback transaction. A late $\overline{\text{S\_TRTRY}}$ during a secondary cache line fill causes the main tag of the secondary cache line to be invalidated. If the MC88410 receives a late $\overline{\text{S\_TRTRY}}$ during a flush operation (flush page or flush all), the MC88410 increments to the next tag entry and the tag entry involved in the retry remains in the cache. Note that the retry is not signaled back to the processor.

5.6.3 Address Retry Transaction Termination

The $\overline{\text{S\_ARTRY}}$ signal is an input that indicates to the initiating bus master that another bus master has requested that it terminate the transaction, relinquish mastership of the address bus, and retry the transaction at a later time. The $\overline{\text{S\_SSTAT1}}$ signal is asserted by another bus master to indicate that it has a modified copy of the data to the current address and the transaction must be retried. The timing for the $\overline{\text{S\_SSTAT1}}$ and $\overline{\text{S\_ARTRY}}$ signals allow the $\overline{\text{S\_SSTAT1}}$ output to be directly or indirectly connected to the $\overline{\text{S\_ARTRY}}$ input of other MC88410s.

When the $\overline{\text{S\_AACK}}$ signal is asserted by the memory system to indicate that the current address has been latched, the bus master relinquishes mastership of the address bus. In this way, an alternate bus master can initiate a transaction while the data from the previous transaction is still being transferred. In systems using this protocol, $\overline{\text{S\_AACK}}$ is also used to qualify $\overline{\text{S\_ARTRY}}$. The $\overline{\text{S\_ARTRY}}$ signal may be asserted before $\overline{\text{S\_AACK}}$ is asserted (but it must remain asserted until $\overline{\text{S\_AACK}}$ is asserted), when $\overline{\text{S\_AACK}}$ is first asserted, or during the first clock cycle after $\overline{\text{S\_AACK}}$ is asserted.

Note that if $\overline{\text{S\_TRTRY}}$ is asserted during address bus tenure, the transaction will be reinitiated begining with address bus arbitration.

If $\overline{\text{S\_AACK}}$ is negated throughout the transaction but the MC88410 is the address and data bus master ($\overline{\text{S\_ABB}}$ and $\overline{\text{S\_DBB}}$ asserted), $\overline{\text{S\_ARTRY}}$ is sampled during the first data beat of a transaction. In this case, $\overline{\text{S\_ARTRY}}$ is ignored after $\overline{\text{S\_ABB}}$ is negated. Note that $\overline{\text{S\_ARTRY}}$ is sampled with data, which is not necessarily during the first $\overline{\text{S\_TA}}$ (in the full-speed mode).

Figure 5-27 shows the qualification window for $\overline{\text{S\_ARTRY}}$ using $\overline{\text{S\_AACK}}$. Note that the figure shows $\overline{\text{S\_ARTRY}}$ asserted one clock cycle after $\overline{\text{S\_TS}}$. This would not be possible if the snooping bus master were an MC88410 because it takes two clock cycles for the MC88410 to determine whether there was a snoop hit; however, the MC88410 may be connected to a device that can assert $\overline{\text{S\_ARTRY}}$ in one clock.

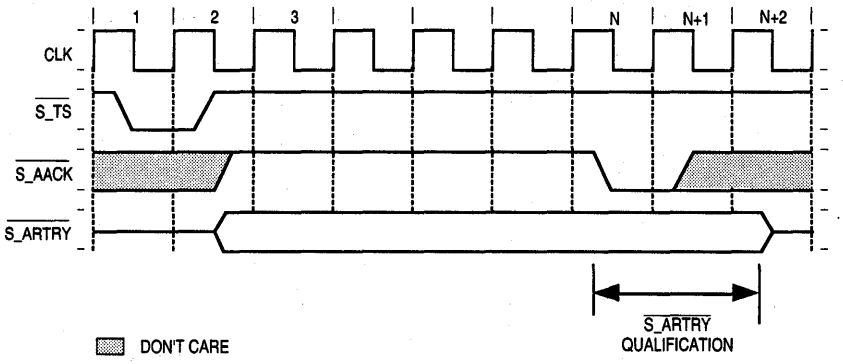


Figure 5-27. $\overline{\text{S\_ARTRY}}$ Qualification with $\overline{\text{S\_AACK}}$

When the initiating bus master detects the qualified assertion of $\overline{\text{S\_ARTRY}}$, it terminates the transaction, releases mastership of the address bus, and reinitiates the transaction. If a qualified $\overline{\text{S\_ARTRY}}$ occurs before or coincident with a qualified data bus grant, the initiating MC88410 does not assert $\overline{\text{S\_DBB}}$. When an MC88410 that is requesting the bus detects that $\overline{\text{S\_ARTRY}}$ is asserted and that $\overline{\text{S\_ABB}}$ was asserted on the previous clock cycle, it removes its bus request and ignores any bus grant. The MC88410 then blocks its bus requests by not asserting $\overline{\text{S\_BR}}$ until $\overline{\text{S\_ARTRY}}$ is negated. For a detailed description of bus blocking, see **5.7.4 Bus Request Blocking.**

5.6.4 Transfer Error Termination

The assertion of $\overline{\text{S\_TEA}}$ while the MC88410 is the data bus master results in an immediate error termination. If the processor is waiting for the completion of a transaction, $\overline{\text{P\_TEA}}$ is asserted to terminate its transaction. The assertion of $\overline{\text{S\_TEA}}$ overrides the assertion of either $\overline{\text{S\_TA}}$ or $\overline{\text{S\_TRTRY}}$ and results in an error termination. The MC88410 relinquishes mastership of the data bus, and, if it is also the address bus master, it relinquishes mastership of the address bus. If there is a different address bus master, the address bus master ignores the assertion of $\overline{\text{S\_TEA}}$. The MC88410 begins monitoring $\overline{\text{S\_TEA}}$ when it takes mastership of the data bus by asserting $\overline{\text{S\_DBB}}$.

If the transfer is a snoop copyback and $\overline{\text{S\_TEA}}$ is asserted, the snoop hit will be forgotten and the secondary cache line is left valid. If the transaction is a secondary cache line fill and the secondary cache line has been partially loaded before the $\overline{\text{S\_TEA}}$ is received, the secondary cache line is marked invalid. If $\overline{\text{S\_TEA}}$ is asserted during a replacement copyback caused by a processor transaction, $\overline{\text{P\_TEA}}$ will be asserted to the processor. If the MC88410 receives an error during a flush operation (flush page or flush all), the MC88410 increments to the next tag entry and the tag entry involved in the error remains in the cache. Note that the error is not signaled back to the processor.

Figure 5-28 shows the timing of a transfer error termination for either a single-beat transaction, or the first beat of a burst transaction. The transaction begins in clock 1, with the MC88410 becoming the data bus master in clock 2. On the rising edge of clock 3, $\overline{\text{S\_TEA}}$ is asserted. The MC88410 ends the transaction and releases mastership of both the data bus and the address bus. If the processor is involved in the data transfer, the MC88410 also asserts $\overline{\text{P\_TEA}}$.

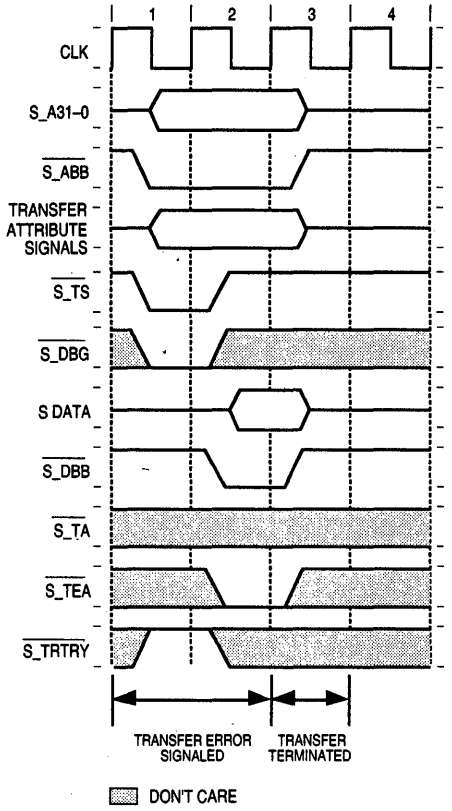


Figure 5-28. Transfer Error Termination

If data is being streamed to the processor and $\overline{\text{S\_TEA}}$ is asserted, $\overline{\text{P\_TEA}}$ must be asserted to the processor. In a cache configuration with a 64-byte line size the entire primary data cache line could be streamed to the processor before the secondary line fill has completed. Since the primary cache cannot have a valid primary cache line which is not allocated in the secondary cache (to maintain vertical cache coherency), the processor does not receive its last $\overline{\text{P\_TA}}$ until the secondary cache line has completed its allocation. This allows the MC88410 to assert $\overline{\text{P\_TEA}}$ on the last data beat if necessary.

Figure 5-29 shows the timing for a transfer error termination that occurs during the last beat of a secondary cache line fill that is being streamed to the processor. In this example, the critical word is word 0 (zero). Note that all four beats could have been sent to the processor, but that would not have allowed the MC88410 to assert $\overline{\text{P\_TEA}}$ to the processor in beat 7 of the system bus data transfer. Instead, $\overline{\text{P\_TA}}$ is negated during clock 15 and although data is driven during the following beats, it is not valid.

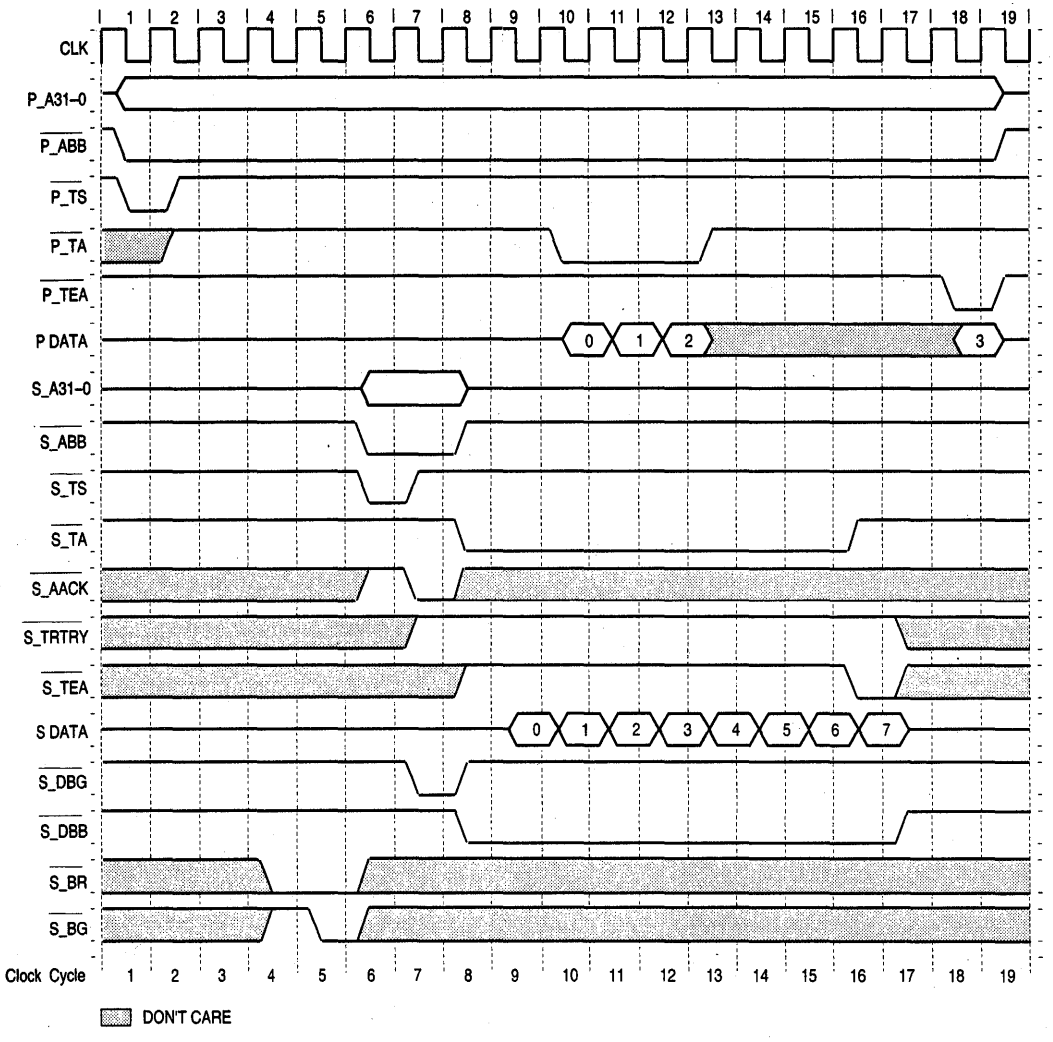


Figure 5-29. Transfer Error Termination During Streaming

The MC88410 receives $\overline{\text{S\_TEA}}$ asserted with the last beat of data in clock 16. Note that in the full-speed mode the last $\overline{\text{S\_TA}}$ is already asserted in clock 15. In clock 18, the MC88410 drives data beat 3 to the processor and asserts $\overline{\text{P\_TEA}}$.

5.7 SYSTEM BUS SNOOPING

The MC88410 uses a bus snooping protocol to monitor bus transactions performed by other system bus masters and to intervene in the access, when required, in order to maintain cache coherency. The MC88410 services system bus snoop transactions from the secondary cache without interaction with the processor unless there is a snoop hit in the primary data cache. The MC88410 contains a snoop latch that allows the MC88410 to save

a snoop lookup that could not be serviced immediately due to internal resource contention without resnooping the saved address. The snoop latch ensures coherency in split-bus systems when the address bus transaction has terminated but the data transaction has not completed.

The differences between the MC88410 snoop protocol and that of the MC88110 are the addition of the $\overline{\text{TSHD}}$ signal which enables shared information to arrive with the data and the addition of the $\overline{\text{S\_SSTAT2}}$ signal which indicates that a copyback transaction will occur. In the full-speed mode, $\overline{\text{S\_BR}}$ is asserted in the clock after the assertion of $\overline{\text{S\_SSTAT1}}$. In the half-speed mode, $\overline{\text{S\_BR}}$ is asserted in the same clock as $\overline{\text{S\_SSTAT1}}$.

The following paragraphs describe the operation of the bus when snooping is enabled. For more information about coherency issues related to the MC88410 and MC88110, refer to **Section 2 Secondary Cache Operation**.

Throughout this discussion of data cache coherency, the terms "initiating MC88410" and "snooping MC88410" are used. The initiating MC88410 is the MC88410 that is the bus master at the beginning of a bus transaction. The snooping MC88410 is the MC88410 that snoops this transaction.

5.7.1 Snoop Control Signal Overview

Table 5-8 lists the snoop control signals of the MC88410. The $\overline{\text{S\_SR}}$ signal is an input to all snooping MC88410s indicating that the current address should be latched because a snoop lookup may be required. The $\overline{\text{S\_SR}}$ signal may simply be connected to the $\overline{\text{S\_TS}}$ signal of the initiating MC88410. The $\overline{\text{S\_SR}}$ signal must be negated and reasserted between two accesses that need to be snooped or it is ignored on the second access. The $\overline{\text{S\_GBL}}$ signal is an output when the MC88410 is initiating a transaction, and an input when it is snooping. The MC88410 only snoops transactions when both the $\overline{\text{S\_SR}}$ and $\overline{\text{S\_GBL}}$ signals are asserted.

Table 5-8. Snoop Control Signal Summary

Signal	Function	Type
$\overline{\text{S\_SR}}$	System snoop request	Input
$\overline{\text{S\_ARTRY}}$	System address retry	Input
$\overline{\text{S\_SSTAT2}}$–$\overline{\text{S\_SSTAT0}}$	System snoop status	Output
$\overline{\text{SHD}}$	Shared	Input
$\overline{\text{TSHD}}$	Transfer shared	Input

The MC88410 needs to sample $\overline{\text{S\_SR}}$, $\overline{\text{S\_GBL}}$, S_R/$\overline{\text{W}}$, $\overline{\text{S\_TBST}}$, and $\overline{\text{S\_INV}}$ to snoop an access. The MC88410 does not require $\overline{\text{S\_ABB}}$ asserted, but if it is not asserted $\overline{\text{S\_SSTAT2}}$–$\overline{\text{S\_SSTAT0}}$ will assert for only one clock.

When the S_SR and S_GBL signals are both asserted, the MC88410 determines whether it has a cache hit in the primary and secondary cache (PTAG and MTAG, see **Section 2 Secondary Cache Operation**) or a collision (see **5.8.2 Split-Bus Snoop Collisions**). If there is a collision, the snooping MC88410 asserts S_SSTAT1 but does not assert S_SSTAT0. If the MC88410 performs a copyback transaction it asserts S_SSTAT2. If there is a cache hit, the snooping MC88410 takes the action described in Table 5-9.

Table 5-9. MC88410 Actions for Snoop Hits

S_SR	S_GBL	S_R/W	S_INV	S_TBST	Action on Snoop Hit
N	x	x	x	x	No action
A	N	x	x	x	No action
A	A	R	N	x	Assert S_SSTAT0; if line was modified, assert S_SSTAT1, assert S_SSTAT2, perform copyback, and mark line shared unmodified
A	A	R	A	x	Assert S_SSTAT0 and invalidate cache line; if line was modified, assert S_SSTAT1, assert S_SSTAT2, perform copyback, and invalidate cache line
A	A	W	A	N	Assert S_SSTAT0; if line was modified, assert S_SSTAT1, assert S_SSTAT2, perform copyback, and invalidate cache line
A	A	W	A	A	Assert S_SSTAT0 and invalidate cache line

A = Asserted
N = Negated
x = Don't Care

5.7.2 TSHD Timing

The TSHD signal is an addition to the MC88110 bus protocol. The TSHD input signal lets other devices on the system interface specify that the current secondary cache line should be marked shared unmodified. The memory system typically derives TSHD from the S_SSTAT0 signal. The MC88410 samples TSHD during data bus mastership, which allows the memory system to collect and pipeline snoop responses from distant snoopers that have a long response time.

The TSHD signal is sampled while S_DBB is asserted. The TSHD signal is ignored if the data tenure is for a transaction which is intent-to-modify. If the MC88410 detects the TSHD signal asserted during the first S_TA of a secondary cache line allocation, the cache line is placed into a shared unmodified state instead of an exclusive modified state. Systems without distant snoopers should leave TSHD negated and allow the state transition to be handled by the SHD signal.

Figure 5-30 shows an example of TSHD timing. In clock 1 MC88410-A initiates a global system bus transaction that is snooped by MC88410-B, which is slow to respond. MC88410-B has a snoop hit and asserts S_SSTAT0. The memory system asserts S_AACK in clock 2, which terminates the address bus tenure of MC88410-A. However, the system could not assert S_SSTAT0 to MC88410-A by the rising edge of clock 3. The TSHD signal

allows the fact that MC88410-B has asserted $\overline{\text{S\_SSTAT0}}$ to be communicated to MC88410-A during its data bus tenure. Later, in clock N, MC88410 receives a qualified data bus grant. In clock N+1 MC88410-A asserts $\overline{\text{S\_DBB}}$ and samples $\overline{\text{TSHD}}$. Since $\overline{\text{TSHD}}$ is asserted, the secondary cache line is placed in the shared state.

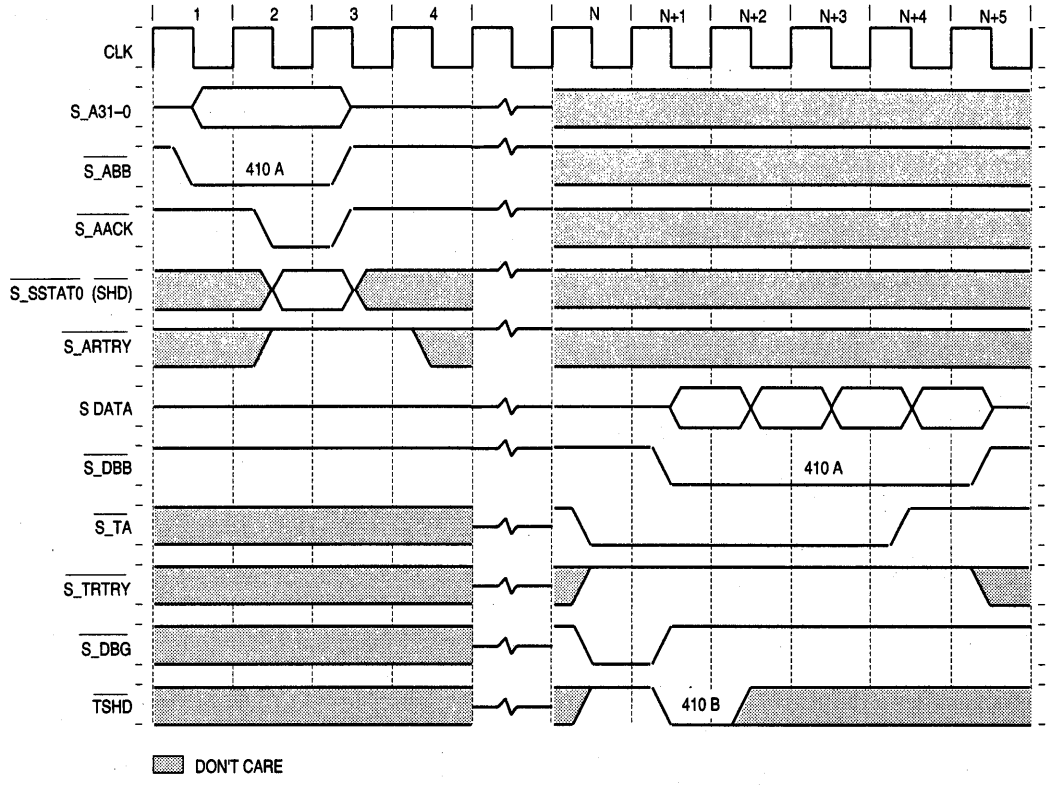


Figure 5-30. $\overline{\text{TSHD}}$ Timing

5

5.7.3 $\overline{\text{S\_SSTAT2}}$–$\overline{\text{S\_SSTAT0}}$ Timing

The MC88410 asserts the $\overline{\text{S\_SSTAT2}}$–$\overline{\text{S\_SSTAT0}}$ signals, if necessary, two system bus clock cycles after the assertion of the $\overline{\text{S\_SR}}$ and $\overline{\text{S\_GBL}}$ inputs. If a snoop copyback transaction must be performed, the MC88410 asserts bus request one clock cycle after the assertion of $\overline{\text{S\_SSTAT1}}$ and $\overline{\text{S\_SSTAT2}}$ in the full-speed mode. In the half-speed mode, $\overline{\text{S\_BR}}$ is asserted with $\overline{\text{S\_SSTAT1}}$ and $\overline{\text{S\_SSTAT2}}$. The $\overline{\text{S\_SSTAT2}}$–$\overline{\text{S\_SSTAT0}}$ signals remain valid until $\overline{\text{S\_ABB}}$ is negated. Note that if the initiating MC88410 is parked, $\overline{\text{S\_ABB}}$ is still negated for one clock between transactions.

Figure 5-31 shows the timing for the $\overline{\text{S\_SSTAT2}}$–$\overline{\text{S\_SSTAT0}}$ signals. The initiating MC88410 starts a global memory access in clock 1, as indicated by $\overline{\text{S\_SR}}$ and $\overline{\text{S\_GBL}}$ asserted. The snooping MC88410 latches the address and asserts the appropriate snoop status signals two clocks later (if necessary) in clock 3. If the snooping MC88410 detects a snoop hit to a

modified primary or secondary cache line, then the snooping MC88410 asserts its bus request one clock after S_SSTAT2–S_SSTAT0. The second transaction in Figure 5-31 shows an example when S_ABB stays asserted for several clock cycles after the snoop status signals are asserted.

The S_SSTAT2, S_SSTAT1, and S_SSTAT0 outputs of the MC88410 and another MC88410 can each be connected without contention. These signals must be connected to pull-up resistors to keep them negated when no processor is driving them. Each time one of the snoop status signals is asserted, the MC88410 negates it before three-stating it. The snoop status signals must be negated in a unique way to avoid contention problems during the transition.

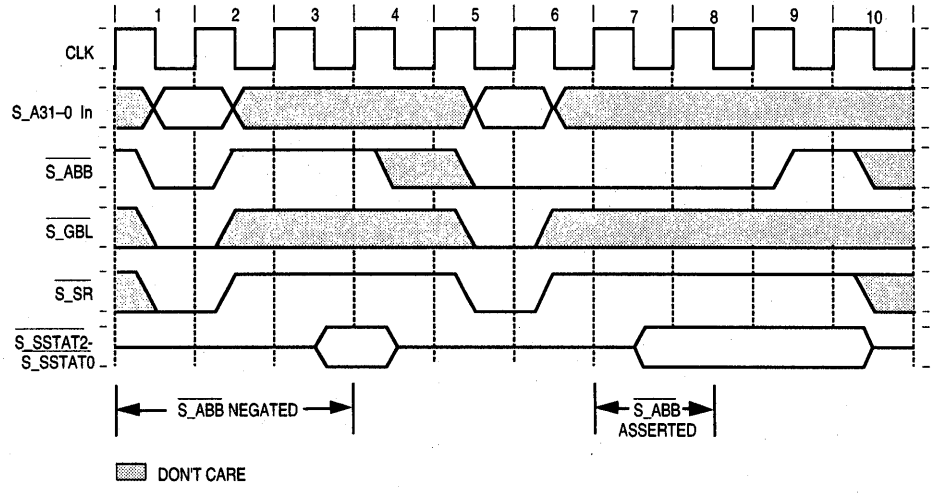


Figure 5-31. Snoop Hit/Miss Indication (S_SSTAT2 - S_SSTAT0)

Figure 5-32 shows two MC88410s driving the S_SSTAT0 signals at the same time (labeled S_SSTAT0-A and SSTAT0-B in the diagram). The two S_SSTAT0 signals are connected together and connected to Vdd through a pull-up resistor. The combined signal is called S_SSTAT0. In clock 1, a third MC88410 starts a global transaction. Note that both S_SSTAT0-A and S_SSTAT0-B are three-stated because neither MC88410 is driving the signal, but S_SSTAT0 is negated because of the pull-up resistor. Two clocks later, both MC88410-A and MC88410-B have a cache hit and assert S_SSTAT0-A and S_SSTAT0-B, respectively. When S_ABB is negated, the MC88410s must negate S_SSTAT0 to prepare for the next snoop cycle. However, if both MC88410s transition from driving the signals low to driving them high, there is the possibility for contention during the transition. Therefore, S_SSTAT0-A and S_SSTAT0-B are each three-stated, then negated, and then three-stated again.

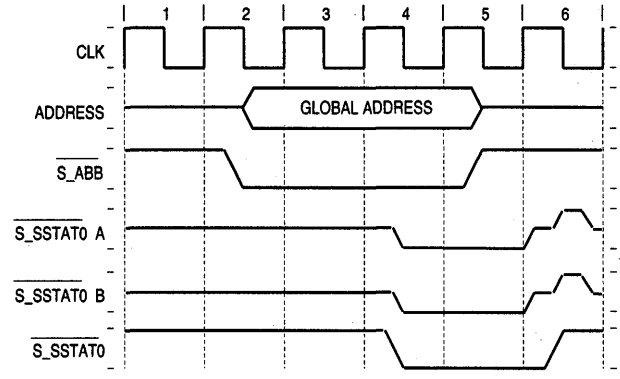


Figure 5-32. Snoop Status Negation Timing

5.7.4 Bus Request Blocking

The MC88410 implements a blocking protocol that allows a snooping device an opportunity to acquire mastership of the address bus. When an MC88410 that is requesting the bus detects that $\overline{\text{S\_ARTRY}}$ is asserted and that $\overline{\text{S\_ABB}}$ was asserted on the previous clock cycle, it removes its bus request and ignores any bus grant. The MC88410 then blocks its bus requests by not asserting $\overline{\text{S\_BR}}$ until one clock after $\overline{\text{S\_ABB}}$ is negated or until $\overline{\text{S\_ARTRY}}$ is negated. Note that the MC88410 does not block $\overline{\text{S\_BR}}$ due to $\overline{\text{S\_ARTRY}}$ if $\overline{\text{S\_ABB}}$ was negated on the previous clock cycle.

Figure 5-33 shows the MC88410 bus request blocking protocol. In clock 1, $\overline{\text{S\_ABB}}$ and $\overline{\text{S\_ARTRY}}$ are both asserted. In clock 2, the MC88410 does not block its bus request, because $\overline{\text{S\_ABB}}$ was negated on the previous clock cycle. In clock 3, however, $\overline{\text{S\_ARTRY}}$ is asserted and $\overline{\text{S\_ABB}}$ was asserted on the previous clock, so the MC88410 negates $\overline{\text{S\_BR}}$ (if it was asserted). The MC88410 continues to block its bus request until it recognizes in clock N that $\overline{\text{S\_ARTRY}}$ is negated.

Figure 5-34 shows an example of the $\overline{\text{S\_BR}}$ blocking protocol assuming a system with two MC88410s and with $\overline{\text{S\_SSTAT1}}$ connected to $\overline{\text{S\_ARTRY}}$. In clock one, MC88410-A has a transaction in progress that causes a snoop hit in MC88410-B. MC88410-B asserts $\overline{\text{S\_SSTAT1}}$ (which is connected to $\overline{\text{S\_ARTRY}}$) in clock 1 and $\overline{\text{S\_BR}}$ in clock 2. Also in clock 2, MC88410-A recognizes that $\overline{\text{S\_ARTRY}}$ is asserted and negates $\overline{\text{S\_BR}}$. Note, however, that $\overline{\text{S\_ARTRY}}$ is not qualified ($\overline{\text{S\_AACK}}$ negated) in clock 2, so MC88410-A maintains control of the address bus. In clock 3, MC88410-A recognizes a qualified $\overline{\text{S\_ARTRY}}$ and negates $\overline{\text{S\_ABB}}$. In the same clock, the arbiter recognizes that MC88410-B is the only device requesting the bus (all other devices are blocking their bus requests) and asserts bus grant to MC88410-B. In clock 4, MC88410-B receives a qualified bus grant, so it asserts $\overline{\text{S\_ABB}}$ and negates $\overline{\text{S\_BR}}$.

The negation of $\overline{\text{S\_ABB}}$ in clock 3 causes MC88410-B to negate $\overline{\text{S\_SSTAT1}}$ in clock 4. Because of $\overline{\text{S\_SSTAT1}}$ negation timing and potential propagation delays, $\overline{\text{S\_ARTRY}}$ may not be negated in time to be recognized in the clock following $\overline{\text{S\_SSTAT1}}$ negation. Therefore,

S_ARTRY must be ignored for bus request blocking purposes in any clock cycle following S_ABB negated.

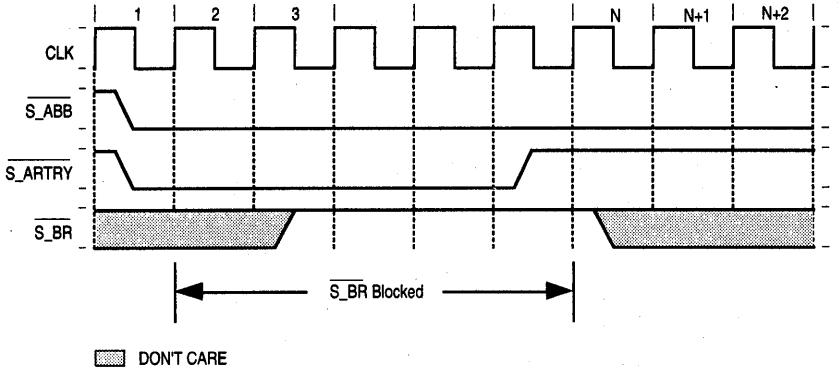


Figure 5-33. S_BR Blocking Protocol

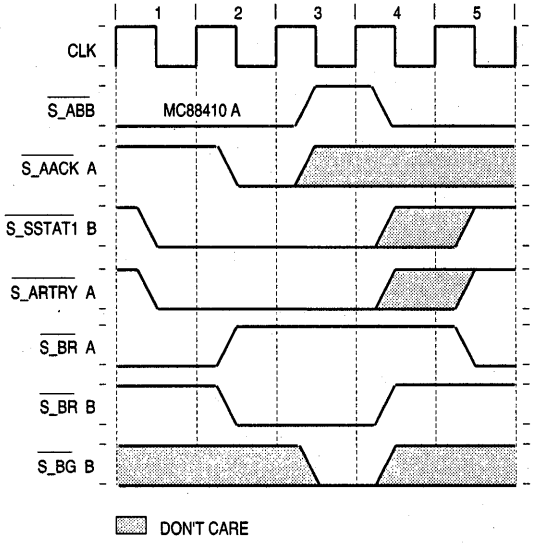


Figure 5-34. S_BR Blocking In a Dual MC88410 System

Note that the memory system can control the length of time that the bus requests are blocked by controlling when S_ARTRY is negated. Assuming that the S_ARTRY signal is controlled by the S_SSTAT1 signal of the snooping MC88410, the memory system can control when S_ARTRY is negated via the S_AACK signal. This is because the S_SSTAT1/S_ARTRY signal remains asserted as long as S_ABB is asserted, and the initiating MC88410 keeps S_ABB asserted until the S_AACK signal is asserted (or the transaction is terminated).

5.7.5 Snoop Miss Timing

When the MC88410 is snooping, it takes two clock cycles from the assertion of $\overline{\text{S\_SR}}$ to assert the snoop status signals; therefore, if the MC88410 is initiating a transaction that another MC88410 will be snooping, there must be a minimum of one wait state inserted into the transaction to allow for the snooping MC88410 to assert the snoop status signals. This can be done by delaying $\overline{\text{S\_TA}}$ or $\overline{\text{S\_DBG}}$ by at least one clock. Otherwise, data may be transferred to the secondary cache before the snoop status is known.

Figure 5-35 illustrates snoop transactions from the perspective of the initiating MC88410. Note that the first transaction begins while the initiating MC88410 is parked on the address bus. In clock cycle 1, the first transaction begins, but $\overline{\text{S\_GBL}}$ is negated so this transaction is not snooped. The first transaction is terminated with the $\overline{\text{S\_TA}}$ in clock 2 and a new one begins in clock 10.

In the second transaction, $\overline{\text{S\_GBL}}$ is asserted, so snooping occurs and a wait state must be inserted to allow the snooping bus masters to assert $\overline{\text{S\_SSTAT2}}$–$\overline{\text{S\_SSTAT0}}$. Since this transaction is a snoop miss, $\overline{\text{S\_SSTAT2}}$–$\overline{\text{S\_SSTAT0}}$ are negated in clock cycle 12. In the second transaction the address bus is split from the data bus, and address bus tenure ends with the assertion of $\overline{\text{S\_AACK}}$ in clock 11. Also during clock 11, the $\overline{\text{S\_ARTRY}}$ signal is negated, so the second transaction is terminated normally with the $\overline{\text{S\_TA}}$ on the rising edge of clock 16.

5.7.6 Secondary Cache Copyback Timing

If a snoop transaction hits in the MTAG of the MC88410 and the secondary cache line is modified, a snoop copyback transaction occurs. The following paragraphs describe the timing for various snoop copyback scenarios.

5.7.6.1 Full-Speed Snoop Hit and Copyback (No Split Bus)

Figure 5-36 shows an example of a snoop hit in full-speed mode without a split bus. MC88410-A begins the transaction with a global single-beat read which hits in the main tag of MC88410-B. In response to the snoop hit, MC88410-B asserts $\overline{\text{S\_SSTAT0}}$ (which is connected to the $\overline{\text{SHD}}$ signal) to indicate that the data is shared. Later, MC88410-B attempts a burst read transaction for different cache line and hits in MC88410-A. MC88410-A retries the transaction and asserts $\overline{\text{S\_SSTAT2}}$–$\overline{\text{S\_SSTAT0}}$ to indicate that a copyback transaction will occur and copies back the modified secondary cache line. Following the copyback transaction, MC88410-B initiates and completes its burst read transaction.

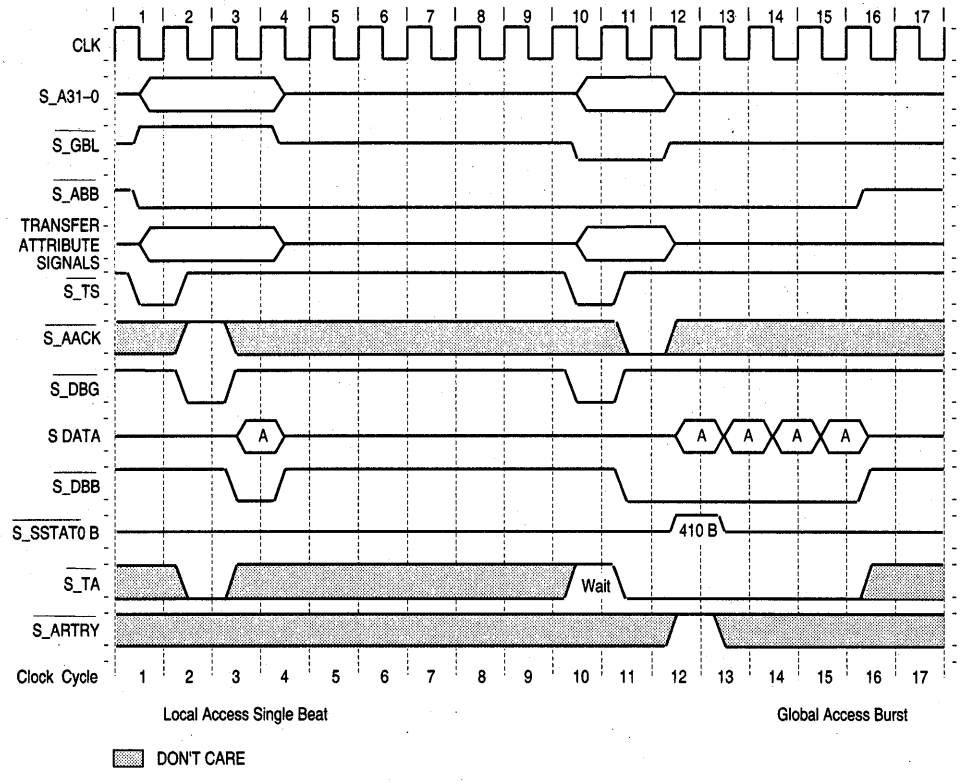


Figure 5-35. Full-Speed Snoop Miss Transactions

During clock 1, MC88410-A recognizes a qualified bus grant and asserts $\overline{\text{S\_TS}}$, the transfer attribute signals, and $\overline{\text{S\_ABB}}$. The assertion of $\overline{\text{S\_TS}}$ (which is connected to $\overline{\text{S\_SR}}$ of other masters) and $\overline{\text{S\_GBL}}$ causes other bus masters to snoop the transaction. On the rising edge of clock 2, the MC88410 recognizes a qualified data bus grant and asserts $\overline{\text{S\_DBB}}$. In this transaction, $\overline{\text{S\_DBG}}$ is asserted during the clock of $\overline{\text{S\_TS}}$. In clock 1 the memory system negates $\overline{\text{S\_TA}}$ for one clock to allow enough time for snooping devices to respond. The memory system asserts $\overline{\text{S\_TA}}$ in clock 2, and MC88410-A drives the beat of valid data in the next clock. In clock 3, MC88410-B asserts $\overline{\text{S\_SSTAT0}}$ to indicate the data is shared.

In clock 6, MC88410-B requests the system bus for a global burst read for a different line, and initiates the transaction in clock 8. In this transaction, $\overline{\text{S\_DBG}}$ is asserted during the clock $\overline{\text{S\_TS}}$ is asserted. The $\overline{\text{S\_TA}}$ signal is negated in clock 8 to insert a wait state to allow snooping devices to respond and then asserted in clock 9 to terminate the transaction, illustrating the minimum bus tenure for snooping. MC88410-A recognizes a snoop hit to a modified line and asserts $\overline{\text{S\_SSTAT2}}$, $\overline{\text{S\_SSTAT1}}$, and $\overline{\text{S\_SSTAT0}}$ in clock 10. The $\overline{\text{S\_ARTRY}}$ input signal is connected to the $\overline{\text{S\_SSTAT1}}$ output signal, and is also asserted in clock 10. MC88410-A asserts $\overline{\text{S\_BR}}$ in clock 11 to request the bus for its copyback transaction. Since this example does not use a split bus, $\overline{\text{S\_AACK}}$ remains negated and the assertion of $\overline{\text{S\_ARTRY}}$ is not recognized by MC88410-B until the rising edge of clock 11. During clock 11, it relinquishes its mastership of the address bus.

MC88410-A recognizes its qualified bus grant during the rising edge of clock 13 and asserts $\overline{\text{S\_TS}}$ and its transfer attribute signals. Note that $\overline{\text{S\_GBL}}$ is negated for the copyback transaction and a wait state is not inserted before asserting $\overline{\text{S\_TA}}$. However, in this example, the external arbiter does not assert $\overline{\text{S\_DBG}}$ until clock 14, causing MC88410-A to wait. MC88410-A copies back its secondary cache line during clocks 16 through 19 and terminates its transaction.

During clock 13, MC88410-B reasserts its bus request to retry its burst read and it receives a qualified bus grant on the rising edge of clock 21. MC88410-B initiates its data transaction in clock 22 and completes it in clock 27.

5.7.6.2 Full-Speed Snoop Hit and Copyback (Split Bus)

Figure 5-37 shows an example similar to Figure 5-36 but with a split-bus protocol. In this example, MC88410-A attempts a burst read transaction that hits in the main cache tag of MC88410-B. The memory system asserts the $\overline{\text{S\_AACK}}$ signal to terminate the transaction on the address bus. This allows the address and data bus to be fully decoupled.

MC88410-B initiates the transaction by asserting $\overline{\text{S\_TS}}$ in clock 3 and receives a qualified data bus grant. The $\overline{\text{S\_TA}}$ signal is negated by the memory system in clock 3 to insert a wait state for snooping. The memory system asserts $\overline{\text{S\_AACK}}$ in clock 4 to terminate address bus tenure. During clock 5, MC88410-A three-states the address and control signals. At the same time, MC88410-B retries the transaction by asserting $\overline{\text{S\_SSTAT1}}$ ($\overline{\text{S\_ARTRY}}$), which is sampled in the clock after $\overline{\text{S\_AACK}}$ is asserted. MC88410-B negates $\overline{\text{S\_SSTAT1}}$ in clock 6 since $\overline{\text{S\_ABB}}$ is no longer asserted.

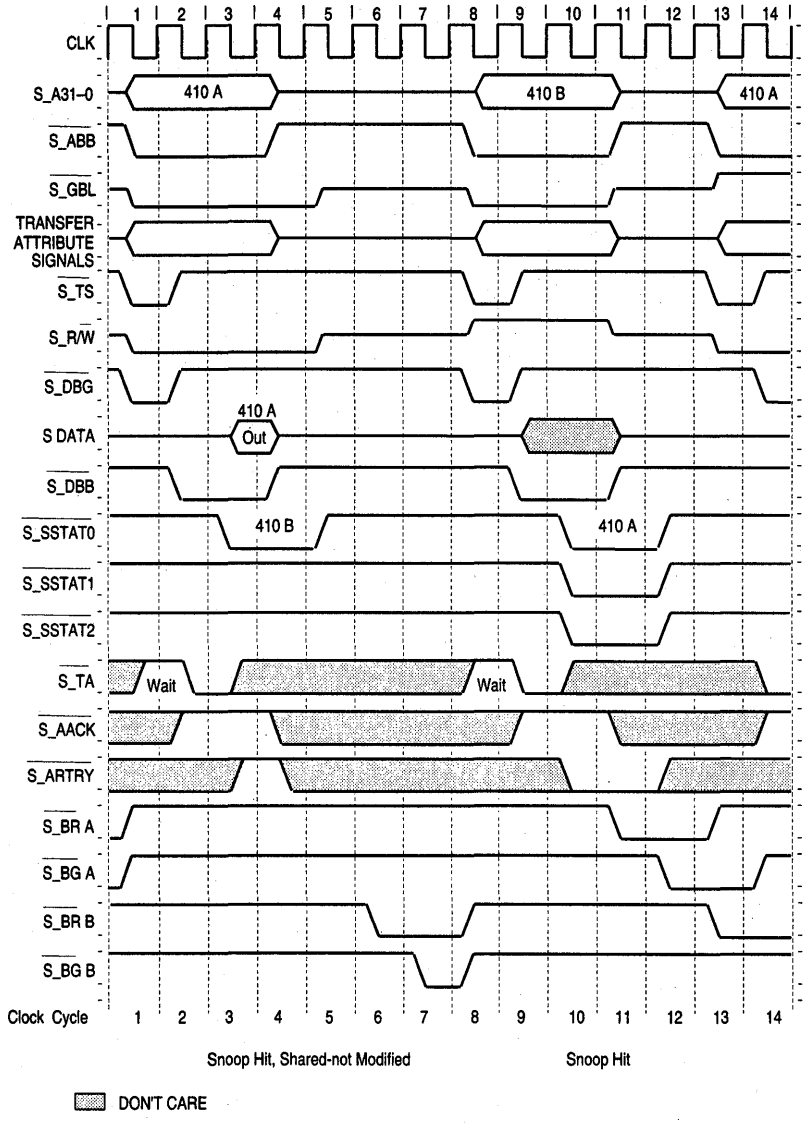


Figure 5-36a. Full-Speed Snoop Hit and Copyback (No Split Bus)

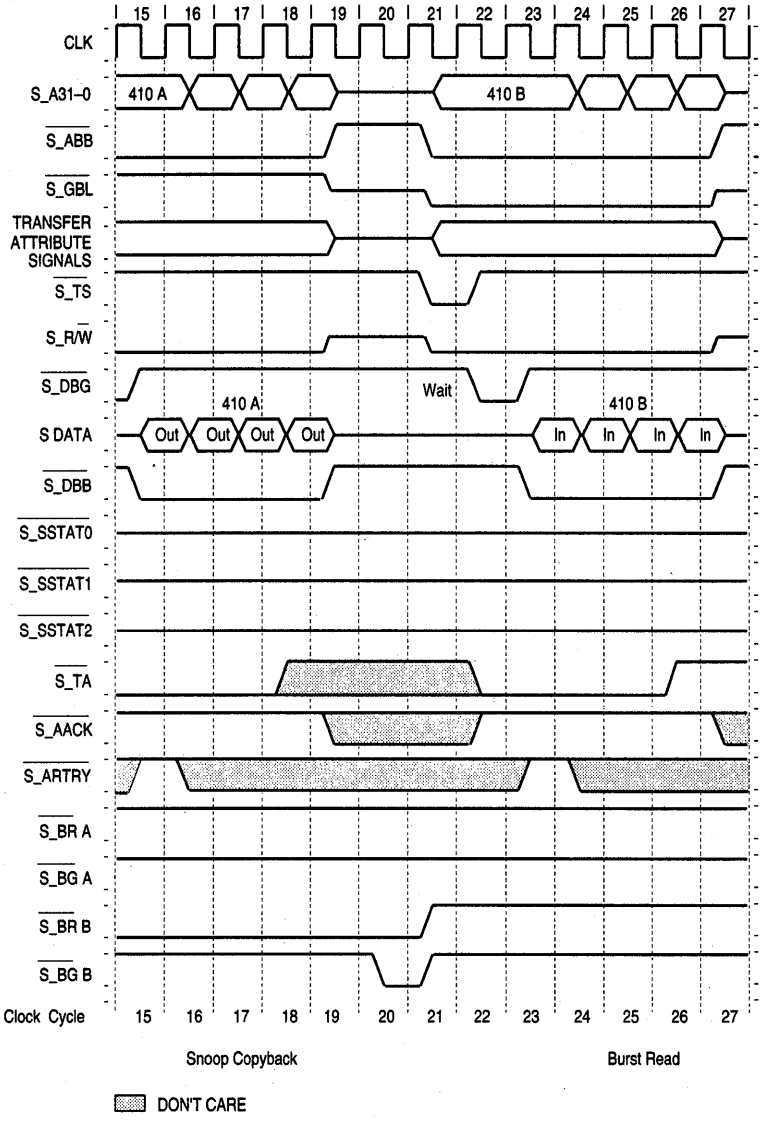


Figure 5-36b. Full-Speed Snoop Hit and Copyback (No Split Bus)

MC88410-B requests the bus in clock 6 and detects a qualified bus grant on the rising edge of clock 9. MC88410-B then asserts $\overline{\text{S\_TS}}$ and takes control of the address bus. The assertion of $\overline{\text{S\_AACK}}$ in clock 10 terminates the address bus tenure of MC88410-B. The negation of $\overline{\text{S\_ABB}}$ and the assertion of $\overline{\text{S\_BG}}$ by the arbiter in clock 11 gives MC88410-A mastership of the address bus on the rising edge of clock 12, even though MC88410-B is still involved in its data transfer. MC88410-A detects $\overline{\text{S\_AACK}}$ asserted on the rising edge of clock 14 and terminates its address bus mastership as MC88410-B completes its copyback. MC88410-A initiates its data bus transaction in clock 16 and completes it in clock 20.

5.7.6.3 Half-Speed Snoop Hit and Copyback (Split Bus)

Figure 5-38 shows a snoop copyback transaction with the half-speed mode. MC88410-A initiates a transaction in clock 1 which is snooped by MC88410-B. MC88410-B asserts $\overline{\text{S\_SSTAT1}}$ in system bus clock 3 to indicate a snoop hit and retry the transaction in order to perform a snoop copyback.

Note that in the half-speed mode, $\overline{\text{S\_BR}}$ is asserted in the same clock as $\overline{\text{S\_SSTAT1}}$. The MC88410 recognizes a qualified bus grant on the rising edge of system bus clock 4 and takes mastership of the address bus. The MC88410 terminates address bus tenure in system bus clock 6 as a result of the assertion of $\overline{\text{S\_AACK}}$. MC88410-B is granted the data bus in system bus clock 6 and begins its snoop copyback transaction in system bus clock 7 and completes it in clock 11.

5.7.7 Snoop Hit with Primary Cache Invalidate

If a system bus transaction causes a snoop hit in the processor tag (PTAG), the data is included in the primary cache. The MC88410 must issue a primary cache invalidate transaction to maintain coherency. For more information about primary cache invalidate transactions, refer to **Section 4 Processor Bus Interface**.

The MC88410 may detect a snoop hit while it is arbitrating for the system bus interface to complete a processor transaction. If the snoop hit requires copyback or a processor invalidate transaction, the processor transaction is interrupted with a retry indication. The interrupted processor transaction is restarted when the copyback transaction is completed. However, a snoop copyback or processor invalidate transaction cannot interrupt system bus snoop activity already in progress and will be retried.

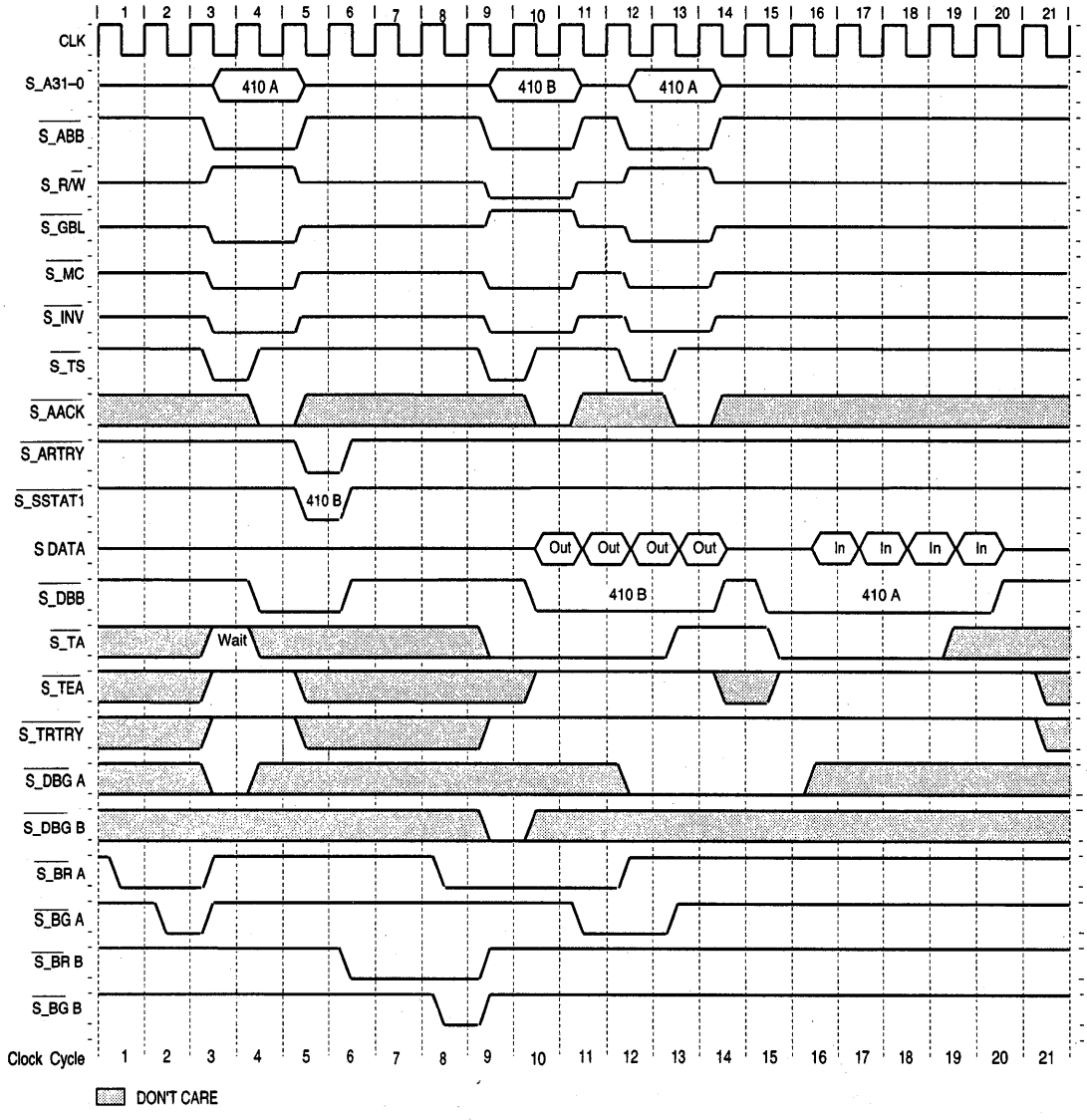


Figure 5-37. Full-Speed Snoop Hit and Copyback (Split Bus)

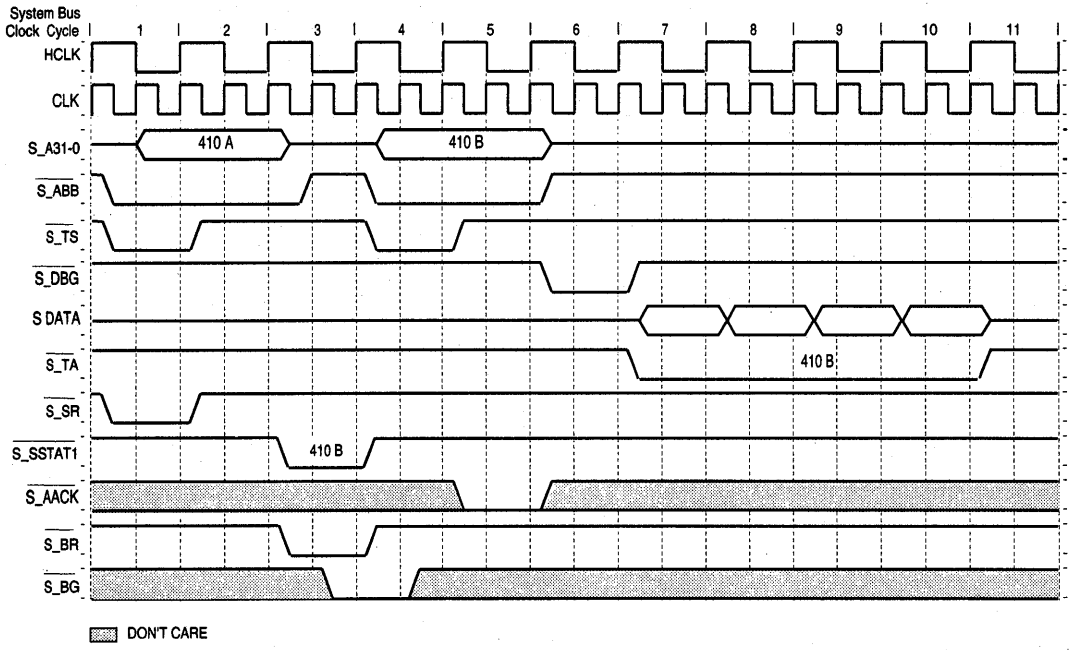


Figure 5-38. Half-Speed Snoop Copyback

Figure 5-39 shows an example of a snoop hit causing a primary cache invalidate transaction. MC88410-A initiates a global burst read-with-intent-to-modify, which hits in MC88410-C and causes a primary cache invalidate transaction to MC88110-C. During the invalidate transaction on the processor bus, MC88410-C is able to snoop a global burst read on the system bus by MC88410-B that misses in MC88410-C. In this example the processor does not have an outstanding request to MC88410-C.

5

Assuming typical bus arbitration, MC88410-A asserts $\overline{\text{S\_TS}}$ in clock 2 to initiate a global burst read which is intent-to-modify. In this example, $\overline{\text{S\_DBG}}$ is negated in clock 2 in order to insert a wait state for snooping. The assertion of $\overline{\text{S\_AACK}}$ by the memory system in clock 3 terminates the address bus tenure of MC88410-A. In the following clock, MC88410-C asserts $\overline{\text{S\_SSTAT0}}$, indicating that the data is shared. Between clocks 2 and 4, MC88410-C completed its tag lookup and determined that the shared data was not modified but was included in the primary cache (PTAG hit). Therefore, in clock 5, the MC88410-C requests the processor bus from the external arbiter in order to perform a primary cache invalidate transaction.

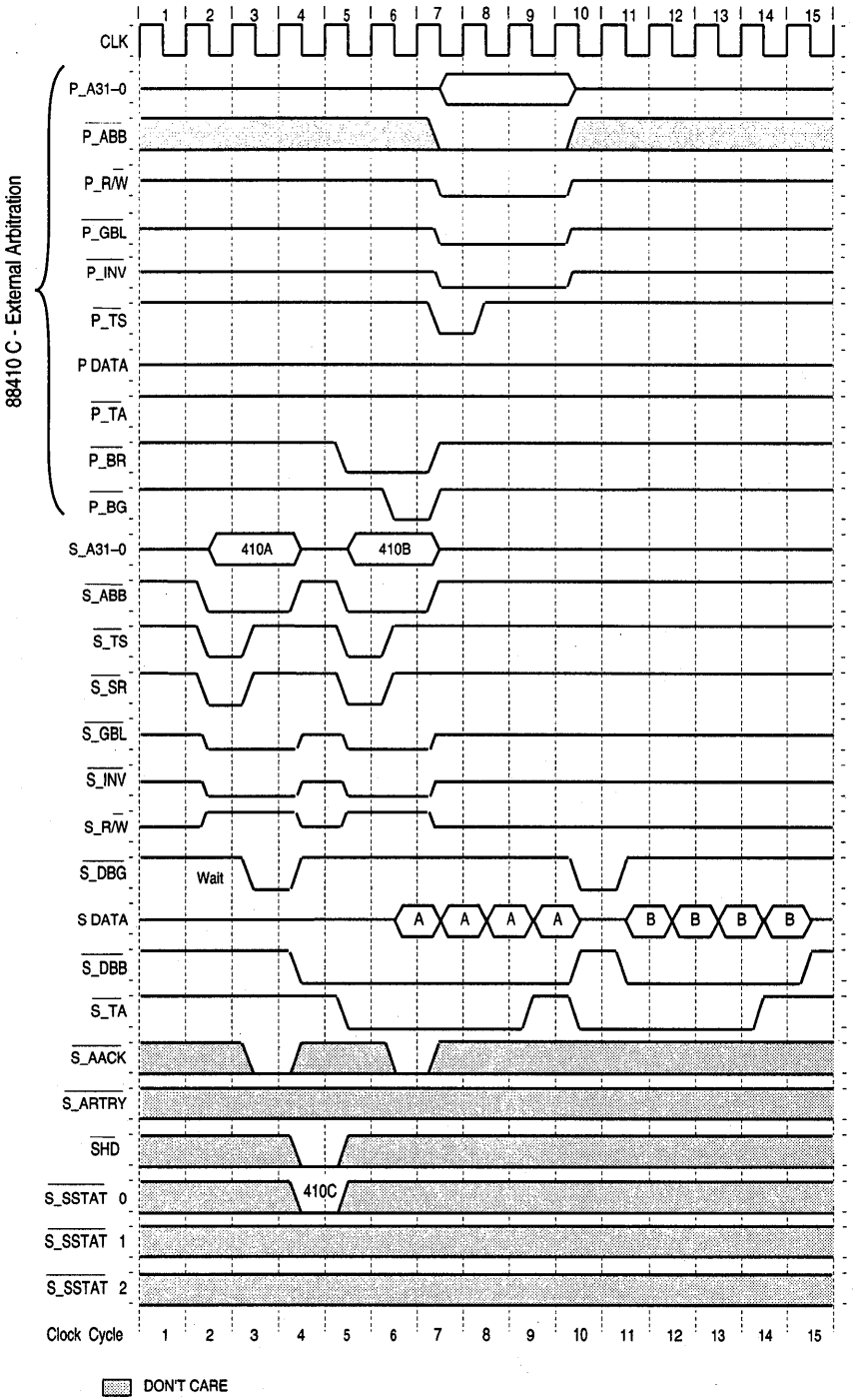


Figure 5-39. Snoop Hit with Processor Invalidation Broadcast (Split Bus)

In clock 5, MC88410-B asserts $\overline{\text{S\_TS}}$ to initiate a global burst read which is also intent-to-modify. Also in clock 5, the memory system asserts $\overline{\text{S\_TA}}$ to MC88410-A, which had received a qualified data bus grant on the rising edge of clock 4. On the rising edge of clock 7 the following occurs: the processor bus interface of MC88410-C detects a qualified processor bus grant from the external arbiter, MC88410-B detects the assertion of $\overline{\text{S\_AACK}}$ by the memory system, and the first beat of data is transferred to the secondary cache of MC88410-A.

During clock 7, MC88410-C performs the primary cache invalidate transaction by asserting $\overline{\text{P\_TS}}$, the transfer attribute signals, and driving the address onto the processor bus. At the same time the MC88410-C has completed its tag lookup of the burst read of MC88410-B and determined that it has missed in the MTAG and PTAG. The processor interface of MC88410-C maintains address bus tenure for two clocks to allow MC88110-C time to respond to the snoop hit. Note that even though the system interface has responded to the original snoop by not asserting $\overline{\text{S\_SSTAT1}}$ (indicating that a processor copyback was not necessary), the processor interface still waits until clock 10 before relinquishing the processor bus in order to detect a processor copyback. On the system bus, MC88410-B relinquishes the address bus during clock 7.

During clock 10, MC88410-C completes its primary cache invalidate transaction, relinquishing the processor bus, and MC88410-A completes its burst read and relinquishes the system bus. MC88410-B detects a qualified data bus grant on the rising edge of clock 11 and performs its burst read during clocks 12 to 15.

Figure 5-40 shows an example of a full-speed primary cache invalidate transaction that interrupts a processor transaction. In clock 1 MC88110-B initiates a replacement copyback transaction. Also during clock 1, MC88410-A requests the system bus and initiates a global single-beat write transaction. The memory system negates $\overline{\text{S\_TA}}$ in clock 2 to insert a wait state to allow MC88410-B to snoop the transaction. An additional wait is inserted by the external arbiter negating $\overline{\text{S\_DBG}}$ until clock 3. The assertion of $\overline{\text{S\_AACK}}$ (not shown) in clock 3 terminates MC88410-A address bus mastership. MC88410-B asserts $\overline{\text{S\_SSTAT0}}$ in clock 4 as a result of the snoop hit on the read of MC88410-A.

Since the snoop hit in the PTAG, MC88410-B must perform a primary cache invalidate transaction so that MC88110-B will invalidate the cache line. The processor is transferring the third beat of the replacement copyback to the secondary cache when MC88410-B asserts $\overline{\text{P\_TRTRY}}$ to retry the transaction. MC88410-B becomes processor bus master by negating $\overline{\text{P\_BG}}$ and begins the primary cache invalidate transaction in clock 7. Since the MC88110-B data cache line is not modified, a copyback transaction is not needed and the processor reinitiates the replacement copyback in clock 11.

While the processor interface completes the primary cache invalidate transaction, the system bus interface services a snoop from a burst transaction by MC88410-C. MC88410-C initiates the burst transaction in clock 6 which misses in the secondary cache of MC88410-B. As a result, $\overline{\text{S\_SSTAT0}}$ remains negated in clock 8 (when it would have been asserted for a snoop hit). MC88410-C performs its burst transaction on the data bus in clocks 10 through 13.

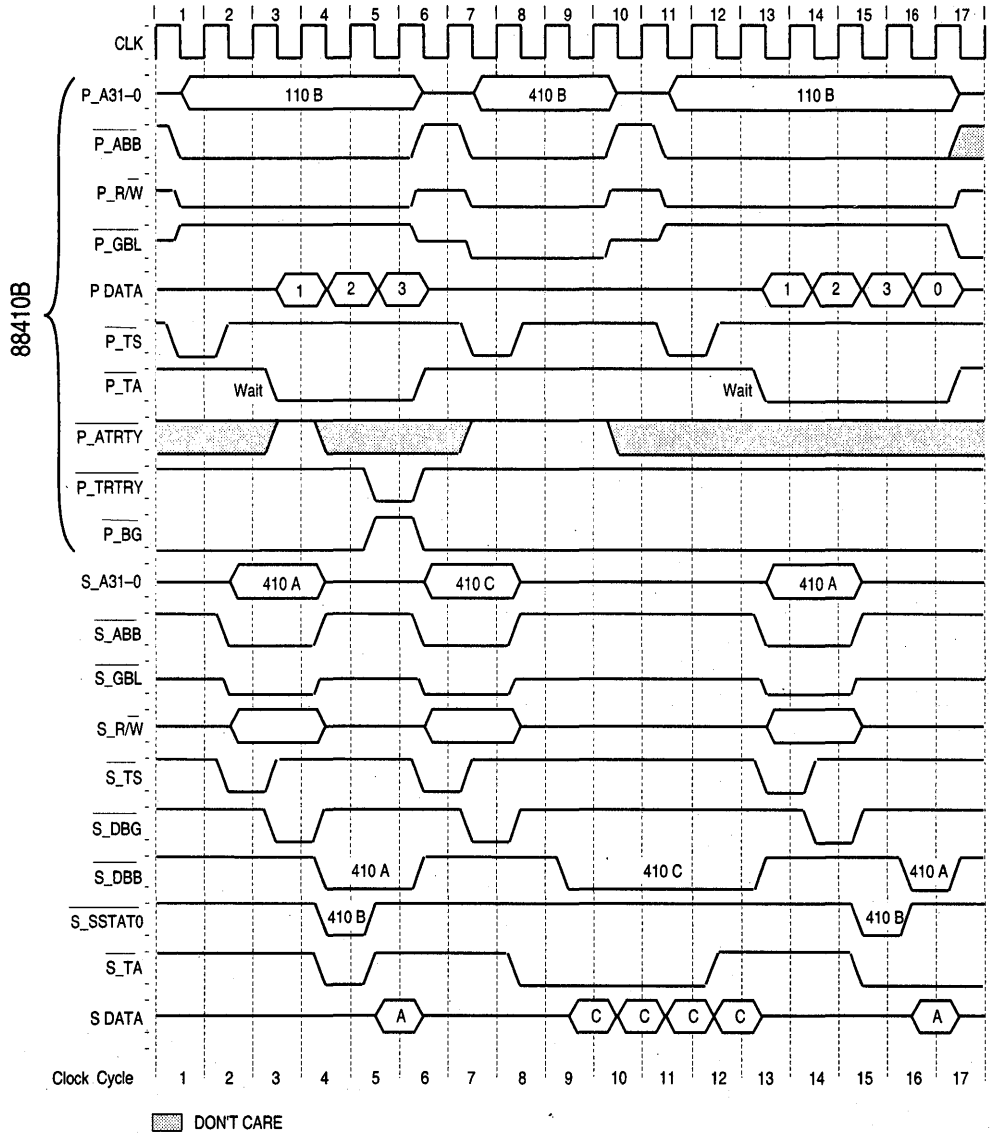


Figure 5-40. Snoop Hit which Interrupts Processor Transaction

In clock 11 MC88110-B reinitiates its replacement copyback transaction to the secondary cache. During the retried replacement copyback transaction of MC88110-B, MC88410-B responds to a snoop request from MC88410-A that hits in the secondary cache by asserting $\overline{\text{S\_SSTAT0}}$ in clock 15. Since the line is not modified, no further action by MC88410-B is required. This example demonstrates the ability of the MC88410 to respond to system bus snoop requests without interrupting the processor transaction unless it is necessary to maintain cache coherency.

5.7.8 Snoop Hit with Processor Copyback Timing

Figure 5-41 shows an example of a system bus snoop hit with a primary cache invalidate transaction that causes a processor copyback transaction. In this case, the processor copies back the data to the secondary cache, and the MC88410 copies back the secondary cache line to main memory. In this example, the secondary cache is configured with a zero-word-first ordering and a 64-byte secondary cache line. This requires two primary cache invalidate transactions if both halves (32-bytes) of the secondary cache line are included in the primary data cache (PTAG hit). In this example, both lines are cached in the primary data cache but only one has been modified, requiring copyback of that primary data cache line. Note that the processor bus transactions are critical-word-first but the secondary cache copyback is zero-word-first.

In this example, MC88410-A requests the system bus and has received a qualified system bus grant to perform a global cache-inhibited, single-beat read transaction (not shown). In clock 1 MC88410-A initiates the single-beat read on the system bus. The external arbiter negates $\overline{\text{S\_DBG}}$ in clock 1 to insert a wait state for snooping. During clock 2, $\overline{\text{S\_TA}}$ is negated by the memory system, inserting another wait state (note that it could have been asserted during clock 2), and MC88410-A detects $\overline{\text{S\_AACK}}$ asserted, ending address bus tenure (for a split bus). This arbitration sequence is repeated for subsequent attempts by MC88410-A to complete its transaction.

MC88410-B detects a snoop hit in both the MTAG and the PTAG and asserts $\overline{\text{S\_SSTAT0}}$, $\overline{\text{S\_SSTAT1}}$, and $\overline{\text{S\_SSTAT2}}$ to indicate a snoop hit with a copyback. Assuming that $\overline{\text{S\_SSTAT1}}$ of MC88410-B is connected to the $\overline{\text{S\_ARTRY}}$ input of other bus masters, MC88410-A detects the retry on the rising edge of clock 4. Since $\overline{\text{S\_ABB}}$ was asserted on the rising edge of clock 3 and $\overline{\text{S\_ARTRY}}$ is asserted on the rising edge of clock 4, the bus request of MC88410-A is blocked during clock 4. In clock 5, MC88410-A asserts $\overline{\text{S\_BR}}$ and receives a qualified bus grant in clock 6.

MC88410-A continues to reinitiate its single-beat read transaction and MC88410-B continues to retry the transaction until MC88410-B has completed its secondary cache copyback. Note that in Figure 5-41, $\overline{\text{S\_BG}}$ is shown asserted in the same clock as $\overline{\text{S\_BR}}$ in subsequent transaction retries by MC88410-A for convienence. If $\overline{\text{S\_BG}}$ is asserted in the clock following $\overline{\text{S\_BR}}$ (as in the first transaction) for subsequent retries, MC88410-B would detect its qualified bus grant on the rising edge of clock 30. An intelligent arbiter could use the assertion of the $\overline{\text{S\_SSTAT2}}$ (which indicates that a copyback will occur) to not grant $\overline{\text{S\_BG}}$ to MC88410-A until the system bus snoop copyback of MC88410-B completes. This could improve system bus bandwidth if there are additional system bus masters.

As a result of the PTAG hit, MC88410-B initiates a primary cache invalidate transaction to MC88110-B. On the processor bus, MC88410-B negates $\overline{\text{P\_BG}}$, and since $\overline{\text{S\_ABB}}$ is negated, takes mastership of the processor bus in clock 7. If MC88110-B had been involved in a transaction, MC88410-B would have asserted $\overline{\text{P\_TRTRY}}$ to terminate the processor transaction and take mastership of the processor bus (for more information about $\overline{\text{P\_TRTRY}}$ and primary cache invalidate transaction, refer to **Section 4 Processor Bus Interface**). The primary cache invalidate transaction is driven for two clocks in order to be snooped by MC88110-B.

The snoop hits a modified line in the processor data cache and so in clock 9, MC88110-B asserts $\overline{\text{SSTATI}}$ (which is connected to the $\overline{\text{P\_ARTRY}}$ input signal of MC88410-B) to indicate the snoop hit. MC88110-B begins its snoop copyback transaction in clock 11 by asserting $\overline{\text{P\_TS}}$. MC88410-B negates $\overline{\text{P\_TA}}$ in clock 12 to allow time to drive the address of the cache line to the secondary cache. The data is written into the first half of the secondary cache line in critical-word-first order and the processor copyback terminates in clock 17.

In clock 19, MC88410-B negates $\overline{\text{P\_BG}}$ to take mastership of the processor bus for the second primary cache invalidate. MC88110-B invalidates its primary cache line and since its data is unmodified, no copyback occurs.

While MC88410-B services MC88110-B on the processor bus, it also retries MC88410-A's repeated attempts to perform the single-beat read on the system bus. In clock 25 MC88410-B requests the system bus for the snoop copyback of the secondary cache line. A wait state is not needed because a copyback transaction always negates $\overline{\text{S\_GBL}}$. The address bus tenure of MC88410-B terminates with the assertion of $\overline{\text{S\_AACK}}$ on the rising edge of clock 29. On the data bus, MC88410-B writes eight beats of data (64-byte line size) out of the secondary cache in zero-word-first order (in clocks 29 through 37) and completes its copyback transaction. MC88410-A becomes the address bus master in clock 38. Finally, MC88410-A receives its single-beat of data during clock 39.

5.7.9 System DMA Invalidate

The MC88410 has a feature that allows external devices to invalidate the MC88410/MCM62110 secondary cache without causing a copyback transaction. If the MC88410 has a snoop hit during a global burst write, it invalidates the cache line without copying the line back (note that $\overline{\text{S\_INV}}$ must be asserted). The MC88410 and MC88110 never perform global burst write transactions. If a global burst write is detected, it must have been generated by an external device (for example, a DMA controller) that is overwriting some portion of memory, thus there is no reason to copy back the line before invalidating. If data is included in the primary data cache, the MC88410 issues a primary DMA invalidate transaction to the processor by asserting $\overline{\text{P\_TBST}}$ during the primary cache invalidate transaction. This causes the MC88110 to invalidate the cached data line without copyback.

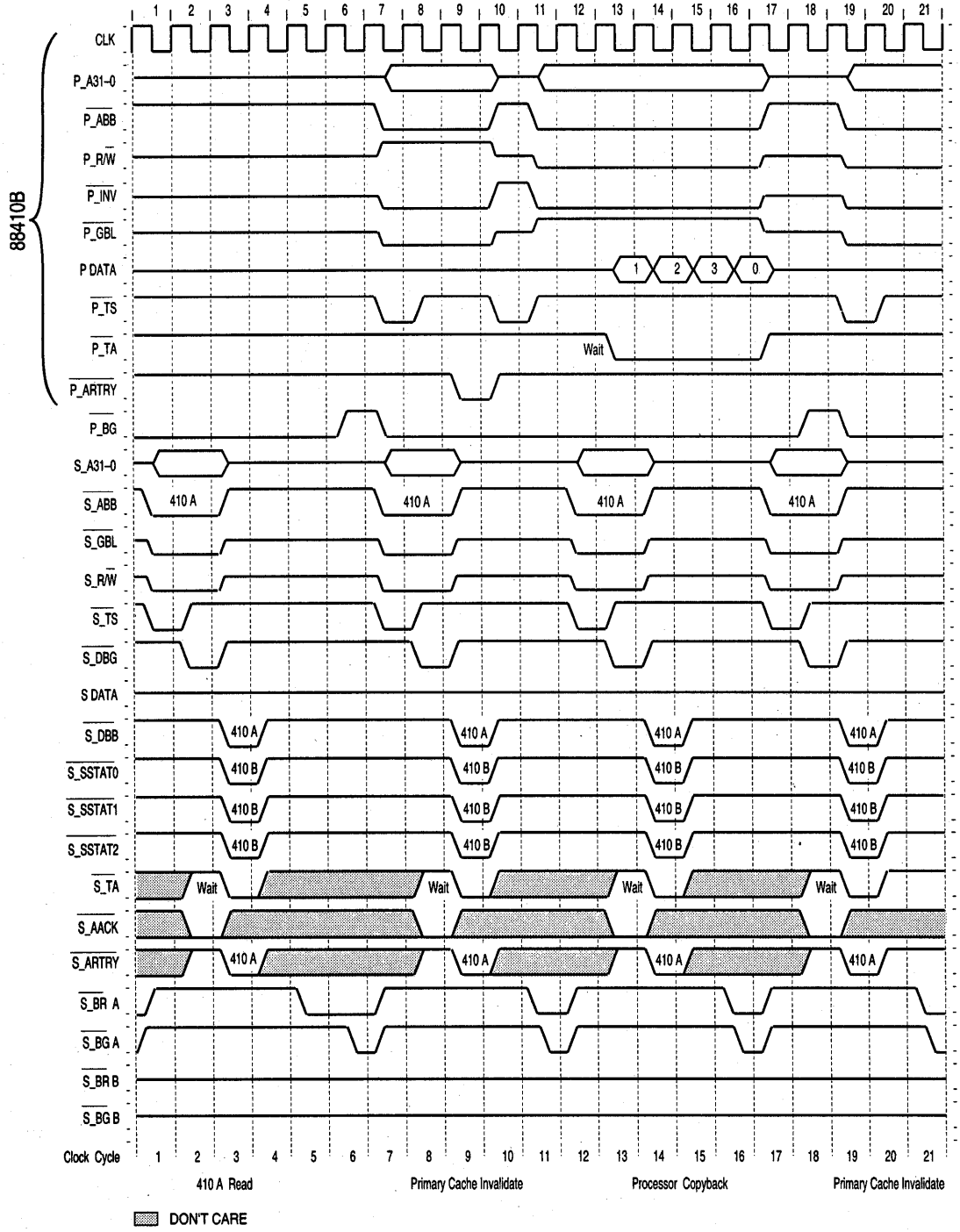


Figure 5-41a. Full-Speed Snoop Hit with Processor Copyback (Split Bus)

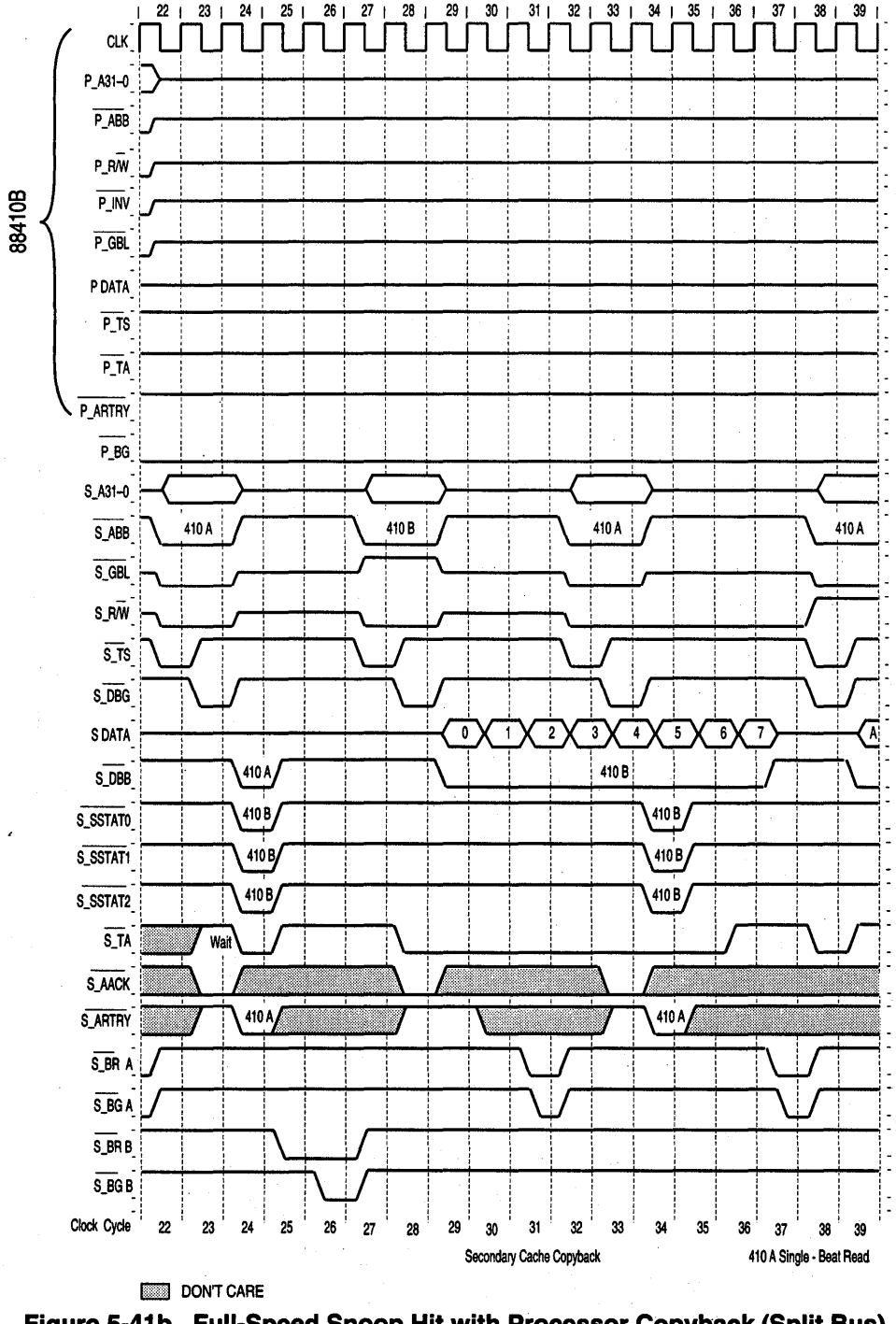


Figure 5-41b. Full-Speed Snoop Hit with Processor Copyback (Split Bus)

5.8 COLLISIONS

Collisions occur when the MC88410 cannot respond to a transaction due to internal resource conflicts. The following paragraphs describe the response of the MC88410 to collisions.

5.8.1 Tag Access Collision

The main tag (MTAG) and processor tag (PTAG) are multiplexed between the system bus interface and the processor bus interface. All cache tag reads, whether for a processor request or a snoop request, take one clock cycle. Transactions that must write to the tag upon completion (for example a read miss with secondary cache line allocation) perform the write in a single cycle upon completion of the transaction. The cache tags are free in the cycle immediately following a tag read or write.

In the event of simultaneous access at both the processor and system bus interfaces, the system bus interface is given priority and the processor's access to the tag is delayed until the tags are free in the next clock. As a result of its priority, the MC88410 always responds to a system bus snoop in two clock cycles.

5.8.2 Split-Bus Snoop Collisions

The MC88410 contains a snoop latch that allows the MC88410 to save a snoop lookup that could not be serviced immediately due to internal resource contention without resnooping the address. The snoop latch ensures coherency in split-bus systems when the address bus transaction has terminated but the data transaction has not completed. In this case, the cache tag may need to be written to upon completion of the data transaction. If another system bus master attempts a global transaction to a cache line which is waiting for a tag update, it will be retried ($\overline{\text{S\_ARTRY}}$ asserted). This condition is defined as a snoop collision.

For example, an MC88410-A may initiate a global transaction and receive an $\overline{\text{S\_AACK}}$ before its data transaction is completed, thus allowing MC88410-B to initiate a transaction. MC88410-B may attempt a global transaction which requires MC88410-A to access the same cache tag that it will write to (upon completion of the data phase of its transaction).

Note that transactions that do not have a pending cache tag write (such as a write-through write or a cache-inhibited write miss) do not generate snoop collisions even though the address and data bus may be split during the transaction.

Also note that certain combinations of processor and system bus transaction timings may cause the MC88410 to issue a snoop collision based on only the index portion (see **2.1 Cache Organization**) of the address instead of the full address due to a potential collision of internal resources.

Figure 5-42 shows a timing example of a snoop collision. MC88410-A begins a global transaction in clock cycle 1. The $\overline{\text{S\_AACK}}$ signal is asserted at the end of clock 2 to signal that the address has been latched. MC88410-A relinquishes mastership of the address bus and internally latches the address it had been driving. In clock 5, MC88410-B begins a

global transaction for the same address. At the end of clock 6, $\overline{\text{S\_AACK}}$ is asserted for MC88410-B. When MC88410-A checks the address of the global transaction initiated by MC88410-B and detects that it is the same as the address for its transaction still in progress, it asserts $\overline{\text{S\_SSTAT1}}$ (which is connected to $\overline{\text{S\_ARTRY}}$) but does not assert a system bus request. MC88410-B then recognizes that it has received a qualified $\overline{\text{S\_ARTRY}}$ and terminates its transaction.

During this time, data is being transferred to MC88410-A. Note that if MC88410-B asserts $\overline{\text{S\_TS}}$ to retry the transaction before the collision is resolved by the data transfer in progress, then another collision occurs. In this example, the external arbiter avoids this condition by waiting to assert the bus grant to MC88410-B until the last clock cycle of data transfer by MC88410-A.

5.8.3 Snoop Latch Full Collision

A snoop latch full collision occurs if the MC88410 detects a snoop hit while the snoop latch is still occupied by the address of a previous snoop transaction in progress (such as a primary cache invalidate transaction or a secondary cache copyback transaction). A snoop latch full collision causes $\overline{\text{S\_ARTRY}}$ to be asserted in the same manner as a snoop collision.

5.8.4 Lock Collision

Between the read and write halves on a cacheable locked transaction (**xmem**), the MC88410 maintains a lock collision buffer which retries any snooped address that attempts to access the same line address as the locked transaction. For information about locked transactions and coherency see **2.7.3 Locked Transactions.**

Figure 5-43 shows a lock collsion during a full-speed load-store locked transaction. The MC88410-A performs a locked transaction load, which hits in the secondary cache, followed by a locked store. In this example, the data is exclusive-unmodified so that a system invalidate transaction is not required before the read. In clock 4 another bus master initiates a transaction to the same cache line as the locked transaction. The MC88410 detects a lock collision with the snooped address and retries the transaction by asserting $\overline{\text{S\_SSTAT1}}$ (which is connected to $\overline{\text{S\_ARTRY}}$) in clock 6.

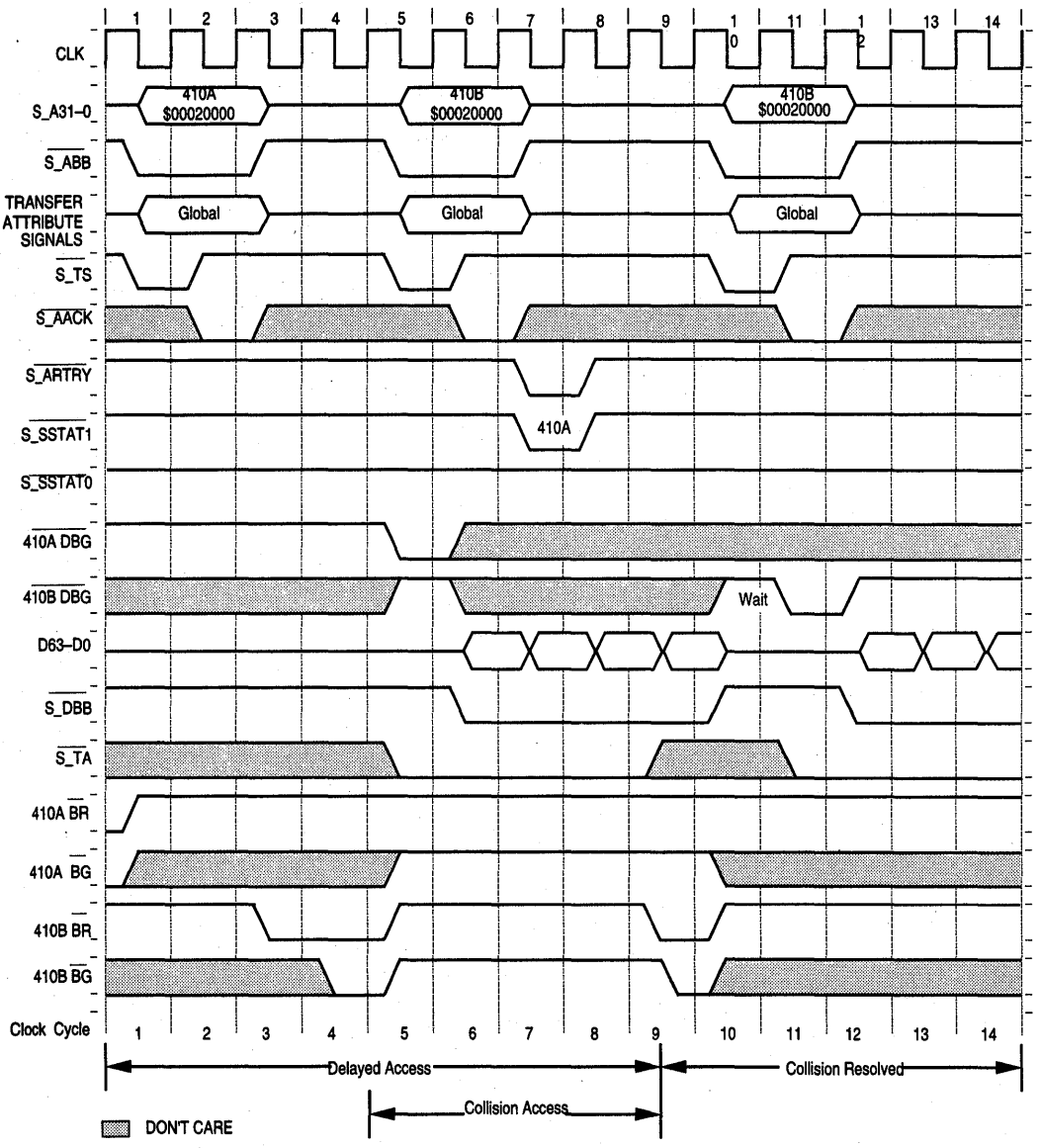


Figure 5-42. Snoop Collision Detection

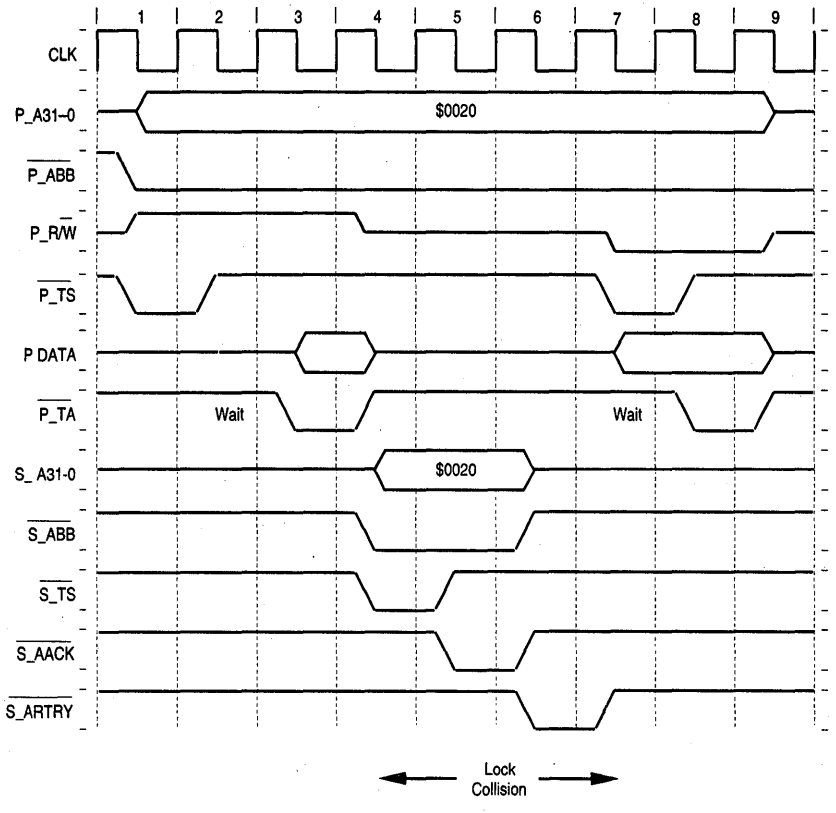


Figure 5-43. Lock Collision

5.9 RESET OPERATION

The reset ($\overline{\text{RST}}$) input signal is asserted by an external device to reset the processor. This initializes all internal logic (except for the cache tags) to a known state. It is recommended that the $\overline{\text{RST}}$ signals of the MC88110 and MC88410 be connected together.

Three to four clocks after $\overline{\text{RST}}$ is asserted, the MC88410 begins sampling the tag monitoring signals (FD2-FD0 and SD3-SD0) for configuration information (these signals are only inputs while $\overline{\text{RST}}$ is asserted). When $\overline{\text{RST}}$ is negated, the state of the configuration signals on the previous clock cycle is saved until the next time $\overline{\text{RST}}$ is asserted. This timing allows the configuration information to be removed at the same time as $\overline{\text{RST}}$. The configuration associated with each signal is described in Table 5-10.

Table 5-10. Reset Configuration Selection

Signal	Configuration Value	Configuration Result
FD2	1*	Reserved
FD1/CWM	0	Zero-word first on system bus interface
	1*	Critical-word first on system bus interface
FD0/LINSIZ	0	64-byte secondary cache line size
	1*	32-byte secondary cache line size
SD3/CSIZ1	11*	1/4-Mbyte cache size (CSIZ1=1, CSIZ0=1)
SD2/CSIZ0	10	Reserved (CSIZ1=1, CSIZ0=0)
	01	1- Mbyte cache size (CSIZ1=0, CSIZ0=1)
	00	Reserved (CSIZ1=0, CSIZ0=0)
SD1/ARBEN	1*	Internal processor interface arbitration
	0	External processor interface arbitration
SD0/CSP	1*	Chip selected when CS is high
	0	Chip selected when CS is low

* Default

When power is applied to the system, external circuitry should assert $\overline{\text{RST}}$ for a minimum of 200 ms after Vcc is within tolerance (assuming external pull-up resistors are used). Note that if internal pull-up transistors are used, additional time may be required. Figure 5-44 is a timing diagram of the power-on reset operation, showing the relationships between Vcc, $\overline{\text{RST}}$, and the bus signals. The CLK signal is required to be stable by the time Vcc reaches the minimum operating specification.

Once $\overline{\text{RST}}$ negates, the MC88410 is internally held in reset for another three clock cycles. During the reset period, $\overline{\text{S\_BR}}$ and $\overline{\text{S\_TS}}$ are negated, and all other three-statable signals are three-stated. Once the internal reset signal negates, the MC88410 grants the processor bus to the MC88110 (assuming on-chip arbitration) by asserting $\overline{\text{P\_BG}}$ in clock n+5. After this, the first bus transaction from the processor begins in clock n+6. The MC88410 asserts $\overline{\text{S\_BG}}$ in clock n+10 to gain system bus mastership to perform the transaction.

For MC88410 resets after the initial power-on reset, $\overline{\text{RST}}$ should be asserted for at least 16 clock cycles. Figure 5-45 shows the timing associated with a reset when the MC88410 is executing bus transactions and is then forced to invalidate the entire cache. Note that $\overline{\text{S\_ABB}}$ is negated before transitioning to a three-stated level. Resetting the MC88410 causes all output signals to three-state.

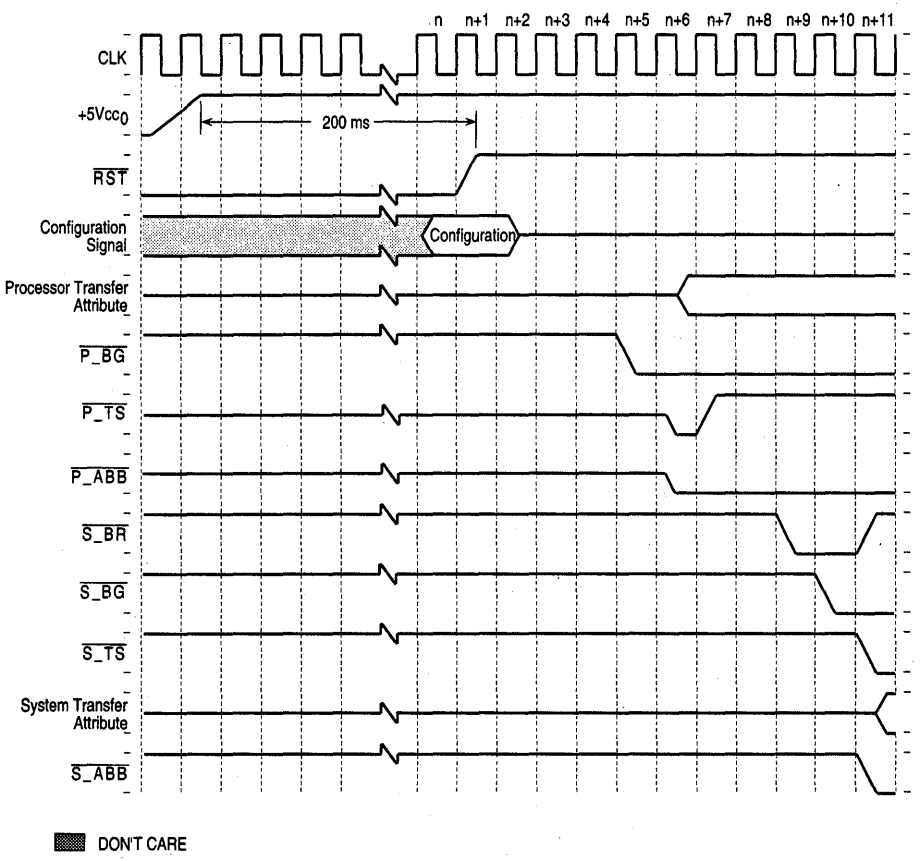


Figure 5-44. Initial Power-On Reset Timing

5

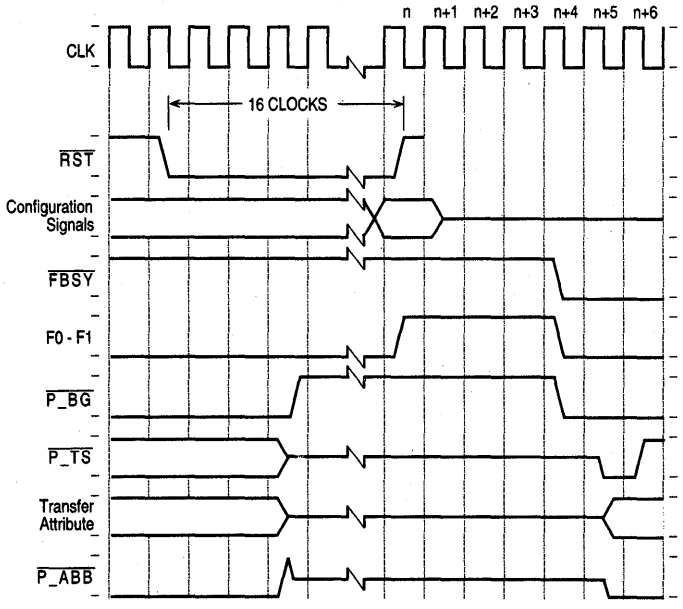


Figure 5-45. Normal Reset into Invalidate All

SECTION 6
DIAGNOSTICS AND JTAG

This section describes MC88410 tag monitoring, the function of the MC88410 in diagnostic mode, and MC88410 support for the *IEEE 1149.1–1990 Standard Test Access Port and Boundary-Scan Architecture.*

NOTE

The terms assert and negate are used extensively in this manual to avoid confusion between active-high and active-low signals. **Assert** or **assertion** indicates that a signal is active or true, regardless of whether the signal is active high or active low. **Negate** or **negation** indicates that the signal is inactive or false.

6.1 MC88410 TAG MONITORING

The MC88410 provides tag monitoring signals (FD2–FD0, SD3–SD0) for system debugging and statistical analysis of secondary cache performance. These signals function as configuration input signals at reset. After reset they are driven as an output on each tag access. The tag monitoring signals reflect the status of the cache tags on the clock previous to when they are sampled (one clock delayed). A logic analyzer can be used to capture the state of these signals.

The tag monitoring signals are grouped as function descriptors (FD2–FD0) and status descriptors (SD3–SD0). The function descriptors indicate the type of operation that is being performed on a tag line. The status descriptors indicate the contents of the tag line after that operation (except for flush operations and snoop operations that modify the tag between the read and write transactions).

Table 6-1 shows the encoding for the function descriptors and their corresponding tag operations. Four of the tag operations are read operations, three are write operations, and one is an idle state which indicates that no operation is being performed during that cycle. Transactions that modify the tags between a tag read and a tag write are identified as read-modify-write (RMW).

Snoop lookup, flush lookup, latched snoop lookup, and processor lookup accesses display the read status from the tag entry. The snoop and flush lookups could result in an RMW tag operation; however, only the read status is driven to SD3–SD0. A latched snoop access indicates that the address comes from the internal snoop latch rather than from the system interface. A PTAG lookup results from any processor access. Tag write

operations reflect all write operations that occur to end the transaction after a bus operation completes. Some operations write only to the MTAG, some only to the PTAG, and some to both.

Table 6-1. Tag Operations

FD2	FD1	FD0	Read	Write
0	0	0	Tag in idle state	—
0	0	1	Processor tag lookup	—
0	1	0	Snoop lookup (possible RMW)	—
0	1	1	Latched snoop lookup (possible RMW)	—
1	0	0	Flush lookup (possible RMW)	—
1	0	1	—	PTAG write
1	1	0	—	MTAG write
1	1	1	—	PTAG/MTAG write

RMW = Read-modify-write to tags

Table 6-2 shows the encoding for the status descriptors. Address misses cause the address to be written to the tag. Address hits do not write the address to the tag but only update the status bits. When a new line is being allocated, both the address and status bits are written to the tag.

Table 6-2. Tag Status Descriptors

SD3	SD2	SD1	SD0	Read	Write
0	0	0	I	Address miss (unused tag)	Go to invalid
0	0	1	I	Address miss (exclusive-unmodified)	Go to exclusive-unmodified
0	1	0	I	Address miss (exclusive-modified)	Go to exclusive-modified
0	1	1	I	Address miss (shared-unmodified)	Go to shared-unmodified
1	0	0	I	Address hit (unused tag)	Go to invalid
1	0	1	I	Address hit (exclusive-unmodified)	Go to exclusive-unmodified
1	1	0	I	Address hit (exclusive-modified)	Go to exclusive-modified
1	1	1	I	Address hit (shared-unmodified)	Go to shared-unmodified
H	S	M	V	Diagnostic MTAG read	—
0	0	0	I	Diagnostic PTAG read	—

I = Inclusion bit
H = MTAG hit bit
S = MTAG shared bit
M = MTAG modified bit
V = MTAG valid bit

The inclusion bit is stored in the PTAG to indicate whether the MC88110 has a copy of the cache line. The SDO signal indicates the state of the inclusion bit for a read or the new state of the inclusion bit for a write. For MTAG writes, SDO is driven low. In the diagnostic

mode, SD3–SD0 only reflect the status of read transactions. For diagnostic MTAG reads, SD3–SD0 reflect the state of the shared, modified, and valid bits of the MTAG entry. For diagnostic PTAG reads, SD0 reflects the state of the inclusion bit in the PTAG.

6.2 MC88410 DIAGNOSTIC MODE

The MC88410 provides the ability for diagnostic access to support read/write testing of the MTAG, PTAG, and MCM62110 array.

6.2.1 Diagnostic Accesses

The MC88410 supports four types of diagnostic accesses, as shown in Table 6-3.

Table 6-3. Diagnostic Access Types

Diagnostic Access	Operation
MTAG read/write	This access does a read or write of the MTAG and cache array without allocating a cache line or otherwise accessing the system interface. The MTAG read operation compares stored data to expected data for all bits on the selected line.
PTAG read/write	This access does a read or write of the PTAG and cache array without allocating a cache line or otherwise accessing the system interface. The PTAG read operation compares stored data to expected data for all bits on the selected line.
Bypass	This access bypasses the secondary cache and accesses the system bus without modifying the tags. This allows access to diagnostic instructions, memory-mapped board control registers, or data tables without affecting the MC88410.
System invalidate	This access is the same as a system invalidate transaction that is initiated under diagnostic mode in order to drive the contents of the MTAG or PTAG onto the system address signals.

6.2.2 Entering Diagnostic Mode

When the $\overline{\text{DIAG}}$ signal is asserted, the MC88410 is placed in diagnostic mode. All subsequent processor transactions are interpreted as diagnostic accesses. Snooping is disabled for all system bus addresses. The MC88410 recognizes F0 and F1 as inputs while $\overline{\text{DIAG}}$ is asserted but does not begin flush operation until $\overline{\text{DIAG}}$ is negated. If a flush or invalidate is in operation when $\overline{\text{DIAG}}$ is asserted, the operation halts, resuming at the next set index upon negation of $\overline{\text{DIAG}}$. The assertion of $\overline{\text{DIAG}}$ also changes the way that addresses are decoded and compared to the tags. If a transaction with a tag update is in progress when $\overline{\text{DIAG}}$ is asserted, the update is allowed to complete before tag decode is changed.

The $\overline{\text{DIAG}}$ signal must be held asserted throughout a diagnostic sequence as it is not latched by the MC88410. The $\overline{\text{DIAG}}$ signal must not be negated until two clocks after the MC88410 negates $\overline{\text{P\_TA}}$ to ensure that write transactions complete internally.

6.2.3 Diagnostic Encodings

The MC88410 qualifies incoming addresses with $\overline{\text{DIAG}}$. If $\overline{\text{DIAG}}$ is asserted the transaction is treated as diagnostic. All diagnostic transactions are cache-inhibited since the MC88110 always asserts $\overline{\text{P\_CI}}$ for diagnostic accesses. To differentiate between diagnostic read/write and diagnostic bypass accesses, the P_TC3–P_TC0 signals are used.

User-mode data accesses are interpreted as diagnostic read/write accesses while other accesses (except system invalidate) are treated as diagnostic bypass transactions. Diagnostic bypass transactions bypass the MC88410 to access external memory without affecting the MC88410 cache tags. Diagnostic system invalidate transactions are initiated with an MC88110 user touch load or allocate load instruction.

Table 6-4 shows the P_TC3–P_TC0 encoding related to diagnostic accesses. Note that the $\overline{\text{P\_CI}}$, $\overline{\text{P\_LK}}$, and $\overline{\text{P\_WT}}$ signals do not affect the interpretation of diagnostic accesses. The P_TC3–P_TC0 signals are interpreted by the MC88410 so that code fetches, table searches, and all supervisor mode requests are interpreted as diagnostic bypasses.

Table 6-4. Diagnostic Access Encoding

$\overline{\text{DIAG}}$	P_TC3	P_TC2	P_TC1	P_TC0	MC88410 Transaction
1	x	x	x	x	Normal
0	0	0	0	1	Diagnostic read or write
0	0	0	1	0	Diagnostic system invalidate
0	Anything else				Diagnostic bypass read or write

x= Don't care

6.2.4 Effect of Diagnostics on Coherence

Access of data in diagnostic mode affects transactions that normally bypass the cache and bus snooping. Diagnostic bypass read or write transactions differ from normal cache-inhibited transactions in their effect on the MTAG and PTAG. Diagnostic bypass transactions neither read nor write to the tags, unlike normal cache-inhibited transactions which check for cache hits and flush the line when necessary. This allows read and write testing of status bits without interference from transactions accessing instructions on the system bus.

6

Snooping is also affected by the diagnostic mode. Like the processor interface, the system interface qualifies all incoming addresses with $\overline{\text{DIAG}}$. System addresses are never snooped if they coincide with the assertion of $\overline{\text{DIAG}}$. The memory system must ensure that global system bus transactions do not occur during diagnostics of an MC88110/MC88410 node or that the MC88410 is kept in diagnostic mode throughout the entire diagnostic sequence. Before leaving diagnostic mode, any cache lines that have been made valid by the diagnostic testing of status bits must be explicitly invalidated with diagnostic writes to avoid unwanted copyback transactions or cache hits.

6.2.5 Addressing the Tags

Diagnostic read and write operations decode the processor address as shown in Figure 6-1. These address fields are used anytime $\overline{\text{DIAG}}$ is asserted, regardless of the programmed configuration of the MC88410. For each access, address bit 3 indicates whether the PTAG or MTAG is being accessed. For MTAG accesses, bits 17–5 are used to select which of the 8K tags are being selected. Bit 4 determines whether the high 8K or low 8K tags are being selected, providing a total of 16K tags. For PTAG accesses, bits 11–5 determine which set is being accessed and bit 4 determines which cache line is being accessed.

The configuration programmed into the MC88410 at reset does not affect the diagnostic mode. Assertion of $\overline{\text{DIAG}}$ places tag mapping into a specific mode that is independent of the normal tag mappings. This implies that the MC88410 is able to test tag bits that may never be used in normal operation. Since all data transfer in diagnostic mode is single-beat, cache line size information is not relevant.

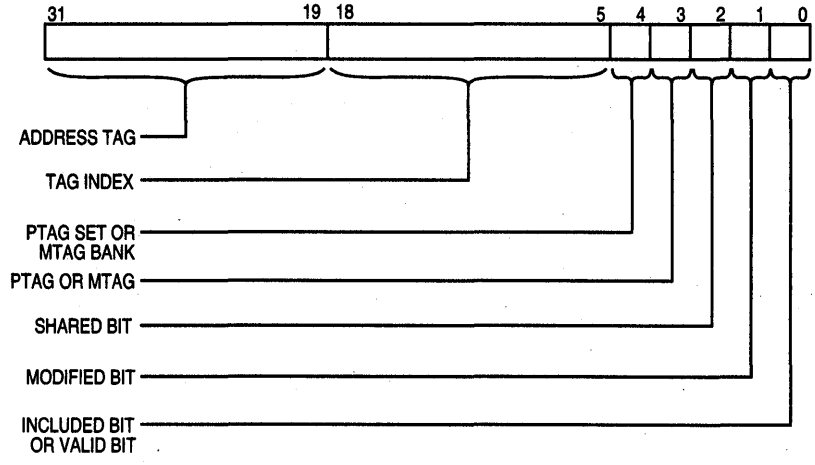


Figure 6-1. Diagnostic Access Address Fields

6.2.6 Reading and Writing to the MTAG

Diagnostic read and write operations may be used to perform write, read, and compare testing of all bits in the MTAG. Using the preceding tag addressing, a diagnostic write operation writes to the MTAG using bits 31–18 of the address on the processor bus for the address tag and bits 2–0 for the shared, modified, and valid bits respectively. Note that to avoid misaligned access exceptions when asserting these low order bits, the MC88110 must have the fault disabled during diagnostics.

A subsequent diagnostic read, using the same 32-bit address as the diagnostic write operation, accesses the specified set and reads the appropriate address bits and status bits. The MC88410 compares the data read from the tag to the address of the diagnostic read. If any of the address or status bits read from the MTAG do not match the appropriate bits from the read address, the MC88410 signals an error ($\overline{\text{P\_TEA}}$ asserted) to terminate the processor diagnostic read. If the address matches, the MC88410 asserts $\overline{\text{P\_TA}}$ to terminate the processor diagnostic read.

6.2.7 Reading and Writing to the PTAG

Diagnostic read and write operations can be used to perform write, read, and compare testing of all bits in the PTAG. Using tag addressing, a diagnostic write operation writes to the PTAG using bits 19–12 of the address on the processor bus for the address tag and bit 0 for the inclusion bit. Note that to avoid misaligned access exceptions when asserting the low order bits, the MC88110 must have the exception disabled during diagnostics.

A subsequent diagnostic read, using the same 32-bit address as the diagnostic write, accesses the tag and reads the address and inclusion bit. The MC88410 compares the data read from the tag to the appropriate address bits of the diagnostic load. If any of the address or status bits read from the MTAG do not match the appropriate bits from the read address, the MC88410 signals an error ($\overline{\text{P\_TEA}}$ asserted) to terminate the processor diagnostic read. If the address matches, the MC88410 asserts $\overline{\text{P\_TA}}$ to terminate the processor diagnostic read.

6.2.8 Reading and Writing to the Secondary Cache

Diagnostic read and write operations provide a convenient way to access the MCM62110 cache array. If data is driven onto the processor or system data bus during the diagnostic access, it will be written into the secondary cache. Data in the array is accessed using the address bits for the given cache configuration as described in **Section 2 Secondary Cache Operation** (for example, bits 11 to 0 are used to access the data locations in 256-Kbyte cache). Data may be accessed as byte, half-word, word, or double-word sizes. Write, read, and compare tests of the MCM62110 array can be done by using MC88410 diagnostic read and write operations to drive data to and read from the array and MC88110 compare instructions to check the results. Since the MC88410 is in diagnostic mode, MTAG and PTAG writes and compares occur in parallel with the intended data transaction. They are transparent to the test of the array unless a failure within the MC88410 causes a mismatch.

For example, asserting $\overline{\text{DIAG}}$ to performing a diagnostic write while driving data causes the address to be decoded as a diagnostic access and the data to be driven to the MCM62110 array. If the same address is then used to perform a diagnostic read, the data is driven to the MC88110 and the MC88410 compares the contents of the tag to the address being driven. If they match, the MC88410 asserts $\overline{\text{P\_TA}}$ to terminate the transaction. If the tag and address do not match, $\overline{\text{P\_TEA}}$ is asserted. The MC88410 compare instructions can be used to compare the data.

6.2.9 Bypassing the Secondary Cache

Transactions that result in a diagnostic bypass of the secondary cache (see Table 6-4) do not affect the diagnostics themselves, even though the instructions or data will be passed through the secondary cache. This is because the tags are neither read nor modified for diagnostic accesses and data is always transferred to or from the system interface.

6.2.10 Diagnostic System Invalidate

Diagnostic system invalidate transactions allow the contents of the MTAG and PTAG to be driven onto the system bus address signals A31–A0. The diagnostic system invalidate can be initiated by the processor using a touch-load or load-allocate transaction. In response to the processor transaction, the MC88410 initiates a system invalidate transaction, but negates $\overline{\text{S\_GBL}}$ instead of asserting it. The negation of $\overline{\text{S\_GBL}}$ for a system invalidate transaction identifies it as a diagnostic system invalidate.

A diagnostic system invalidate uses the tag addressing shown in Figure 6-1. If the access is for the MTAG, system address bits 31–18 are from the MTAG, bits 17–3 are from the address of the processor transaction, and bits 2–0 are from the V, M, and S bits of the MTAG. For PTAG accesses, bits 31–20 are from the processor address, bits 19–12 are from the PTAG, bits 11–1 from the processor, and bit 0 is from the inclusion bit of the PTAG.

6.3 IEEE 1149.1–1990 TEST ACCESS PORT

The MC88410 includes dedicated user-accessible test logic that is fully compatible with the *IEEE 1149.1–1990 Standard Test Access Port and Boundary-scan Architecture*. Problems associated with testing high density circuit boards have led to development of this standard under the sponsorship of the Test Technology Technical Committee of the IEEE Computer Society and the Joint Test Action Group (JTAG). The MC88410 implementation supports circuit board test strategies based on this standard.

The test logic implemented on the MC88410 includes a test access port (TAP) consisting of five dedicated signals, a 16-state controller, and two test data registers. A boundary-scan register links all device signals into a single shift register. The test logic is implemented using static logic design and is independent of the system logic of the device. Unlike the MC88110, which contains only input and bidirectional test cells, the MC88410 also includes output-only test cells. The MC88410 implementation provides capabilities to do the following:

- Perform boundary-scan operations to test circuit board electrical continuity.
- Bypass the MC88410 for a given circuit board test by effectively reducing the test data register to a single cell.
- Sample the MC88410 system signals during operation and transparently shift out the result in the boundary-scan register.
- Statically control the output state (high, low, or high-impedance) of all signals that can be outputs. The control state is latched or clamped within the MC88410 device even though the enabled test data register is the single-bit bypass register.
- Quickly force all bidirectional signals into the high-impedance state while enabling the single-bit bypass register as the test data register.
- Enable a weak pull-up current device on all signals controlled by the boundary-scan register while performing boundary-scan operations to provide for a deterministic test result in the event of a continuity exception.

NOTE

Certain precautions must be observed to ensure that the IEEE 1149.1–1990 test logic does not interfere with nontest operation. See **6.3.2.8 Non-IEEE 1149.1–1990 Operation** for details.

6.3.1 JTAG Overview

This document includes those aspects of the IEEE 1149.1–1990 implementation that are specific to the MC88410 and is intended to be used in conjunction with the supporting IEEE document. The scope of this description includes those items required by the standard to be defined and, in certain cases, provides additional information specific to the MC88410 implementation. For internal details and applications of the standard, refer to *IEEE 1149.1–1990 Standard Test Access Port and Boundary-scan Architecture.*.

A block diagram of the MC88410 implementation of IEEE 1149.1–1990 test logic is shown in Figure 6-2. The MC88410 implementation includes a dedicated TAP consisting of the signals shown in Table 6-5.

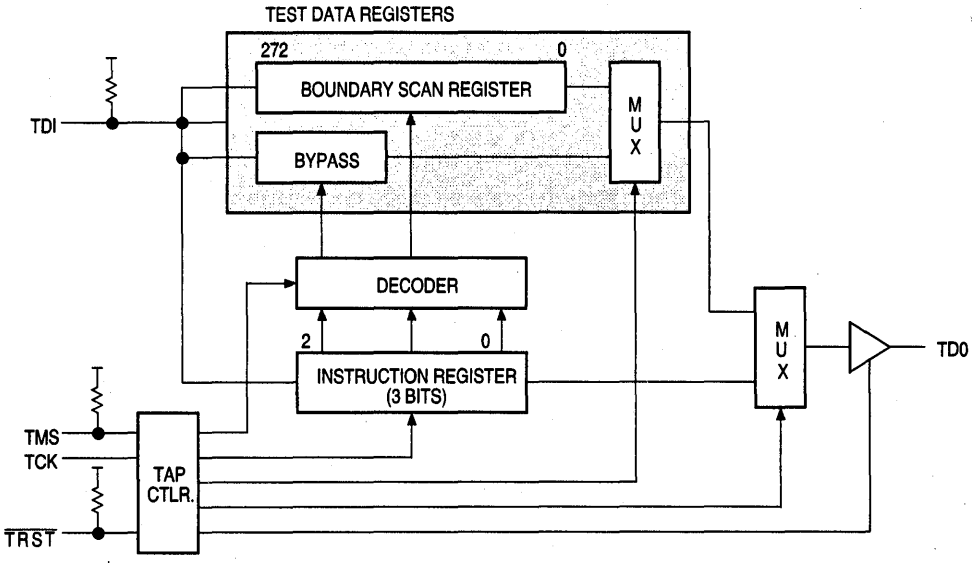


Figure 6-2. IEEE 1149.1 Test Logic Block Diagram

Table 6-5. Test Access Port Signals

Signal	Function
TCK	A test clock input to synchronize the test logic
TMS	A test mode select input (with an internal pull-up resistor) sampled on the rising edge of TCK to sequence the test controller's state machine
TDI	A test data input (with an internal pull-up resistor) sampled on the rising edge of TCK
TDO	A three-statable test data output actively driven in the shift-IR and shift-DR controller states that changes on the falling edge of TCK
$\overline{\text{TRST}}$	An asynchronous reset with an internal pull-up resistor which provides initialization of the TAP controller and other logic as required by the standard

NOTE

The pull-up resistor will pull $\overline{\text{TRST}}$ out of test reset.

6.3.2 Three-Bit Instruction Register

The MC88410 IEEE 1149.1–1990 implementation includes the three mandatory public instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) and three optional public instructions (CLAMP, HI-Z, and EXTEST_PULLUP). The EXTEST_PULLUP instruction is very similar to the EXTEST instruction; however, in the EXTEST_PULLUP instruction, the DC parametric of each signal controlled by the boundary-scan register is affected by the addition of a weak pull-up device. The MC88410 includes a 3-bit instruction register without parity as shown in Figure 6-3. The register consists of an instruction shift register

and a parallel output register. Data is transferred from the instruction shift register to the parallel output register during the update-IR controller state. The three bits are used to decode the six unique instructions as shown in Table 6-6.

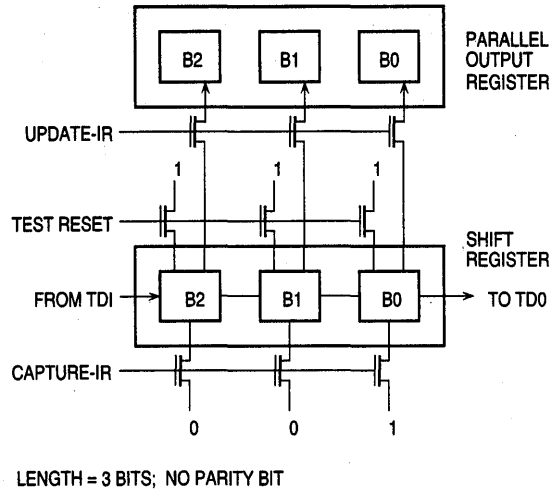


Figure 6-3. Instruction Register Implementation

The parallel output of the instruction register is preset to all ones in the test-logic-reset controller state. Note that this preset state is equivalent to the BYPASS instruction.

Table 6-6. Instruction Register Encodings

Code			Instruction
B2	B1	B0	
1	1	1	BYPASS
1	1	0	Reserved (BYPASS)
1	0	1	Reserved (BYPASS)
1	0	0	SAMPLE/PRELOAD
0	1	1	CLAMP
0	1	0	EXTEST_PULLUP
0	0	1	HI-Z
0	0	0	EXTEST

During the capture-IR controller state, the parallel inputs to the instruction shift register are loaded with the 3-bit binary value, 001. The parallel outputs, however, remain unchanged by this action since an update-IR signal is required to modify them.

Note that skipping the shift-IR state allows the 001 value to be updated as the current instruction, therefore entering the HI-Z instruction. This is useful for the board test applications that are not using the fully integrated boundary-scan test techniques, but would still like to use the HI-Z instruction for board test isolation purposes.

6.3.2.1 EXTEST (000)

The external test (EXTEST) instruction selects the boundary-scan register, including cells for all device, clock, and associated control signals. The EXTEST instruction also asserts internal reset for the MC88410 system logic in order to force a predictable internal state while performing external boundary-scan operations.

By using the TAP, the boundary-scan register is capable of scanning user-defined values into the output buffers, capturing values presented to input signals, and controlling the direction and value of bidirectional signals.

The boundary-scan register has bit cells associated with 30 pure input signals and 29 pure output signals. The other 214 cells are associated with 107 bidirectional signals. Each MC88410 bidirectional signal has both a boundary-scan register bit for signal data and a boundary-scan register bit for direction control. This allows great flexibility and control of the direction of every bidirectional signal. Due to the implementation of the individual direction control cell for each signal, some signals that are otherwise output-only can be programmed as input and have input data sampled into the boundary-scan register. For an executable boundary-scan description language (BSDL) listing see **6.3.3 Boundary-scan Definition List.**

The BSDL references the four boundary-scan cell types depicted in Figures 6-4, 6-5, 6-6, and 6-8. Figure 6-7 shows the bidirectional cell arrangement. The input-only cell (I.CELL) corresponds to BC_4 in the BSDL. The compound input and output cell (IO.CELL) corresponds to BC_6 in the BSDL. The bidirectional control cell (IO.CTL1) corresponds to BC_2. The output-only cell (O.LATCH) corresponds to BC_1. Note that when sampling the bidirectional data cells (IO.CELL), the cell data can be interpreted only after examining the IO.CTL1 cell to determine signal directionality.

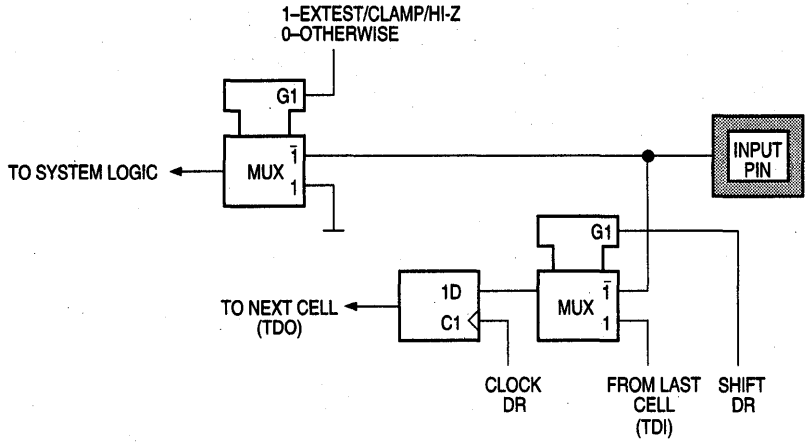


Figure 6-4. Input Pin Cell (I. Pin)

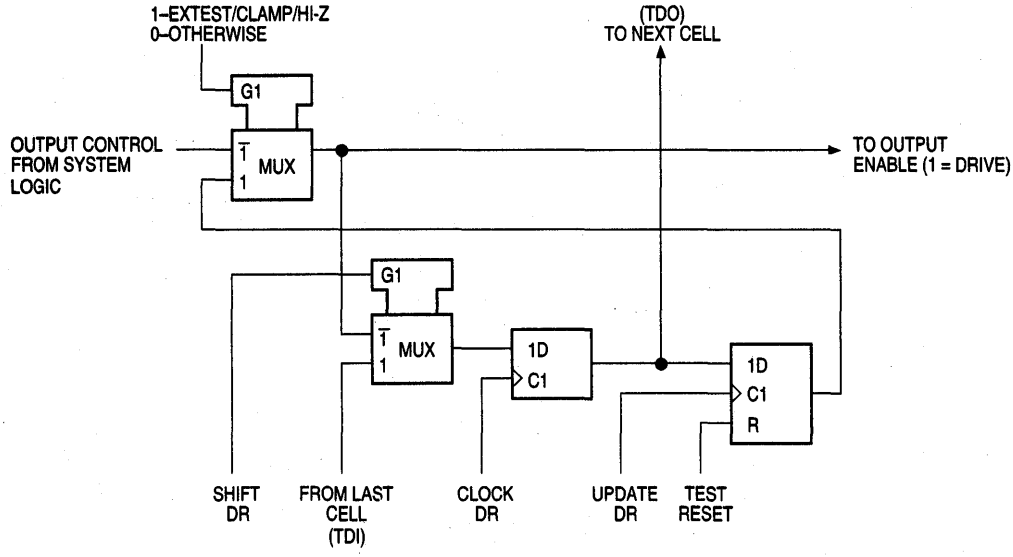


Figure 6-5. Active High Output Control Cell (IO.CTL1)

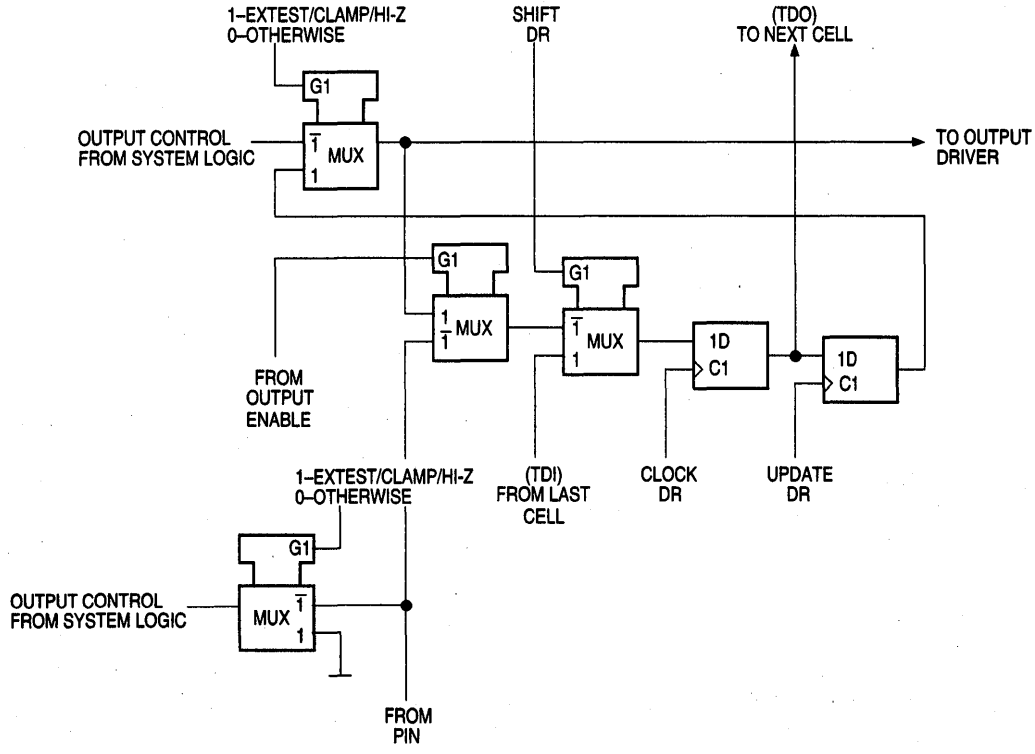


Figure 6-6. Bidirectional Data Cell (IO.Cell)

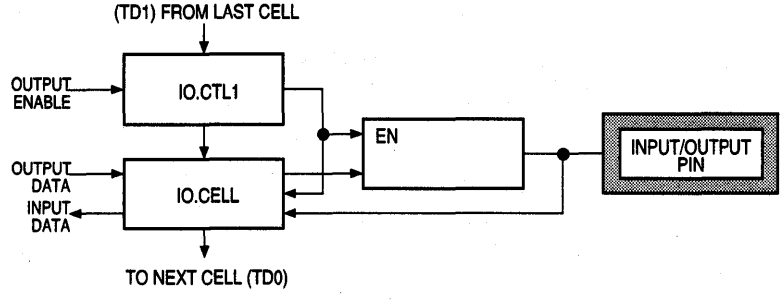


Figure 6-7. Bidirectional Cell Arrangement

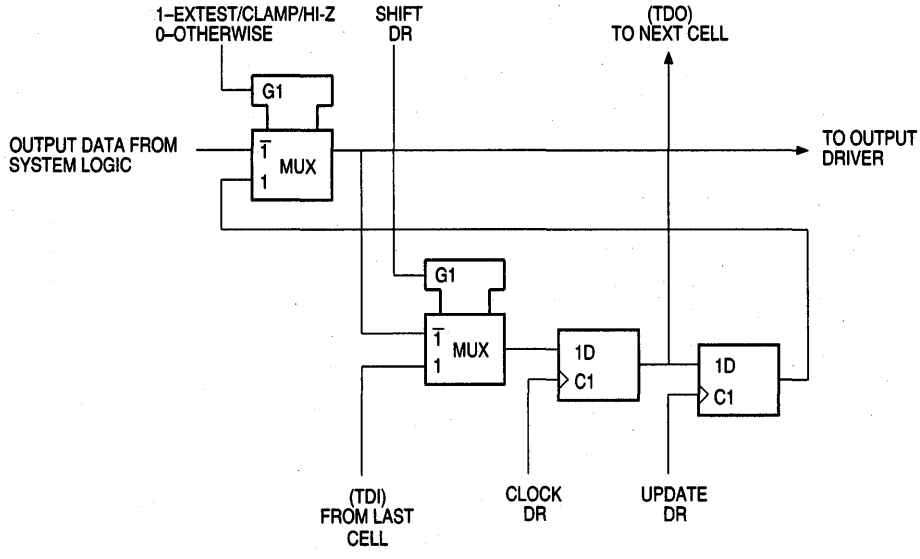


Figure 6-8. Output Latch Cell (O.Latch)

6.3.2.2 BYPASS (111)

The BYPASS instruction selects the single-bit bypass register as shown in Figure 6-9. This creates a shift-register path from the TDI signal to the bypass register and finally to the TDO signal, circumventing the boundary-scan register. This instruction improves test efficiency when a component other than the MC88410 is tested. In this instruction, the MC88410 system logic is independent of the test access port.

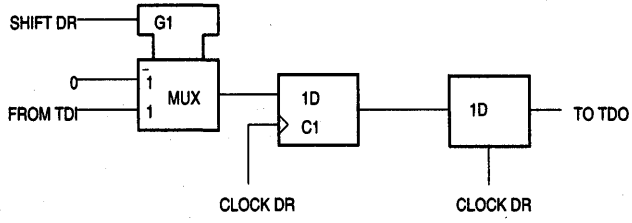


Figure 6-9. Bypass Register

When the bypass register is selected by the current instruction, the shift-register stage is set to a logic 0 on the rising edge of TCK following entry into the capture-DR controller state. Therefore, the first bit to be shifted out after selecting the bypass register is always a logic 0.

6.3.2.3 SAMPLE/PRELOAD (100)

The SAMPLE/PRELOAD instruction provides two separate functions. It provides a snapshot of system data and control signals. The snapshot occurs on the rising edge of TCK in the capture-DR controller state. The data can be observed by shifting it transparently through the boundary-scan register. In a normal system configuration many signals require external pull-ups to ensure proper system operation. Consequently, the same is true for the SAMPLE/PRELOAD functionality. The data latched into the boundary-scan register during capture-DR may not match the drive state of the package signal if the system-required pull-ups are not present within the test environment.

NOTE

Since there is no internal synchronization between the IEEE 1149.1–1990 clock (TCK) and the system clock (CLK), the user must provide some form of external synchronization to achieve meaningful results.

The SAMPLE/PRELOAD instruction also initializes the boundary-scan register output cells prior to selection of the EXTEST instruction. This ensures that known data appears on the outputs when entering the EXTEST instruction. During TAP reset, bidirectional signals preload the output control cell with output disable. In the SAMPLE/PRELOAD instruction, system logic is independent of the TAP.

6.3.2.4 CLAMP (011)

The CLAMP instruction is not included in the IEEE 1149.1–1990 standard, but it is provided as an optional public instruction to prevent having to backdrive the output signals during some methods of circuit board testing. When the CLAMP instruction is invoked, the package signals respond to the preconditioned values within the update latches of the boundary-scan register, even though the bypass register is enabled as the test data register.

In-circuit testing can be made easier by setting up the guarding signal conditions with use of the SAMPLE/PRELOAD or EXTEST instructions, and then as the MC88110 enters into the CLAMP instruction, the state and drive of all signals remain static until the instruction is disabled. While the signals continue to supply the guarding inputs to the in-circuit test location, the bypass register is enabled and thus should reduce overall test time.

6.3.2.5 HI-Z (001)

The HI-Z instruction is not included in the IEEE 1149.1–1990 standard. It is provided as an optional public instruction in order to prevent having to backdrive the output signals during circuit board testing. When the HI-Z instruction is invoked, all bidirectional drivers are turned off (i.e., three-state). However, R_A16–R_A0, $\overline{\text{RWE7}}$–$\overline{\text{RWE0}}$, $\overline{\text{PIE}}$, $\overline{\text{SIE}}$, and

$\overline{SOE}$ are pure output signals and are not three-stated by the HI-Z instruction. The instruction selects the bypass register.

6.3.2.6 EXTEST_PULLUP (010)

The EXTEST_PULLUP instruction is not included in the IEEE 1149.1–1990 standard, but is provided as an optional public instruction to aid in exception diagnoses during boundary-scan testing of a circuit board. This instruction is like EXTEST, except for a weak pull-up device on all signals. The MC88410 is a CMOS design and therefore could suffer from a logically indeterminate input value if an input or bidirectional signal programmed as an input were inadvertently unconnected. The pull-up current will, given an appropriate charging delay, supply a deterministic logic 1 result on an open input. Note that heavily loaded nodes may require a charging delay greater than the two TCK periods needed to transition from the update-DR state to the capture-DR state. Two solutions are available: transfer into the run-test/idle state for extra TCK periods of charging delay or simply change the period of TCK leading up to the capture edge of the capture-DR state.

6.3.2.7 MC88410 Restrictions

The control provided by the output enable signals using the boundary-scan register and the EXTEST or CLAMP instructions requires a compatible circuit board test environment to avoid configurations that can damage devices. The user must avoid enabling the MC88410 output drivers into actively driven networks.

The MC88410 includes on-chip circuitry to detect the initial application of power to the device. The power-on reset (POR) signal is the output of this circuitry and is used to reset both the system and IEEE 1149.1–1990 logic. POR is applied to the IEEE 1149.1–1990 circuitry to avoid the possibility of bus contention during power-on. The time to complete the device power-on process depends on the power supply. The IEEE 1149.1–1990 TAP controller, however, remains in the test-logic-reset state while POR is asserted. The TAP controller does not respond to user commands until POR is negated.

6.3.2.8 Non-IEEE 1149.1–1990 Operation

In non-IEEE 1149.1–1990 operation, two constraints must be met. The test clock input does not include an internal pull-up resistor; therefore, it should not be left unconnected (this is in order to prevent mid-level inputs). Also, the IEEE 1149.1–1990 test logic must be kept transparent to the system logic by forcing the TAP controller into the test-logic-reset controller state. During power-on, the POR signal forces the TAP controller into this state. However, to ensure that the controller remains in the test-logic-reset state, several options are described below.

- If TMS either remains unconnected or is connected to V_{CC}, the TAP controller cannot leave the test-logic-reset state regardless of the state of the TCK pin.
- $\overline{TRST}$ can be asserted either by connecting it to ground or by means of a logic network. Connecting $\overline{TRST}$ to the functional reset $\overline{RST}$ signal and tying TCK either high or low also meets this requirement.

- If $\overline{\text{TRST}}$ is asserted by a pulse signal, the controller remains in the test-logic-reset state in the absence of a rising edge on the TCK pin when TMS is low.

6.3.3 Boundary-Scan Definition List

The Boundary-Scan Description Language (BSDL) is a subset of the VHSIC Hardware Description Language (*IEEE 1076-1987VHDL*) and is a description of the testability features in IEEE 1149.1–1990. This language can be used by test equipment to provide testability analysis, test generation, and failure diagnosis. Design elements which are mandated by IEEE 1149.1–1990 are not included in the BSDL because they are described by the standard.

```
-- Motorola 88410 BSDL description


entity MC88410 is

  generic(PHYSICAL_PIN_MAP:string := "PGA_19x19");

  port(TRST_B: in   bit;
       TMS:   in   bit;
       TCK:   in   bit;
       TDI:   in   bit;
       TDO:   out  bit;
     T1OUT:  out  bit;
     RESET_B: in   bit;
     SD:          inout bit_vector(0 to 3);
     P_CL:   in   bit;
     P_TC:        in   bit_vector(0 to 3);
     P_WT_B:  in   bit;
     P_CI_B: in   bit;
     P_UPA_B: in   bit_vector(0 to 1);

     P_LK_B: in   bit;
     P_TSIZ: in   bit_vector(0 to 1);
     P_TBST_B: inout bit;
     P_RW_B: inout bit;
     P_A:         inout bit_vector(0 to 31);
     P_ABB_B: inout bit;
     P_ARTRY_B: in  bit;
     P_INV_B: inout bit;
     P_GBL_B: inout bit;
     P_TS_B: inout bit;
     CS:    in   bit;
     P_PTA_B: out  bit;
     P_BG_B: inout bit;
     P_BR_B: out  bit;
```

```
POE_B:  out   bit;
R_A:        out   bit_vector(0 to 16);
PIE_B:  out   bit;
RWE_B:    out   bit_vector(0 to 7);
SIE_B:      out   bit;
SOE_B:    out   bit;
P_TEA_B: out   bit;
P_TRTRY_B: out bit;
P_TA_B:  out   bit;
S_TA_B:  in    bit;
S_TRTRY_B: in  bit;
S_TEA_B: in    bit;
S_TS_B:  out   bit;
S_ARTRY_B: in  bit;
HCLK:   in    bit;
S_A:    inout bit_vector(0 to 31);

TSHD_B: in  bit;
SHD_B: in bit;
S_DBB_B: inout bit;
S_BR_B:  out   bit;
S_BG_B:  in    bit;
   S_DBG_B: in    bit;
   S_MC_B:  out   bit;
   S_INV_B: inout bit;
   S_ABB_B: inout bit;
   S_RW_B:  inout bit;
   S_TBST_B: inout bit;
   S_LK_B:  out   bit;
   F:         in    bit_vector(0 to 1);
   FBSY_B:  out   bit;
   S_SSTAT_B: out bit_vector(0 to 2);
   S_AACK_B: in   bit;
   DIAG_B:  in    bit;
   S_SR_B:  in    bit;
   S_GBL_B: inout bit;

   S_TSIZ:  out   bit_vector(0 to 1);
   S_UPA_B: out   bit_vector(0 to 1);
   S_CI_B:  out   bit;
   S_TC:    out   bit_vector(0 to 3);
   FD:        inout bit_vector(0 to 2);
   clk:    in    bit;

GND_E:  linkage bit_vector(1 to 32);
VDD_E:  linkage bit_vector(1 to 33);
```

```
    GND_I:  linkage bit_vector(1 to 17);
    VDD_I:  linkage bit_vector(1 to 19);
    GND_C:  linkage bit;
    VDD_C:  linkage bit;
       N_C:   linkage bit_vector(1 to 3);
       PLL_DIS: linkage bit;
       FL_TEST: linkage bit);

use STD_1149_1_1990.all;

attribute PIN_MAP of MC88410 : entity is PHYSICAL_PIN_MAP;

-- 19x19 PGA Pin Map

constant PGA_19x19 : PIN_MAP_STRING :=

       " TRST_B: L2, " &
       " TMS:  L3, " &
       " TCK:  K1, " &
       " TDI:  K2, " &
       " TDO:  K3, " &
       " CKMON: J2, " &
       " RESET_B: J3, " &
       " SD: (G1, H3, H2, H1), " &
       " P_CL: G2, " &
       " P_TC: (F3, F2, F1, G3), " &
       " P_WT_B: E1, " &
       " P_CI_B: E2, " &
       " P_UPA_B: (E3, D1), " &
       " P_LK_B: E4, " &
       " P_TSIZ: (C2, D3), " &
       " P_TBST_B: B1, " &
       " P_RW_B: B2, " &

    " P_A: (B15, A15, B14, A14, B13, A13, B12, A12, " &
    "      B11, A11, A10, B10, A9,  B9,  A8,  B8, " &
    "      A7,  B7,  A6,  B6,  A5,  B5,  A4,  B4, " &
    "      C5,  A3,  C4,  D5,  B3,  A2,  D4,  C3), " &
       " P_ABB_B: C14, " &
       " P_ARTRY_B: A16, " &
       " P_INV_B: B16, " &
       " P_GBL_B: C15, " &
       " P_TS_B: A17, " &
       " CS: C16, " &
       " P_PTA_B: B17, " &
       " P_BG_B: A18, " &
       " P_BR_B: B18, " &
```

```
    " POE_B: B19, " &

" R_A: (H19, H18, G19, G18, F19, F18, E19, E18, " &
"     F17, D19, D18, E17, C19, D17, C18, D16, " &
"     C17), " &
    " PIE_B: J18, " &
    " RWE_B: (N19, M18, M19, L18, L19, K18, K19, J19), " &
    " SIE_B: N18, " &
    " SOE_B: P17, " &
    " P_TEA_B: P19, " &
    " P_TRTRY_B: P18, " &
    " P_TA_B: R19, " &
    " S_TA_B: R18, " &
    " S_TRTRY_B: S19, " &
    " S_TEA_B: R17, " &
    " S_TS_B: S18, " &
    " S_ARTRY_B: T19, " &
    " HCLK: T18, " &
    " S_A: (U19, S17, T17, S16, U18, V19, " &
    "         V8, U8, V7, U7, V6, U6, V5, U5, " &
"     V4, U4, T5, V3, U3, T4, V2, S5, " &
"     T3, U2, S4, V1, T2, S3, U1, T1, " &
"     S2, S1), " &
    " TSHD_B: V18, " &
    " SHD_B: U17, " &
    " S_DBB_B: T16, " &
    " S_BR_B: V17, " &
    " S_BG_B: U16, " &
    " S_DBG_B: T15, " &
    " S_MC_B: V16, " &
    " S_INV_B: U15, " &
    " S_ABB_B: T14, " &
    " S_RW_B: V15, " &
    " S_TBST_B: U14, " &
    " S_LK_B: V14, " &
    " F: (U13, V13), " &
    " FBSY_B: U12, " &
    " S_SSTAT_B: (V12, V11, U11), " &
    " S_AACK_B: V10, " &
    " DIAG_B: U10, " &
    " S_SR_B: V9, " &
    " S_GBL_B: U9, " &
    " S_TSIZ: (R2, R1), " &
    " S_UPA_B: (P3, P2), " &
    " S_CI_B: P1, " &
    " S_TC: (N3, N2, N1, M3), " &
    " FD: (L1, M1, M2), " &
```

```
        " CLK: L5, " &
        " GND_E:    (K4, F5, G5, N5, P5, E6, R6, " &
        "              C7, E7, R7, T7, E9, R9, C10, " &
        "              T10, E11, R11, C12, E12, R12, T12, " &
        "              D14, S14, F15, H15, J15, L15, M15, " &
        "              P15, J16, H17, M17), " &
        " VDD_E:    (G4, L4, E5, R5, D7, S7, C8, " &
        "              T8, C9, D9, S9, T9, E10, R10, " &
        "              C11, T11, D12, S12, C13, E14, R14, " &
        "              D15, K15, S15, F16, H16, L16, M16, " &
        "              P16, G17, J17, L17, N17), " &
        " GND_I:    (F4, H4, M4, P4, D6, S6, D8, " &
        "              S8, D11, S11, D13, S13, E16, G16, " &
        "              K16, N16, R16), " &
        " VDD_I:    (J1, J4, N4, H5, M5, C6, T6, " &
        "              E8, R8, D10, S10, E13, R13, T13, " &
        "              E15, G15, N15, R15, K17), " &
        " GND_C: J5, " &
        " VDD_C: K5, " &
        " N_C: (C1, D2, A19), " &
        " PLL_DIS: R4, " &
        " FL_TEST: R3 " ;


-- Other Pin Maps here when documented


  attribute TAP_SCAN_IN of TDI:signal is true;
  attribute TAP_SCAN_OUT of TDO:signal is true;
  attribute TAP_SCAN_MODE of TMS:signal is true;
  attribute TAP_SCAN_CLOCK of TCK:signal is (10.0e6, BOTH);
  attribute TAP_SCAN_RESET of TRST_B:signal is true;

  attribute INSTRUCTION_LENGTH of MC88410:entity is 3;

  attribute INSTRUCTION_OPCODE of MC88410:entity is

   "EXTEST    (000),"    &
   "HIZ       (001),"    &
   "PULLUP    (010)," &
   "CLAMP     (011)," &
   "SAMPLE    (100)," &
   "BYPASS    (101, 111, 110)";

  attribute INSTRUCTION_CAPTURE of MC88410:entity is "001";
  attribute INSTRUCTION_DISABLE of MC88410:entity is "HIZ";
```

```
attribute REGISTER_ACCESS of MC88410:entity is
 "BOUNDARY (PULLUP), " &
 "BYPASS  (HIZ, CLAMP) ";

attribute BOUNDARY_CELLS of MC88410:entity is
 "BC_1, BC_2, BC_4, BC_6";

attribute BOUNDARY_LENGTH of MC88410:entity is 273;

attribute BOUNDARY_REGISTER of MC88410:entity is
  --num      cell    port     function safe ccell dsval rslt
       "0    (BC_1,  T1OUT,           output2,          X), " &
       "1    (BC_4,  RESET_B, input,          X), " &
       "2    (BC_6,  SD(3),  bidir,           X,    3, 0, Z), " &
       "3    (BC_2,  *,      controlr,        0), "                &
       "4    (BC_6,  SD(2),  bidir,   X,    5, 0, Z), " &
       "5    (BC_2,  *,      controlr,        0), "                &
       "6    (BC_6,  SD(1),  bidir,       X,    7, 0, Z), " &
       "7    (BC_2,  *,      controlr,        0), "                &
       "8    (BC_6,  SD(0),  bidir,       X,    9, 0, Z), " &
       "9    (BC_2,  *,      controlr,        0), "                &
       "10   (BC_4,  P_CL,   input,       X), " &
       "11   (BC_4,  P_TC(3), input,          X), " &
       "12   (BC_4,  P_TC(2), input,       X), " &
       "13   (BC_4,  P_TC(1), input,          X), " &
       "14   (BC_4,  P_TC(0), input,       X), " &
       "15   (BC_4,  P_WT_B,         input,  X), " &
       "16   (BC_4,  P_CI_B,         input,      X), " &
       "17   (BC_4,  P_UPA_B(1), input,      X), " &
       "18   (BC_4,  P_UPA_B(0), input,     X), " &
       "19   (BC_4,  P_LK_B,         input,  X), " &
       "20   (BC_4,  P_TSIZ(1), input,     X), " &
       "21   (BC_4,  P_TSIZ(0), input,        X), " &
       "22   (BC_6,  P_TBST_B,  bidir,     X,    23, 0, Z), " &
       "23   (BC_2,  *,      controlr,        0), "                &
       "24   (BC_6,  P_RW_B,       bidir,      X,    25, 0, Z), " &
       "25   (BC_2,  *,      controlr,        0), "                &
       "26   (BC_6,  P_A(31), bidir,      X,    27, 0, Z), " &
       "27   (BC_2,  *,      controlr,        0), "                &
       "28   (BC_6,  P_A(30), bidir,      X,    29, 0, Z), " &
       "29   (BC_2,  *,      controlr,        0), "                &
       "30   (BC_6,  P_A(29), bidir,      X,    31, 0, Z), " &
       "31   (BC_2,  *,      controlr,        0), "                &
       "32   (BC_6,  P_A(28), bidir,      X,    33, 0, Z), " &
       "33   (BC_2,  *,      controlr,        0), "                &
       "34   (BC_6,  P_A(27), bidir,      X,    35, 0, Z), " &
       "35   (BC_2,  *,      controlr,        0), "                &
```

```
"36     (BC_6,  P_A(26), bidir,     X,    37, 0, Z), " &
"37     (BC_2,  *,       controlr,        0), "                &
"38     (BC_6,  P_A(25), bidir,     X,    39, 0, Z), " &
"39     (BC_2,  *,       controlr,        0), "                &
"40     (BC_6,  P_A(24), bidir,     X,    41, 0, Z), " &
"41     (BC_2,  *,       controlr,        0), "                &
"42     (BC_6,  P_A(23), bidir,     X,    43, 0, Z), " &
"43     (BC_2,  *,       controlr,        0), "                &
"44     (BC_6,  P_A(22), bidir,     X,    45, 0, Z), " &
"45     (BC_2,  *,       controlr,        0), "                &
"46     (BC_6,  P_A(21), bidir,     X,    47, 0, Z), " &
"47     (BC_2,  *,       controlr,        0), "                &
"48     (BC_6,  P_A(20), bidir,     X,    49, 0, Z), " &
"49     (BC_2,  *,       controlr,        0), "                &
"50     (BC_6,  P_A(19), bidir,     X,    51, 0, Z), " &
"51     (BC_2,  *,       controlr,        0), "                &
"52     (BC_6,  P_A(18), bidir,     X,    53, 0, Z), " &
"53     (BC_2,  *,       controlr,        0), "                &
"54     (BC_6,  P_A(17), bidir,     X,    55, 0, Z), " &
"55     (BC_2,  *,       controlr,        0), "                &
"56     (BC_6,  P_A(16), bidir,     X,    57, 0, Z), " &
"57     (BC_2,  *,       controlr,        0), "                &
"58     (BC_6,  P_A(15), bidir,     X,    59, 0, Z), " &
"59     (BC_2,  *,       controlr,        0), "                &
"60     (BC_6,  P_A(14), bidir,     X,    61, 0, Z), " &
"61     (BC_2,  *,       controlr,        0), "                &
"62     (BC_6,  P_A(13), bidir,     X,    63, 0, Z), " &
"63     (BC_2,  *,       controlr,        0), "                &
"64     (BC_6,  P_A(12), bidir,     X,    65, 0, Z), " &
"65     (BC_2,  *,       controlr,        0), "                &
"66     (BC_6,  P_A(11), bidir,     X,    67, 0, Z), " &
"67     (BC_2,  *,       controlr,        0), "                &
"68     (BC_6,  P_A(10), bidir,     X,    69, 0, Z), " &
"69     (BC_2,  *,       controlr,        0), "                &
"70     (BC_6,  P_A(9),          bidir,     X,    71, 0, Z), " &
"71     (BC_2,  *,       controlr,        0), "                &
"72     (BC_6,  P_A(8),          bidir,     X,    73, 0, Z), " &
"73     (BC_2,  *,       controlr,        0), "                &
"74     (BC_6,  P_A(7),          bidir,     X,    75, 0, Z), " &
"75     (BC_2,  *,       controlr,        0), "                &
"76     (BC_6,  P_A(6),          bidir,     X,    77, 0, Z), " &
"77     (BC_2,  *,       controlr,        0), "                &
"78     (BC_6,  P_A(5),          bidir,     X,    79, 0, Z), " &
"79     (BC_2,  *,       controlr,        0), "                &
"80     (BC_6,  P_A(4),          bidir,     X,    81, 0, Z), " &
"81     (BC_2,  *,       controlr,        0), "                &
"82     (BC_6,  P_A(3),          bidir,     X,    83, 0, Z), " &
```

```
"83     (BC_2, *,       controlr,       0), "               &
"84     (BC_6, P_A(2),          bidir,      X,    85, 0, Z), " &
"85     (BC_2, *,       controlr,       0), "               &
"86     (BC_6, P_A(1),          bidir,      X,    87, 0, Z), " &
"87     (BC_2, *,       controlr,       0), "               &
"88     (BC_6, P_A(0),          bidir,      X,    89, 0, Z), " &
"89     (BC_2, *,       controlr,       0), "               &
"90     (BC_6, P_ABB_B, bidir,      X,    91, 0, Z), " &
"91     (BC_2, *,       controlr,       0), "               &
"92     (BC_4, P_ARTRY_B, input,    X), " &
"93     (BC_6, P_INV_B, bidir,      X,    94, 0, Z), " &
"94     (BC_2, *,    controlr,    0), "          &
"95     (BC_6, P_GBL_B, bidir,                X,       96, 0, Z), " &
"96     (BC_2, *,    controlr,    0), "          &
"97     (BC_6, P_TS_B, bidir,           X,    98, 0, Z), " &
"98     (BC_2, *,    controlr,    0), "          &
"99     (BC_4, CS,      input,          X), " &
"100    (BC_6, P_PTA_B, bidir,      X,    101, 0, Z), " &
"101    (BC_2, *,       controlr,       0), "               &
"102    (BC_6, P_BG_B,          bidir,      X,    103, 0, Z), " &
"103    (BC_2, *,       controlr,       0), "               &
"104    (BC_6, P_BR_B,          bidir,      X,    105, 0, Z), " &
"105    (BC_2, *,       controlr,       0), "               &
"106    (BC_6, POE_B,           bidir,      X,    107, 0, Z), " &
"107    (BC_2, *,       controlr,       0), "               &
"108    (BC_1, R_A(16), output2,    X), " &
"109    (BC_1, R_A(15), output2,    X), " &
"110    (BC_1, R_A(14), output2,    X), " &
"111    (BC_1, R_A(13), output2,    X), " &
"112    (BC_1, R_A(12), output2,    X), " &
"113    (BC_1, R_A(11), output2,    X), " &
"114    (BC_1, R_A(10), output2,    X), " &
"115    (BC_1, R_A(9),  output2,    X), " &
"116    (BC_1, R_A(8),  output2,    X), " &
"117    (BC_1, R_A(7),  output2,    X), " &
"118    (BC_1, R_A(6),  output2,    X), " &
"119    (BC_1, R_A(5),  output2,    X), " &
"120    (BC_1, R_A(4),  output2,    X), " &
"121    (BC_1, R_A(3),  output2,    X), " &
"122    (BC_1, R_A(2),  output2,    X), " &
"123    (BC_1, R_A(1),  output2,    X), " &
"124    (BC_1, R_A(0),  output2,    X), " &
"125    (BC_1, PIE_B,   output2,    X), " &
"126    (BC_1, RWE_B(7), output2,   X), " &
"127    (BC_1, RWE_B(6), output2,   X), " &
"128    (BC_1, RWE_B(5), output2,   X), " &
"129    (BC_1, RWE_B(4), output2,   X), " &
```

```
"130    (BC_1, RWE_B(3), output2,     X), " &
"131    (BC_1, RWE_B(2), output2,     X), " &
"132    (BC_1, RWE_B(1), output2,     X), " &
"133    (BC_1, RWE_B(0), output2,     X), " &
"134    (BC_1, SIE_B,  output2,     X), " &
"135    (BC_1, SOE_B,  output2,     X), " &
"136    (BC_6, P_TEA_B, bidir,      X,     137, 0, Z), " &
"137    (BC_2, *,       controlr,          0), "                &
"138    (BC_6, P_TRTRY_B, bidir,    X,     139, 0, Z), " &
"139    (BC_2, *,       controlr,          0), "                &
"140    (BC_6, P_TA_B,        bidir,       X,     141, 0, Z), " &
"141    (BC_2, *,       controlr,          0), "                &
"142    (BC_4, S_TA_B,        input,       X), " &
"143    (BC_4, S_TRTRY_B, input,     X), " &
"144    (BC_4, S_TEA_B, input,       X), " &
"145    (BC_6, S_TS_B,        bidir,       X,     146, 0, Z), " &
"146    (BC_2, *,     controlr,     0), "            &
"147    (BC_4, S_ARTRY_B, input,     X), " &
"148    (BC_4, HCLK,  input,       X), " &
"149    (BC_6, S_A(0), bidir,            X,       150, 0, Z), " &
"150    (BC_2, *,     controlr,     0), "            &
"151    (BC_6, S_A(1), bidir,       X,     152, 0, Z), " &
"152    (BC_2, *,     controlr,     0), "            &
"153    (BC_6, S_A(2), bidir,       X,     154, 0, Z), " &
"154    (BC_2, *,     controlr,     0), "            &
"155    (BC_6, S_A(3), bidir,       X,     156, 0, Z), " &
"156    (BC_2, *,     controlr,     0), "            &
"157    (BC_6, S_A(4), bidir,       X,     158, 0, Z), " &
"158    (BC_2, *,     controlr,     0), "            &
"159    (BC_6,          S_A(5), bidir,       X,     160, 0, Z), " &
"160    (BC_2,          *,     controlr,     0), "            &
"161    (BC_4,          TSHD_B,  input,     X), " &
"162    (BC_4,          S_SHD_B, input,     X), " &
"163    (BC_6,          S_DBB_B, bidir,       X,     164, 0, Z), " &
"164    (BC_2,          *,     controlr,     0), "            &
"165    (BC_6,          S_BR_B,         bidir,       X,     166, 0, Z), " &
"166    (BC_2,          *,     controlr,     0), "            &
"167    (BC_4,          S_BG_B,         input,       X), " &
"168    (BC_4,          S_DBG_B, input,       X), " &
"169    (BC_6, S_MC_B,        bidir,       X,     170, 0, Z), " &
"170    (BC_2, *,       controlr,          0), "                &
"171    (BC_6, S_INV_B, bidir,       X,     172, 0, Z), " &
"172    (BC_2, *,       controlr,          0), "                &
"173    (BC_6, S_ABB_B, bidir,       X,     174, 0, Z), " &
"174    (BC_2, *,       controlr,          0), "                &
"175    (BC_6, S_RW_B,        bidir,       X,     176, 0, Z), " &
"176    (BC_2, *,       controlr,          0), "                &
```

```
"177   (BC_6, S_TBST_B, bidir,        X,    178, 0, Z), " &
"178   (BC_2, *,        controlr,     0), "                &
"179   (BC_6, S_LK_B,        bidir,   X,    180, 0, Z), " &
"180   (BC_2, *,        controlr,     0), "                &
"181   (BC_4, F(0),    input,      X), " &
"182   (BC_4, F(1),    input,      X), " &
"183   (BC_6, FBSY_B,       bidir,     X,    184, 0, Z), " &
"184   (BC_2, *,        controlr,     0), "                &
"185   (BC_6, S_SSTAT_B(0), bidir,  X,    186, 0, Z), " &
"186   (BC_2, *,        controlr,     0), "                &
"187   (BC_6, S_SSTAT_B(1), bidir,  X,    188, 0, Z), " &
"188   (BC_2, *,        controlr,     0), "                &
"189   (BC_6, S_SSTAT_B(2), bidir,  X,    190, 0, Z), " &
"190   (BC_2, *,        controlr,     0), "                &
"191   (BC_4, S_AACK_B, input,     X), " &
"192   (BC_4, DIAG_B,         input,    X), " &
"193   (BC_4, S_SR_B,         input,    X), " &
"194  (BC_6, S_GBL_B, bidir,     X,    195, 0, Z), " &
"195  (BC_2, *,    controlr,    0), "          &
"196   (BC_6, S_A(6),         bidir,     X,    197, 0, Z), " &
"197   (BC_2, *,        controlr,     0), "                &
"198   (BC_6, S_A(7),         bidir,     X,    199, 0, Z), " &
"199   (BC_2, *,        controlr,     0), "                &
"200   (BC_6, S_A(8),         bidir,     X,    201, 0, Z), " &
"201   (BC_2, *,        controlr,     0), "                &
"202   (BC_6, S_A(9),         bidir,     X,    203, 0, Z), " &
"203   (BC_2, *,        controlr,     0), "                &
"204   (BC_6, S_A(10), bidir,     X,    205, 0, Z), " &
"205   (BC_2, *,        controlr,     0), "                &
"206   (BC_6, S_A(11), bidir,     X,    207, 0, Z), " &
"207   (BC_2, *,        controlr,     0), "                &
"208   (BC_6, S_A(12), bidir,     X,    209, 0, Z), " &
"209   (BC_2, *,        controlr,     0), "                &
"210   (BC_6, S_A(13), bidir,     X,    211, 0, Z), " &
"211   (BC_2, *,        controlr,     0), "                &
"212   (BC_6, S_A(14), bidir,     X,    213, 0, Z), " &
"213   (BC_2, *,        controlr,     0), "                &
"214   (BC_6, S_A(15), bidir,     X,    215, 0, Z), " &
"215   (BC_2, *,        controlr,     0), "                &
"216   (BC_6, S_A(16), bidir,     X,    217, 0, Z), " &
"217   (BC_2, *,        controlr,     0), "                &
"218   (BC_6, S_A(17), bidir,     X,    219, 0, Z), " &
"219   (BC_2, *,        controlr,     0), "                &
"220   (BC_6, S_A(18), bidir,     X,    221, 0, Z), " &
"221   (BC_2, *,        controlr,     0), "                &
"222   (BC_6, S_A(19), bidir,     X,    223, 0, Z), " &
"223   (BC_2, *,        controlr,     0), "                &
```

```
"224    (BC_6, S_A(20), bidir,      X,    225, 0, Z), " &
"225    (BC_2, *,       controlr,         0), "                 &
"226    (BC_6, S_A(21), bidir,      X,    227, 0, Z), " &
"227    (BC_2, *,       controlr,         0), "                 &
"228    (BC_6, S_A(22), bidir,      X,    229, 0, Z), " &
"229    (BC_2, *,       controlr,         0), "                 &
"230    (BC_6, S_A(23), bidir,      X,    231, 0, Z), " &
"231    (BC_2, *,       controlr,         0), "                 &
"232    (BC_6, S_A(24), bidir,      X,    233, 0, Z), " &
"233    (BC_2, *,       controlr,         0), "                 &
"234    (BC_6, S_A(25), bidir,      X,    235, 0, Z), " &
"235    (BC_2, *,       controlr,         0), "                 &
"236    (BC_6, S_A(26), bidir,      X,    237, 0, Z), " &
"237    (BC_2, *,       controlr,         0), "                 &
"238    (BC_6, S_A(27), bidir,      X,    239, 0, Z), " &
"239    (BC_2, *,       controlr,         0), "                 &
"240    (BC_6, S_A(28), bidir,      X,    241, 0, Z), " &
"241    (BC_2, *,       controlr,         0), "                 &
"242    (BC_6, S_A(29), bidir,      X,    243, 0, Z), " &
"243    (BC_2, *,       controlr,         0), "                 &
"244    (BC_6, S_A(30), bidir,      X,    245, 0, Z), " &
"245    (BC_2, *,       controlr,         0), "                 &
"246    (BC_6, S_A(31), bidir,      X,    247, 0, Z), " &
"247    (BC_2, *,       controlr,         0), "                 &
"248    (BC_6, S_TSIZ(0), bidir,    X,    249, 0, Z), " &
"249    (BC_2, *,       controlr,         0), "                 &
"250    (BC_6, S_TSIZ(1), bidir,    X,    251, 0, Z), " &
"251    (BC_2, *,       controlr,         0), "                 &
"252    (BC_6, S_UPA_B(0), bidir,   X,    253, 0, Z), " &
"253    (BC_2, *,       controlr,         0), "                 &
"254    (BC_6, S_UPA_B(1), bidir,   X,    255, 0, Z), " &
"255    (BC_2, *,       controlr,         0), "                 &
"256    (BC_6, S_CI_B,          bidir,    X,    257, 0, Z), " &
"257    (BC_2, *,       controlr,         0), "                 &
"258    (BC_6, S_TC(0), bidir,      X,    259, 0, Z), " &
"259    (BC_2, *,       controlr,         0), "                 &
"260    (BC_6, S_TC(1), bidir,      X,    261, 0, Z), " &
"261    (BC_2, *,       controlr,         0), "                 &
"262    (BC_6, S_TC(2), bidir,      X,    263, 0, Z), " &
"263    (BC_2, *,       controlr,         0), "                 &
"264    (BC_6, S_TC(3), bidir,      X,    265, 0, Z), " &
"265    (BC_2, *,       controlr,         0), "                 &
"266    (BC_6, FD(2),   bidir,      X,    267, 0, Z), " &
"267    (BC_2, *,       controlr,         0), "                 &
"268    (BC_6, FD(1),   bidir,      X,    269, 0, Z), " &
"269    (BC_2, *,       controlr,         0), "                 &
"270    (BC_6, FD(0),   bidir,      X,    271, 0, Z), " &
```

```
        "271    (BC_2, *,       controlr,       0), "           &
        "272    (BC_4, CLK,     input,          X)" ;

end MC88410;
```

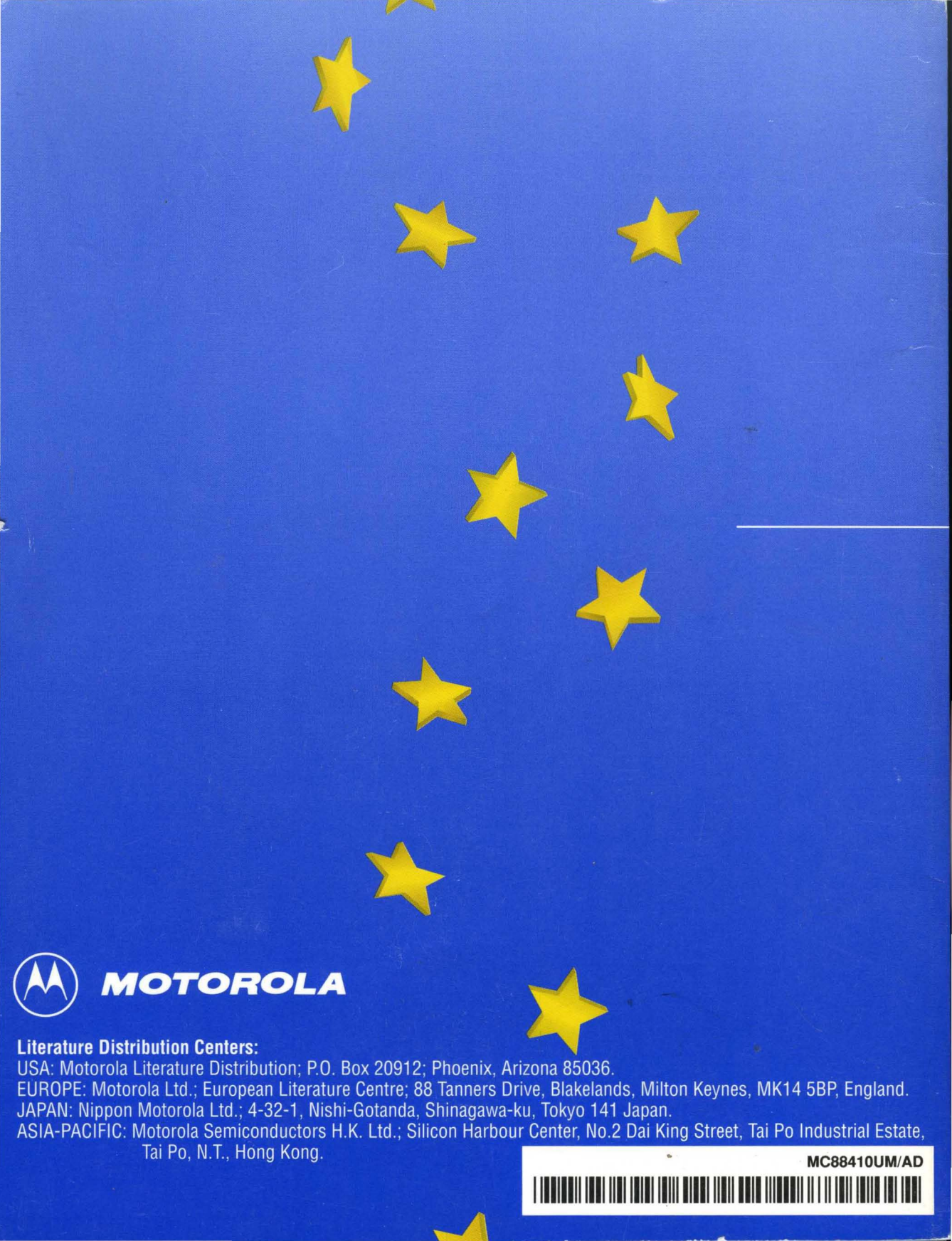
Index

–V–

–W–

–X–

I



MOTOROLA

Literature Distribution Centers:
USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.
ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No.2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

MC88410UM/AD