PowerPC™

The IBM27-82650 PowerPC to PCI Bridge User's Manual

PowerPC to PCI Bridge, Memory Controller, Arbiter, ROM Controller, and System Resource Manager

Introducing the 650 Bridge, with the 653 Buffer and 654 Controller, as a complete solution for interfacing PowerPC 60X microprocessors to the PCI local bus.

IBM

[illegible] 5

The IBM27-82650 PowerPC to PCI Bridge

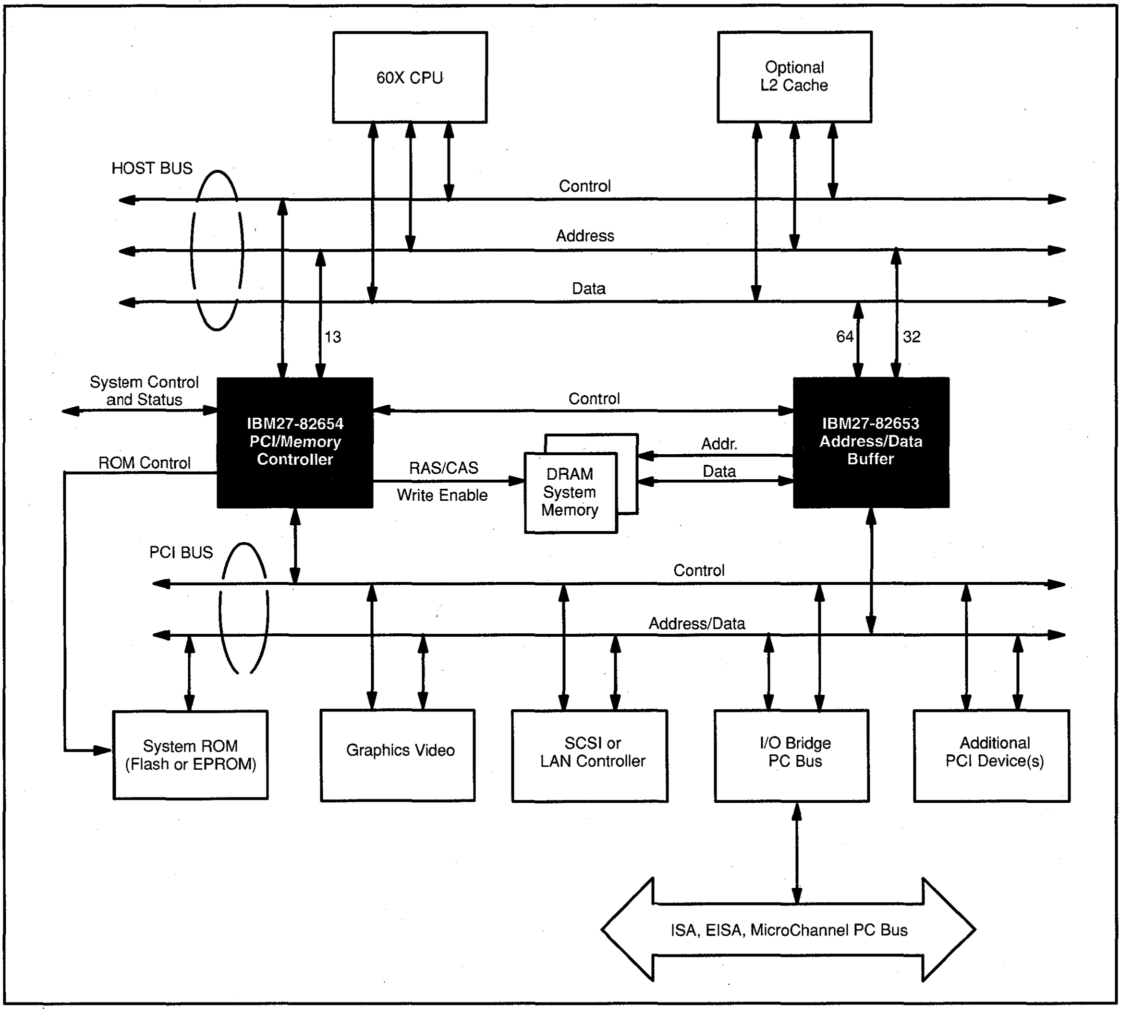
User's Manual

PowerPC to PCI Bridge, Memory Controller, Arbiter, ROM Controller, and System Resource Manager

IBM

This document contains preliminary information about version 2.0 of the chip set and is subject to change by IBM without notice. IBM assumes no responsibility or liability for any use of the information contained herein. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. The products described in this document are not intended for use in implantation or other direct life support applications where malfunction may result in physical harm or injury to persons. NO WARRANTIES OF ANY KIND, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ARE OFFERED IN THIS DOCUMENT.

The IBM27-82650
PowerPC to PCI Bridge
User's Manual



IBM 650 Bridge Chip Set in a Typical Configuration

The IBM27-82650 PCI Bridge Chip Set

PowerPC™ to PCI Local Bus Bridge and Memory Controller

- **Used in the PowerPC Reference Platform Reference Implementation**
- **Supports the PowerPC 601, 603, and 604 (60X) Microprocessors**
  - **Big- and Little-Endian Address Modes**
  - **Up to 66MHz 60X CPU Bus Clock**
- **Interfaces the 60X CPU Host Bus to the PCI Local Bus (level 2.0)**
  - **Up to 33MHz PCI Local Bus Clock Rate**
  - **Enables Industry-Standard PCI Devices**
  - **Integrated Arbiter, up to Six PCI Devices**
  - **One Load on PCI Bus**
- **Support for Optional L2 Cache**
  - **Write-Back and Write-Through Policy**
  - **Snoops PCI to System Memory Addresses**
- **ISA Master to System Memory Support**
- **Direct Support for 8-Bit Flash/EPROM**
- **System Memory (DRAM) Controller**
  - **64-Bit Data Path for System Memory**
  - **Directly Supports 8M and 32M 8-Byte SIMMs**
  - **Supports up to 256M of System Memory**
  - **Memory Configuration Registers**
  - **70ns SIMMs Supported**
  - **System Memory Parity Generation and Checking**
  - **Supports 60X CPU Burst Reads and Writes of System Memory**
  - **PCI Burst Reads and Writes to System Memory**
- **Companion L2 Cache Controller Chip**
  - **The IBM27-82681-66**
  - **44-Pin PLCC Package**
  - **256K or 512K Bytes**
- **3.3V or 3.6V Power Supply**
- **5V-Compatible I/O's**
- **304-Pin and 160-Pin Quad Flatpacks**

The IBM27-82650 PCI Bridge chip set (the 650 Bridge) provides an interface that can connect a PowerPC 60X CPU to high-performance, PCI (Peripheral Component Interconnect) devices like graphics, LAN, and SCSI controllers. The PCI bus standard defines an environment for high-speed, local bus operations. The 650 Bridge chip set provides the necessary control and communications logic to connect a PowerPC 60X CPU to PCI-compliant devices through the PCI bus.

The 650 Bridge chip set is comprised of the IBM27-82653 Address/Data Buffer (the 653 Buffer) and the IBM27-82654 PCI/Memory controller (the 654 Controller). The 650 Bridge supports the PowerPC 601, 603, and 604 microprocessor chips. The 650 Bridge supports both the L1 memory cache of the 60X CPU and an optional L2 cache. Either cache can use write-through or write-back modes of operation.

The 650 Bridge performs the following distinct functions:

- PCI and 60X CPU Bus Arbitration Logic
- System Memory (DRAM) Controller—up to 256M
- Memory-Mapping of CPU Addresses to PCI Transactions
- 60X CPU to 32-bit PCI Local Bus Bridge
- Supports the PowerPC Memory Coherence Model
- System ROM Controller (Including Flash ROM Writes)
- Parity Error and System Error Detection and Reporting

Table of Contents

About This Book

Section 3

Section 4

Section 5

Section 6

Section 9

Appendixes:

Appendix A

Appendix B

Appendix C

Tables:

Figures:

About This Book

Audience:

This book is designed for engineers and system designers who are interested in implementing PowerPC systems with a PCI bus. The material requires an understanding of computer systems at the hardware level.

Document Organization:

Section 1 — An architectural overview of the 650 Bridge with detailed lists of the features and functions of the 650 Bridge chip set.

Section 2 — A background review of the PCI Bus and 60X CPU.

Section 3 — 653 Buffer and 654 Controller pin description tables arranged in functional groups with a separate table for all the interconnections between the two chips.

Section 4 — Theory of operations, including basic timing diagrams.

Section 5 — A functional description of the 650 Bridge.

Section 6 — Electrical characteristics of the chip set.

Section 7 — Detailed timing diagrams and tables.

Section 8 — Alphabetic and numeric pin lists for the 653 Buffer and 654 Controller.

Section 9 — Mechanical drawings.

Appendix A — Initialization and setup requirements.

Appendix B — Example implementation schematics.

Appendix C — 653 Buffer details of operation.

Appendix D — Addresses of sales offices.

Reference Material:

- *PowerPC 601 RISC Microprocessor* User's Manual, IBM document number MPR601UMU-02
- *PowerPC 601 RISC Microprocessor Hardware Specifications*, IBM document number MPR601HSU-02
- *PCI Local Bus Specification*, Revision 2.0, April 30, 1993, available from the PCI SIG
- *PCI System Design Guide,* Revision 1.0, September 1993, available from the PCI SIG
- *32MB SIMM Engineering Specification,* IBM document number MMDS08DSU-00
- *8MB SIMM Engineering Specification,* IBM document number MMDS06DSU-00
- *PowerPC Reference Platform Specification,* Version 1.0, June 20, 1994
- *The IBM27-82681-66 PowerPC L2 Cache Controller User's Guide,* IBM document number MPRCL2UMU-01

Document Conventions:

Kilobytes, megabytes, and gigabytes are indicated by a single capital letter after the numeric value. For example, 4K means 4 kilobytes, 8M means 8 megabytes, and 4G means 4 gigabytes.

Fractional time values are identified with the terms ms, us, and ns, which represent milliseconds, microseconds, and nanoseconds respectively.

Hexadecimal values are identified with a lower-case letter h at the end of the value. For example, 001Fh means a 16-bit hexadecimal value of 1F. The letters A through F in the hexadecimal numbering system are always capitalized.

Binary values are identified with a lower-case letter b at the end of the value. For example, 0101b means a 4-bit binary value of 0101 (decimal five).

In identifying ranges of values *from* and *to* are used whenever possible. The range statement *from 0 to 2M* means from zero up to (but not including) two megabytes. The hexadecimal value for the range from 0 to 64K is: 0000h to FFFFh. This method is used in preference to constantly adding a – *1* term to the end of range statements.

The # symbol at the end of a signal name indicates that the active or asserted state of the signal occurs with a low voltage level. When the # symbol is not present after the signal name, the signal is asserted with a high voltage level.

The terms *asserted* and *negated* are used extensively. The term *asserted* indicates that a signal is active, regardless of whether that level is represented by a high or low voltage. The term *negated* means that a signal is inactive. The term *deasserted* is also used to indicate a signal that is negated.

The names of signals are in all upper-case letters. For example, TS#, AACK#, and PCI_CLK are all signal names.

Signals with more than one pin are identified with square brackets and numbers after the pin names. For example, PCI_AD[31:0] and TT[0:2] are signals with multiple pins. The # symbol that indicates that a signal is asserted low appears after the square brackets. For example, CAS[7:0]#.

Individual pins within a multi-pin signal group are identified with the pin number within the group in square brackets after the pin name. For example, PCI_AD[5].

Multiple-pin signals that have the first number larger than the second number (PCI_AD[31:0] for example) are little-endian signals. Multiple-pin signals that have the first number smaller than the second number (TT[0:2] for example) are big-endian signals.

Acronyms and Abbreviations:

In this document, the term 60X CPU refers to the PowerPC 601, 603, and 604 microprocessors.

The term I/O Bridge refers to a PCI master that serves to connect the PCI bus to a PC-standard bus like the ISA, EISA, or MicroChannel buses.

The term RAS refers to the row address select lines of the memory controller.

The term CAS refers to the column address select lines of the memory controller.

The term write-back means the same as copy-back in reference to a mode of cache operation.

The acronym PIO refers to I/O controller interface operations on the 60X CPU bus.

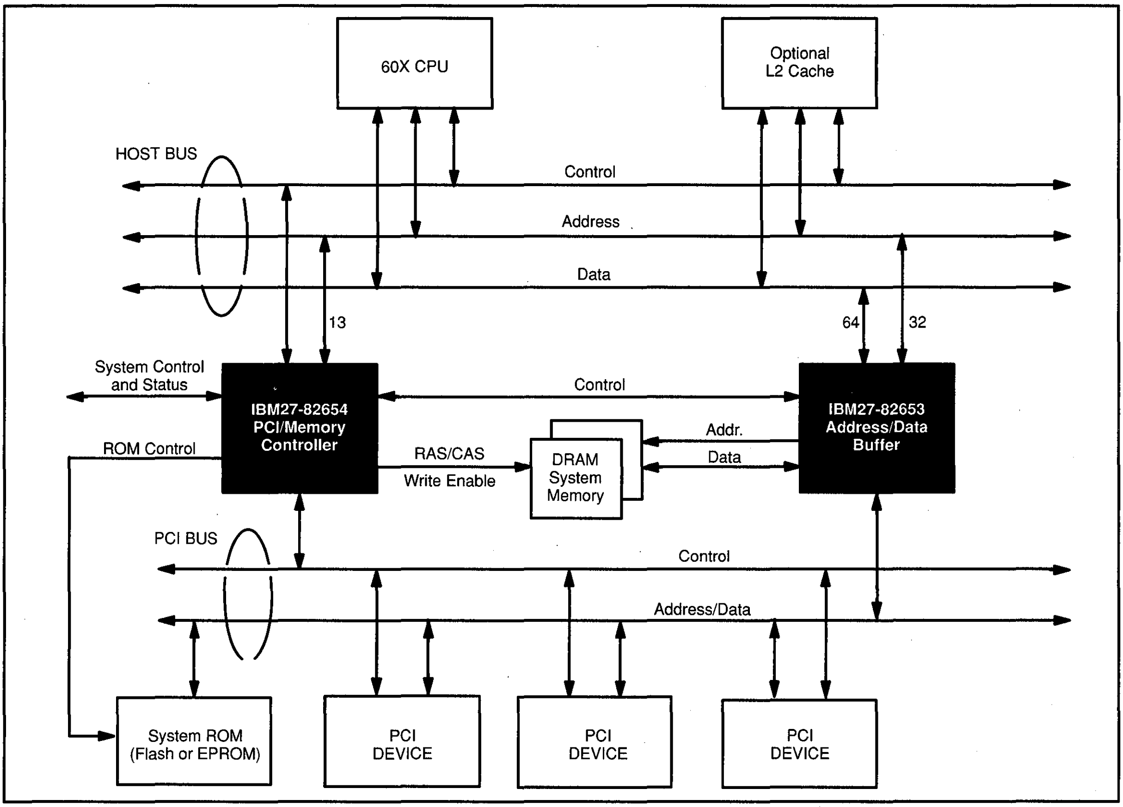


Figure 1–1. IBM 650 Bridge Chip Set in a Typical System Configuration

Section 1
650 Bridge Architectural Overview

The IBM27-82650 PCI Bridge chip set (the 650 Bridge) provides an interface that can connect a PowerPC 60X CPU to high-performance PCI (Peripheral Component Interconnect) devices like graphics, LAN, and SCSI controllers. The PCI bus standard defines an environment for high-speed local bus operations. The 650 Bridge chip set provides the necessary control and communications logic to connect a PowerPC 60X CPU to PCI-compliant devices through the PCI bus.

The 650 Bridge chip set is comprised of the IBM27-82653 Address and Data Buffer (the 653 Buffer) and the IBM27-82654 PCI and Memory Controller (the 654 Controller). The 650 Bridge supports the PowerPC 601™, PowerPC 603™, and PowerPC 604™ microprocessor chips. Within this document, the three microprocessor chips (601, 603, and 604) are referred to generically as the 60X CPU. The 650 Bridge supports both the L1 memory cache of the 60X CPU and an optional L2 cache. Either cache can use write-through or write-back modes of operation.

Local bus standards like PCI and the VL-Bus have evolved to answer the need for higher performance I/O operations on microcomputer systems. The 650 Bridge provides an interface mechanism between PowerPC CPUs and the PCI bus. This interface allows system designers to take advantage of the standard PCI controllers that are available for many I/O applications.

Figure 1–1 shows a typical PowerPC to PCI system. The address, data, and control signals from the 60X CPU host bus are connected to the 650 Bridge. An optional L2 (level 2) cache can also be connected to the host bus. (The L1 cache resides in the 60X microprocessor.) The 650 Bridge is connected to the PCI local bus (address/data and control signals) and also to the DRAM system memory and ROM devices. Communication between the 60X CPU and its I/O devices and system memory is managed by the 650 Bridge.

1.1 Summary of 650 Bridge Features

This section summarizes the features of the 650 Bridge—including the central arbiter, the memory controller, the PowerPC local bus, the PCI expansion bus, the address translation logic, the L2 cache, the ROM controller, and the interrupt and exception logic.

The 650 Bridge operates from 3.0V to 3.78V, allowing either 3.3V or 3.6V power sources.

1.1.1 60X Microprocessor Support

The 650 Bridge supports the PowerPC 601, 603, and 604 microprocessors as follows:

- PowerPC 601
  - Supports all 601 external clocking modes
  - Supports CPU bus speed up to 66MHz
- PowerPC 603
  - Supports all CPU clock multiplier modes except 1:1
  - Supports CPU bus speed up to 66MHz (Without 1:1 mode, 66MHz:66MHz is not allowed, 66MHz:33MHz is allowed, 80MHz:40MHz is allowed.)
  - Supports 64-bit mode of the 603 CPU
- PowerPC 604
  - Supports all CPU clock multiplier modes
  - Supports CPU bus speed up to 66MHz

1.1.2 Central Arbiter

- DRAM refresh support
- Prioritized arbitration among the following devices:
  1. DRAM refresh (highest priority)
  2. 60X CPU
  3. L2 write-back cache
  4. I/O bridge
  5. Five PCI masters (priority 5 to 9)
- Support for ISA bus masters when an ISA I/O bridge is installed on the PCI bus
- Operates CPU bus and PCI bus as a single-bus system
- Implements a fairness algorithm
- Has a 63-count PCI bus latency timer to prevent lockup due to inoperative PCI devices
- During idle periods, the PCI bus grant is parked on the 60X CPU

1.1.3 Memory Controller

- Supports memory operations for the PowerPC Architecture™
- Eight RAS outputs, eight CAS outputs, and two write-enable outputs
- The memory is eight-bytes wide (plus eight parity bits)
- Fast page-mode is supported

- Supports industry-standard 70ns SIMMs
  - Directly supports 168-pin eight-byte 8M, 16M, and 32M SIMMs
  - Supports 72-pin four-byte 4M, 8M, 16M, and 32M SIMMs
- Mixed use of 8M and 32M eight-byte SIMMs
- Memory configurations available from 8M to 256M
- Empty SIMM sockets are allowed at any position in the eight socket array
- Provides row-address and column-address multiplexing for SIMMs requiring:

  10, 11, or 12 row by 9 column
  10, 11, or 12 row by 10 column
  11 row by 11 column
  Combined 12 row by 10 column and 11 row by 11 column
- Non-interleaved memory access operation
- Memory refresh address counter
  - Auto-increment on every refresh cycle
  - Auto-wrap at end of page
  - Outputs multiplexed to memory address lines
- Burst-mode memory address generation logic
  - 32-byte CPU bursts to and from memory
  - Any length PCI burst to and from memory
- Generates even parity, one bit per byte
- Checks parity eight-bytes wide on all memory reads
- Little-endian and big-endian addressing modes
- ISA master to DRAM access
- Optimized Timing is as follows:
  - CPU to memory write hit or read hit at 66MHz—7-5-5-5 (CPU bus cycles)
  - CPU to memory write hit or read hit at 50MHz, 40MHz, and 33MHz—6-4-4-4
  - PCI to memory read hit at 33MHz—5-3-4-3 (PCI bus cycles)
  - PCI to memory write hit at 33MHz—5-4-4-4
  - See Table 5–10 for more details on memory timing

1.1.4 PowerPC Local Bus

- 64-bit CPU data bus
- 32-bit CPU address bus
- CPU can operate in big-endian or little-endian mode
- Logic to swap byte lanes and translate addresses for big-endian and little-endian modes
- Synchronous CPU bus speed support up to 66MHz
- PCI bus clock can be equal to or half the speed of CPU bus clock—up to 33MHz

1.1.5 PCI Expansion Bus

- 650 Bridge chip set (653 Buffer and 654 Controller) presents one load to PCI bus
- PCI bus frequency 20 MHz to 33 MHz (maximum of PCI 2.0 specification)
  - PCI bus frequency can be equal to or one-half the frequency of the CPU bus clock
- 32-bit multiplexed PCI address and data path
- Support for I/O Bus Bridge (ISA, EISA, MicroChannel)
- Support for ISA bus master access to system memory when an I/O bridge is installed
- PCI to DRAM access—with L1 and L2 cache snooping
- Supports all 60X to PCI transfers that do not cross a four-byte boundary

1.1.6 Address Translation Logic

- Support for memory mapping 60X address space into PCI spaces
  - PCI I/O reads and writes
  - PCI memory reads and writes
  - PCI configuration reads and writes
  - PCI interrupt acknowledge reads
- Support for reverse translation of PCI addresses for snoops and PCI to memory access
- Support for contiguous ISA I/O and non-contiguous ISA I/O mappings (non-contiguous I/O allows operating systems to memory-protect 32-byte blocks of ISA I/O space)
- Forces the PCI_AD bits[1:0] to 00b during the address phase of all PCI transactions except PCI I/O transactions
- Support for low-order address translation (unmunging) in little-endian mode
- Inputs for translation override

1.1.7 L2 Cache Support

- L2 write-through or write-back cache support
- Handshakes with IBM27-82681-66 PowerPC L2 Cache Controller
- Snoop cycles to CPU generated for PCI reads and writes of system memory
- Parity checking on read cycles
- Allows timing of burst read hits up to 3-1-1-1

1.1.8 System ROM Controller

- Supports up to 8M of 8-bit ROM, flash, or EPROM connected to PCI_AD lines
  - Conversion buffers support 8-bit to 64-bit conversion
- Logic for flash write
- Write lock-out support
- Single-beat (one-byte to eight-byte) read cycle
- Single-beat (one-byte) write cycle
- Pseudo burst-mode (32-byte) read cycle
- Approximately 1.7us read cycle time at 66MHz

1.1.9 Interrupt and Exception Logic

- Interrupt pass-through to CPU
- Non-maskable interrupt (NMI) support
- The following types of errors are reported:
  - CPU or PCI system memory read parity errors
  - Illegal transfers:

    The CPU attempts an illegal size, type, or alignment transfer
    A PCI device target aborts to the CPU
    A missing or unresponsive PCI device
    A PCI bus hangup condition
  - L2 cache parity errors
- Readable error address register
- Drives CPU data lines to all one-bits on out-of-range memory reads
- PCI configuration cycles return all one-bits when no device responds
- Retimes the soft reset input to meet the 60X specifications

Section 2
The PCI Bus and 60X CPU Background Review

The material in this section reviews the PCI local bus specifications and the 60X CPU features and functions. This material is intended for readers who are not familiar with the PCI specification or the operation of the 60X CPU.

2.1 The PCI Local Bus Review

This section provides a review of the operation of the PCI local bus. The PCI local bus standard defines a high-performance, 32-bit or 64-bit local bus with multiplexed address and data lines. The PCI local bus standard has been defined by the PCI SIG (Special Interest Group), a computer-industry standards group. The PCI local bus provides an interconnect mechanism between peripheral controllers, like graphics controllers and SCSI controllers, and host computer systems.

2.1.1 PCI Local Bus References

The *PCI Local Bus Specification, Production Version, Revision 2.0*, dated April 30, 1993 contains the detailed information necessary for a full understanding of the PCI bus standard. The 650 Bridge provides the signals that are necessary to interact with devices that conform with the PCI standard as described in the specification. Implementing a PowerPC to PCI system with the 650 Bridge requires a full understanding of the PCI standard.

2.1.2 PCI Local Bus Overview

The PCI bus can be either a 32-bit or a 64-bit multiplexed address/data bus implementation. The 650 Bridge is a 32-bit implementation. The 32-bit multiplexed address and data lines can encode addresses in the range of 0 to 4G (0000 0000h to FFFF FFFFh). During data phases, the 32-bit bus can transfer four bytes per phase. A PCI bus transaction consists of an address phase followed by one or more data phases.

The PCI bus can operate in single-beat or burst mode. The beginning address of a transfer can be followed by a variable number of consecutive 32-bit data words. Burst data transfer can occur at the rate of 32 bits per PCI bus clock cycle. The maximum PCI clock rate of 33MHz can support up to 132M bytes per second burst transfer rates.

2.1.3 PCI Signals

Table 2–1 shows the standard PCI signals that are interfaced directly with the 650 Bridge chip set. The column labeled PCI Signal Name contains the signal name that is used in the PCI standard document. The column labeled 650 Bridge Signal Name contains the 650 Bridge signal name for the PCI signal. The *PCI Local Bus Specification* document describes all the possible PCI signals.

The # symbol at the end of a signal name indicates that the active or asserted state of the signal occurs with a low voltage level. When the # symbol is not present after the signal name, the signal is asserted with a high voltage level.

The terms *assert* and *negate* are used extensively. The term *assert* indicates that a signal is active, regardless of whether that level is represented by a high or low voltage. The term *negate* means that a signal is inactive. The term *deasserted* is also used to indicate a signal that is negated.

Table 2–1. PCI Signals in the 650 Bridge

Family	PCI Signal Name	650 Bridge Signal Name	Description
Address/Data	AD[31:00]	PCI_AD[31:00]	Address and data bus, 32 bits multiplexed.
	C/BE[3:0]#	PCI_C/BE[3:0]#	C (bus command) and BE (byte enable) multiplexed lines. An address phase is a bus command, a data phase is BE.
	PAR	PCI_PAR	Parity bit for PCI_AD and PCI_C/BE# combined, even parity bit for the combination of 36 bits.
Arbitration	REQ#	IO_BRDG_REQ#	ISA or EISA PCI bus request, input to arbiter.
	GNT#	IO_BRDG_GNT#	ISA or EISA PCI bus grant, output from arbiter.
	REQ#	PCI_REQ[1:5]#	Five PCI bus request lines, input to arbiter.
	GNT#	PCI_GNT[1:5]#	Five PCI bus grant lines, output from arbiter.
Interface Control	FRAME#	PCI_FRAME#	PCI frame, asserted by the current master to indicate the beginning and duration of a bus access.
	TRDY#	PCI_TRDY#	PCI target ready, asserted by the target device to indicate its completion of the current data phase of a transaction.
	IRDY#	PCI_IRDY#	PCI initiator ready, asserted by the master device to indicate completion of the current data phase of a transaction. When PCI_TRDY# and PCI_IRDY# are asserted on the same bus clock cycle, the current data phase is complete.
	STOP#	PCI_STOP#	PCI stop, asserted by a target to stop a transaction.
	DEVSEL#	PCI_DEVSEL#	PCI device select, asserted by a device that claims the address range and bus command of a cycle on the PCI bus.
System	CLK	PCI_CLK	PCI clock, provides the timing for PCI transactions (up to 33MHz).
	RST#	RESET#	Reset, initializes PCI registers, signals, and sequencers.

2.1.4 PCI Masters and Targets

The PCI bus standard uses a master and target architecture. Master devices can gain control of the bus and then direct other devices to perform reads, writes, configuration operations, and other types of transactions. Masters on the PCI bus use dedicated REQ# and GNT# lines to gain control of the bus. Targets on the PCI bus do not use REQ# and GNT# lines. Targets are selected by a range of addresses within the various types of PCI transactions.

A PCI master device requests the bus by asserting its REQ# line. When the GNT# line for the requesting master device is asserted, the master device can then take control of the PCI bus to communicate with other master or target devices on the PCI bus. An arbiter (which the 650 Bridge provides) is necessary to manage the REQ# and GNT# activity on the PCI bus.

2.1.5 PCI Arbitration

Since there can be more than one PCI master device on the PCI bus, and since each master device has its own independent REQ# and GNT# lines, there must be an arbitration mechanism for any PCI bus implementation. The 650 Bridge incorporates arbitration logic for DRAM refresh cycles, the 60X CPU, the L2 cache, an I/O bridge (ISA, EISA, MicroChannel), and up to five other PCI master devices. The 650 Bridge arbitration logic ensures PCI latency requirements and allocates host and PCI bus accesses according to a priority and fairness algorithm.

2.1.6 Basic Transfer Control

After PCI_REQ# and PCI_GNT#, the fundamentals of all PCI data transfers are controlled with the following five signals:

1. PCI_FRAME#—which is asserted by the master to indicate the beginning and end of a PCI bus transaction.
2. PCI_DEVSEL#—PCI device select, when asserted, indicates that the device that is driving PCI_DEVSEL# has decoded its address as the target of the current address.
3. PCI_IRDY#—initiator ready, deasserted by the master to force wait states.
4. PCI_TRDY#—target ready, deasserted by the target to force wait states.
5. PCI_STOP#—PCI stop is asserted by a target to stop a transaction.

The PCI bus is idle when both PCI_FRAME# and PCI_IRDY# are deasserted. The first clock edge on which PCI_FRAME# is asserted is the *address phase*. The 32-bit address and 4-bit bus command code (PCI_C/BE[3:0]#) are asserted (see Section 2.1.7) on the PCI bus during the *address phase*.

Target devices have up to three PCI bus cycles after PCI_FRAME# is asserted to recognize an address and respond by asserting PCI_DEVSEL#. If no device asserts PCI_DEVSEL# within three clocks of PCI_FRAME# a device using *subtractive decode* can claim the transaction by asserting PCI_DEVSEL#. A PCI device that provides ISA, EISA, or MicroChannel bus logic usually uses subtractive decoding for device selection.

One or more *data phases* follow the address phase. The master is required to assert its IRDY# signal when it is ready to receive or when it is providing valid data. The target asserts its TRDY signal when it is ready to receive or when it is providing valid data. When PCI_TRDY# and PCI_IRDY# occur on the same bus cycle, the current data phase is concluded.

When the last data phase begins, the master deasserts PCI_FRAME# to indicate the last 32-bit transfer. For single-cycle transfers, PCI_FRAME# is deasserted on the first data phase.

2.1.7 PCI Bus Commands

During the address phase of a PCI transaction, the PCI_C/BE[3:0]# signals encode bus commands. (During the data phase on the PCI bus, the PCI_C/BE[3:0]# signals are byte enables for the four bytes on the PCI_AD[31:00] address/data bus.) Table 2–2 shows the 16 possible PCI bus commands. During the address phase on the PCI bus these bus commands determine the action that is to be taken by the target of the address phase.

Table 2–2. PCI Bus Commands

PCI_C/BE[3:0]#	Command Type
0000b	Interrupt Acknowledge
0001b	Special Cycle
0010b	I/O Read
0011b	I/O Write
0100b	Reserved
0101b	Reserved
0110b	Memory Read
0111b	Memory Write
1000b	Reserved
1001b	Reserved
1010b	Configuration Read
1011b	Configuration Write
1100b	Memory Read Multiple
1101b	Dual Address Cycle
1110b	Memory Read Line
1111b	Memory Write and Invalidate

2.1.8 Termination of PCI Cycles

PCI transactions can be terminated in the following non-standard ways:

- Master abort—The master deasserts FRAME# then deasserts IRDY#
- Target abort—The target asserts PCI_STOP# with PCI_DEVSEL# deasserted
- Target retry or disconnect—The target asserts both PCI_STOP# and PCI_DEVSEL#

2.2 PowerPC 60X CPU Review

This section provides a review of the operation of the 60X CPU as it relates to the 650 Bridge and the PCI local bus. The 650 Bridge links the PowerPC 601, 603, or 604 (60X) microprocessors to a 32-bit implementation of the PCI local bus and to system memory. The 650 Bridge connects to the 60X CPU host bus and communicates with the 60X CPU through the 60X CPU host bus transaction types.

2.2.1 601 CPU References

The *PowerPC 601 RISC Microprocessor User's Manual, MPR601UMU-02* and the *PowerPC 601 RISC Microprocessor Hardware Specifications, MPR601HSU-02* contain detailed information regarding the operational and electrical characteristics of the 601 microprocessor.

2.2.2 PowerPC 60X CPU Overview

The PowerPC 60X microprocessors implement the 32-bit portion of the PowerPC architecture, which provides 32-bit effective addresses, integer data types of 8-bits, 16-bits, and 32 bits, and 32-bit and 64 bit floating-point data types. The 60X CPU has a 32-bit address bus and a 64-bit data bus. The 60X CPU system interface protocol allows multiple masters to compete for the 60X CPU host bus, but the 650 Bridge implements a uni-processor topology. The 650 Bridge communicates with the 60X CPU in conformance with the system interface (host bus) protocol.

The 60X CPU is a register-oriented microprocessor. All computation and data manipulation commands are performed in the internal registers of the CPU. An internal (L1) cache minimizes memory reads and writes for frequently accessed system memory data.

The 60X CPUs are superscalar microprocessors that are capable of issuing and retiring multiple instructions per clock. Instructions can complete out of order, but the 60X CPUs make execution appear sequential. The 60X CPUs use three types of execution units—an integer unit (IU), a branch processing unit (BPU), and a floating-point unit (FPU). Most integer instructions execute in one clock cycle. The FPU is pipelined so that a single-precision multiply-add instruction can be issued every clock cycle. The BPU features static branch prediction and performs condition register (CR) look-ahead.

2.2.3 CPU to 650 Bridge Signals

Table 2–3 shows the 60X CPU signals that are interfaced directly with the 650 Bridge chip set. The column labeled User's Manual Signal Name contains the signal name that is used in the *PowerPC 601 RISC Microprocessor User's Manual.* The column labeled 650 Bridge Signal Name contains the 650 Bridge signal names that are used for the 60X CPU signals.

See Section 2.1.3 for a description of the signal naming conventions used in this document.

Table 2–3. 60X CPU Signals Connected to the 650 Bridge

Family	User's Manual Signal Name	650 Bridge Signal Name	Description
Arbiter	BR#	CPU_REQ#	Bus request from the 60X CPU. This signal indicates that the 60X CPU wants control of the host bus.
	BG#	CPU_GNT#	Bus grant to the 60X CPU. The 650 Bridge arbiter grants control of the host bus to the 60X CPU with this signal. The arbiter parks the host bus on the 60X CPU (asserts this signal) when no other masters are requesting a bus.
Transfer Attributes	TT[0:3]	TT[0:3]	60X CPU bus transfer type. TT[4] is not used in the 650 Bridge and should be negated—pulled low. See Table 2–4 for TT[0:3] codes.
	TSIZ[0:2]	TSIZ[0:2]	60X CPU bus transfer size—number of bytes. The 650 Bridge supports transfers of 1, 2, 3, 4, 8, and 32 bytes (1, 2, 3, or 4 on the PCI bus).
	TBST#	TBST#	Indicates a burst transfer of four 8-byte double words on the 60X CPU bus. Burst transfers by the 60X CPU are only allowed to system memory, not to the PCI bus.

Table 2–3. 60X CPU Signals Connected to the 650 Bridge (Continued)

Family	User's Manual Signal Name	650 Bridge Signal Name	Description
Address Transfer Start	TS#	TS#	60X CPU bus transfer start. TS# is asserted by the current bus master when the address on the CPU_ADDR[31:00] lines is valid.
	XATS#	XATS#	Extended address transfer start. When asserted, this signal indicates that the 60X CPU is performing I/O controller interface operations (PIO). If the T-bit in the 60X CPU segment register is set it indicates that addresses in the range of the segment register are I/O controller interface accesses. The 650 Bridge does not support PIO operations from the 60X CPU. If this signal is asserted, the 650 Bridge generates an error to the 60X CPU with TEA#.
Address	A[0:31]	CPU_ ADDR[00:31]	60X CPU address bus. Of the 32 bits on the bus, the 654 Controller decodes 60X CPU addresses using CPU_ADDR[00:08], [19], [29:31], where [31] is the least-significant bit. By analyzing these bits, the 654 Controller can determine if the CPU is addressing system memory, PCI memory, PCI I/O space, PCI interrupt acknowledge space, system ROM, or PCI configuration space. The 653 Buffer uses all 32 bits of the CPU_ADDR address lines.
Address Termination	AACK#	AACK#	60X CPU bus address acknowledge.
	ARTRY#	ARTRY#	60X CPU bus address retry. This signal can indicate that the 60X CPU or the L2 cache has detected a condition where a snooped address must be retried. ARTRY# is also asserted by the 650 Bridge during target-retry terminations on the PCI bus.
Data Transfer	DH[0:31]	CPU_ DATA[0:31]	32 bits of the 64-bit 60X CPU data bus. 0 = most-significant bit.
	DL[0:31]	CPU_ DATA[32:63]	32 bits of the 64-bit 60X CPU data bus. 63 = least-significant bit.
	DPE#	DPE#	Data Parity Error from 60X CPU. The 60X CPU asserts DPE# two CPU bus clocks after each TA# if data parity is invalid.
Data Termination	TA#	TA#	60X CPU bus transfer acknowledge. The 650 Bridge asserts TA# to indicate that a data transfer has completed successfully. The 650 Bridge asserts TA# for one CPU clock for a single beat data transfer transaction. For a four-beat burst transaction, the 650 Bridge asserts TA# at the conclusion of each of the four data transfer phases.
	TEA#	TEA#	60X CPU bus transfer error acknowledge. The 650 Bridge asserts this signal instead of TA# to terminate 60X CPU bus cycles and report error conditions such as—an NMI (non-maskable interrupt), memory access out-of-range error, transfer type error, transfer size error, PCI target abort, or a parity error from system memory.
60X CPU Control	INT#	INT_CPU#	60X CPU Interrupt. Asserted by the 650 Bridge to signal the 60X CPU to run an interrupt cycle.
	SRESET#	SRESET_ CPU#	This signal is a soft reset—the 60X CPU warm boots when SRESET_CPU# is asserted. SRESET_CPU# is asserted by asserting SRESET_REQ# on the 654 Controller chip.

2.2.4 Cache (L1)

The 60X CPUs contain an L1 cache that can implement a write-back policy. The caches in the three 60X CPUs operate differently. The following discussion relates only to the 601 CPU.

The 601 CPU contains a 32-Kbyte, eight-way set-associative, unified (instruction and data) cache. The cache sector size is eight 32-bit words within a cache line of 64 bytes. The cache is designed to conform to a write-back policy, but the 60X CPUs allow control of cacheability, write policy, and memory coherency at the page and block level. The cache uses a least-recently-used (LRU) replacement policy.

The instruction unit provides the cache with the address of the next instruction to be fetched. In the case of a cache hit, the cache returns the requested instruction and as many of the instructions following it as can be placed in the eight-word instruction queue up to the cache sector boundary. If the queue is empty, as many as eight 32-bit words can be loaded into the queue in parallel.

The cache tag directory has one address port dedicated to instruction fetch and load/store accesses and one port dedicated to snooping transactions on the system interface (host bus). Therefore snooping does not require extra clock cycles unless a snoop hit that requires a cache status update occurs.

2.2.5 System Interface

The 60X CPU system interface (host bus) has 64 data lines and 32 address lines plus various control lines. An external arbiter (the 650 Bridge) controls access to the host bus. The arbiter can grant host bus access to the 60X CPU when memory reads or writes are requested. When PCI transactions are being processed, the arbiter can direct the host bus to perform snoops of addresses on the host bus.

Because the 60X CPU has an L1 cache, the predominant type of transaction for most applications is burst memory operations, where four beats of 64-bit double words are transferred after a single address phase. These bursts are linear within a cache sector. That is, wherever the burst begins within the 32-byte cache sector, it fills the 32-byte sector in a circular fashion.

2.2.6 TT[0:3] (Transfer Type)

The TT[0:3] signals are transfer type codes for the 60X CPU host bus. These signals are asserted with TS# (transfer start) and the CPU_ADDR[00:31] address lines at the beginning of a bus transaction on the 60X CPU host bus. Table 2–4 shows the 16 possible transfer type codes and their 60X CPU descriptions.

Table 2–4. TT[0:3]—Transfer Type Codes

TT[0:3]	60X Bus Mnemonic
0000b	Clean sector
0001b	Write with flush
0010b	Flush sector
0011b	Write with kill
0100b	Sync
0101b	Read
0110b	Kill sector

Table 2–4. TT[0:3]—Transfer Type Codes (Continued)

TT[0:3]	60X Bus Mnemonic
0111b	Read with intent to modify
1000b	— (Reserved)
1001b	Write with flush atomic
1010b	External control out (ecowx)—not supported
1011b	— (Reserved)
1100b	TLB invalidate
1101b	Read atomic
1110b	External control in (eciwx)—not supported
1111b	Read with intent to modify atomic

2.2.7 Pipelining and Split Transactions

The 60X address and data buses can be independent to support pipelining and split transactions; however, the 650 Bridge does not use pipelining and split transactions. During 60X host bus burst transactions, AACK# is asserted by the 650 Bridge with the last TA# in order to keep the 60X CPU from pipelining addresses.

2.2.8 Big-Endian and Little-Endian Modes of Operation

The 60X CPU powers up and resets in big-endian (BE) mode. In this mode of operation, the most-significant byte of any data format using multiple bytes is the lowest numbered byte. For example, a four-byte integer with the value 01020304h is stored at address 1000h with the 01h in address 1000h and the 02h in address 1001h, etc. In little-endian mode the 01h of the previous example is stored at 1003h, the 02h is stored at 1002h, etc.

Table 2–5 shows examples of big-endian and little-endian data storage. In the case of the integer and short integer fields, the bytes are byte reversed. However, a string is stored in the same fashion in both big-endian and little-endian modes because strings are treated as individual bytes and there is no difference in big-endian and little-endian addressing at the byte level.

Table 2–5. Big-Endian and Little-Endian Data Storage

	Big-Endian Data Format								Little-Endian Data Format							
Data description	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**
4-byte integer—01020304h	01	02	03	04					04	03	02	01				
2-byte short integer—1122h	11	22							22	11						
String—ABCDEF	A	B	C	D	E	F			A	B	C	D	E	F		
A single byte—31h			31								31					
A single byte—66h							66								66	
8-byte double word—1122334455667788h	11	22	33	44	55	66	77	88	88	77	66	55	44	33	22	11

In little-endian mode, instruction fetches use byte-swapping logic within the 32-bit instruction. Byte 0 becomes byte 3, byte 1 becomes byte 2, and so on. The four bytes within a 32-bit instruction word are reversed.

When it is in big-endian mode, the 60X CPU addresses system memory as if it is organized with big-endian byte significance. However, when it is in little-endian mode, the 60X CPU addresses system memory in an intermediate form that is neither big-endian nor little-endian. The 650 Bridge translates the addressing that the 60X CPU uses in little-endian mode to produce a true little-endian memory mapping. The 650 Bridge provides the address manipulation and byte swapping that is required to support system memory as exactly little-endian representation in little-endian mode and exactly big-endian in big-endian mode. Therefore, data is maintained in system memory by the 650 Bridge in the same endian order as the media on which it resides, hard disk for example.

The 60X CPU defaults to big-endian mode at power-on or reset. The 650 Bridge defaults to the setting of the LE_MODE_REQ# signal at power-on or reset. The endian mode can be switched as required.

2.2.9 PIO or I/O Controller Operation (XATS#)

The 650 Bridge does not support the PIO I/O controller interface bus protocol. PIO operations on the 60X CPU host bus are indicated by the assertion of the XATS# signal. The 650 Bridge responds to the assertion of XATS# with TEA# to indicate a host bus transfer error.

Section 3
650 Bridge Pin Descriptions

Figure 3–1 shows how the 653 Buffer and 654 Controller are connected to the 60X CPU bus, the PCI bus, DRAM, ROM, the L2 cache, and external logic. Figure 3–1 also shows the interconnections between the 653 Buffer and the 654 Controller. The following tables describe groups of signals that connect to the 653 Buffer and 654 Controller.

The tables in Section 3.2 describe the pins that connect the 653 Buffer with the 60X CPU, the PCI bus, system memory (DRAM), and external logic. The tables in Section 3.2 describe the pins in the 654 Controller. The tables in Section 3.3 describe the pins that interconnect the 653 Buffer and the 654 Controller.

Section 8.1 and Section 8.2 contain numeric and alphabetic lists of the pins in the 653 Buffer and 654 Controller respectively. Pin numbers are included in these tables.

3.1 653 Buffer Pin Descriptions

The # symbol at the end of a signal name indicates that the active or asserted state of the signal occurs with a low voltage level. When the # symbol is not present after the signal name, the signal is asserted with a high voltage level.

The terms *asserted* and *negated* are used extensively. The term *asserted* indicates that a signal is active, regardless of whether that level is represented by a high or low voltage. The term *negated* means that a signal is inactive. The term *deasserted* is also used to indicate a signal that is negated.

The following terms are used to describe the signal type:

in	Input is a standard input-only signal.
out	Output is a standard active driver
I/O	Bi-directional

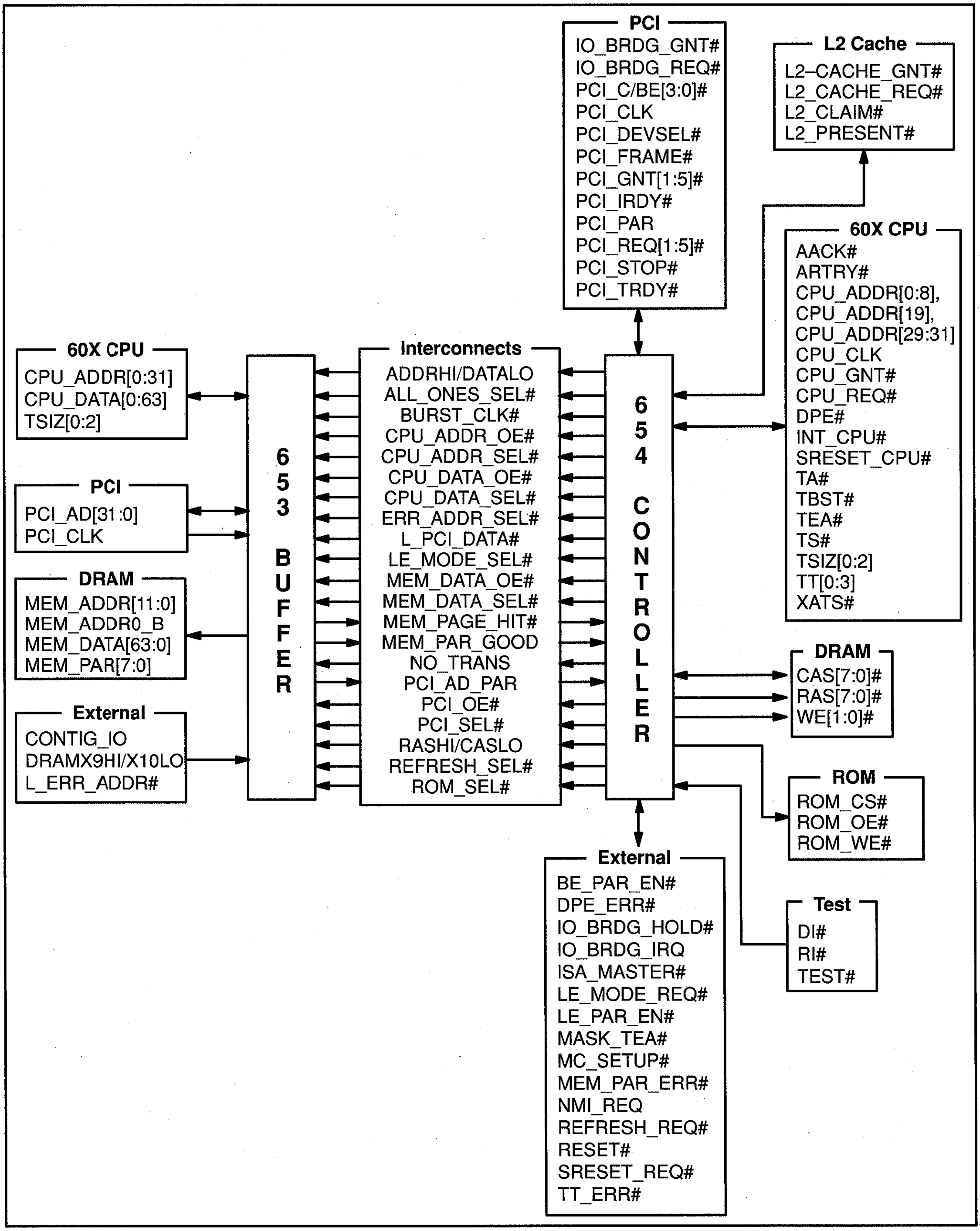


Figure 3–1. 650 Bridge Pin Connections

3.1.1 653 Buffer to 60X CPU Bus Interface Signals

Table 3–1 describes the signals that interface the 653 Buffer to the 60X CPU bus.

Table 3–1. 653 Buffer to 60X CPU Bus Interface

Signal Name	Type	Description
CPU_ADDR[0:31]	I/O	The 60X CPU address bus, bit 0 = most-significant bit (MSB). All buses connected to the 650 Bridge, except the 60X buses, use little-endian nomenclature.
CPU_DATA[0:63]	I/O	The 64-bit 60X CPU data bus, bit 0 = MSB. CPU_DATA[0:31] connect to the 60X CPU signals DH[0:31]. CPU_DATA[32:63] connect to the 60X CPU signals DL[0:31].
TSIZ[0:2]	in	60X CPU bus transfer size—number of bytes. The 650 Bridge supports transfers of 1, 2, 3, 4, 8, and 32 bytes (1, 2, 3, or 4 to the PCI bus).

3.1.2 653 Buffer to PCI Bus Interface Signals

Table 3–2 describes the signals that interface the 653 Buffer to the PCI bus.

Table 3–2. 653 Buffer to PCI Bus Interface

Signal Name	Type	Description
PCI_AD[31:0]	I/O	PCI address and data bus. The 32-bit PCI_AD bus is a multiplexed address and data bus. The PCI_AD bus is numbered in little-endian order.
PCI_CLK	in	PCI clock. The PCI clock signal—up to 33MHz. The rising edge of the PCI_CLK signal at the 653 Buffer must be synchronized to the rising edge of CPU_CLK within -1.0ns to +1.0ns. The PCI_CLK can be the same frequency or half the frequency of the CPU clock. The PCI clock at the 653 Buffer may or may not be the same physical signal line as the PCI clock at the 654 Controller. See Section 7.2 for clocking details.

3.1.3 653 Buffer to System Memory Interface Signals

Table 3–3 describes the signals that interface the 653 Buffer to system memory.

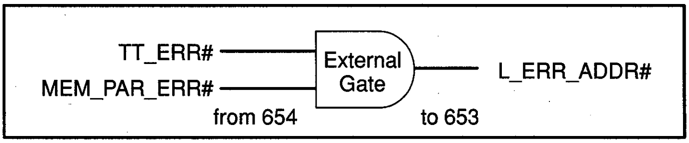
Table 3–3. 653 Buffer to System Memory Interface

Signal Name	Type	Description
MEM_ADDR[11:0]	out	Memory address bus. MEM_ADDR is 12 bits, multiplexed, and little-endian. While the 654 Controller asserts RASHI/CASLO high, the MEM_ADDR lines contain row addresses selected from the internal address bus of the 653 Buffer. While RASHI/CASLO is low, the MEM_ADDR lines contain the column addresses.
MEM_ADDR0_B	out	A duplicate of MEM_ADDR[0]. (Required by some SIMMs.)
MEM_DATA[63:0]	I/O	64-bit memory data bus. MEM_DATA[63] is the most significant bit.
MEM_PAR[7:0]	I/O	8-bit memory parity bus. MEM_PAR[7] is the most significant bit. Bit 7 corresponds to MEM_DATA[63:56]. Even parity is generated and written on memory write cycles. System memory is always read in eight-byte double words, regardless of the transfer size requested. Parity across eight bytes is checked on all memory read cycles.

3.1.4 653 Buffer to External Logic and System Interface Signals

Table 3–4 describes the signals that are used to interface the 653 Buffer to the rest of the system via external logic, command bit storage elements, the test interface, power, and ground.

Table 3–4. 653 Buffer to External Logic and System Interface

Signal Name	Type	Description
CONTIG_IO	in	Contiguous I/O. External logic asserts CONTIG_IO high to enable direct mapping of addresses from 2G to 2G + 8M. When CONTIG_IO is driven low, it enables non-contiguous addressing in the 2G to 2G + 8M address range. Non-contiguous I/O is a mapping of the low 32 bytes of each 4k page of CPU memory space to 32 bytes of PCI/ISA I/O space. See Section 4.1.1 and 4.1.2. This signal should only be changed between 60X to PCI I/O cycles.
DRAMX9HI/X10LO	in	DRAM type. DRAMX9HI/X10LO is asserted high for addressing DRAMs with nine column address bits (X9 mode), low for X10 mode. This signal is used to format the addresses presented to the DRAMs.
L_ERR_ADDR#	in	Latch error address. The address on the 653 Buffer internal address bus is latched into the 653 error address register while L_ERR_ADDR# is asserted. This signal can be derived by external logic from the 654 Controller signals TT_ERR#, MEM_PAR_ERR# and, optionally, any other signal indicating an error condition requiring the address to be latched. L_ERR_ADDR# must be held asserted to hold the contents of the latch. Any signal used with TT_ERR# and MEM_PAR_ERR# to derive L_ERR_ADDR# must also be held until after the latch is read. This diagram illustrates support logic needed to latch the address of memory parity errors or transfer type errors.  TT_ERR# MEM_PAR_ERR# External Gate L_ERR_ADDR# from 654 to 653
TEST#	in	Test mode. Pull to logic high during normal operation. Assert TEST#, L_ERR_ADDR#, and ERR_ADDR_SEL# to tri-state the outputs.

3.2 654 Controller Pin Descriptions

The following tables describe the signals connected to the 654 Controller chip.

See Section 3.1 for a description of the signal naming conventions used in this document.

The following terms are used to describe the signal type:

in	Input is a standard input-only signal.
out	Output is a standard active driver
I/O	Bi-directional
s/o/d	Sustained open drain input/output
t/s	Tri-state is a bi-directional, tri-state input/output signal
s/t/s	Sustained tri-state is an active low tri-state signal owned and driven by one agent at a time. The agent that drives the s/t/s pin low must drive it high for at least one clock before letting it float. A new agent cannot drive the pin any sooner than one clock after the previous owner tri-states it. An external pull-up is required to sustain the inactive state.

3.2.1 654 Controller to 60X CPU Bus Interface Signals

Table 3–5 shows the 654 Controller signals that communicate with the 60X CPU.

Table 3–5. 654 Controller to 60X CPU Bus Interface

654 Controller Signal Name	Signal Type	Description
AACK#	I/O	CPU bus address acknowledge. During host bus transactions initiated by the 60X CPU, the 654 Controller asserts AACK# with the last TA# (transfer acknowledge). This prevents the 60X CPU from pipelining addresses. During a transaction initiated by the 60X CPU on the PCI bus or during PCI to system memory transactions, AACK# can be asserted on target initiated terminations and on cache snoop hits. See the discussion of these events in the ARTRY# description.
ARTRY#	I/O	CPU bus address retry. During a transaction initiated by the 60X CPU on the PCI bus, the PCI target can assert PCI_STOP# to the 654 Controller (target initiated termination). If PCI_STOP# and PCI_DEVSEL# are asserted together by the target (a target retry), the 654 Controller then asserts AACK# on one CPU clock followed by ARTRY# on the next CPU clock. This sequence allows the 60X CPU to retry the transaction. If PCI_STOP# is asserted without PCI_DEVSEL# (a target abort), the 654 Controller asserts TEA# to signal a PCI target abort and does not assert ARTRY#. During PCI to system memory cycles, the 654 Controller asserts TS# on the second CPU clock after PCI_FRAME# is asserted on the PCI bus. Then the 654 Controller asserts AACK# on the CPU clock after TS#, initiating L1 and L2 cache snoops of the address that has been set on the 60X CPU host bus by the 653 Buffer. Both the 60X CPU and a write-back L2 can assert ARTRY# on the CPU clock immediately after AACK# to signal a snoop hit. When the 654 Controller senses ARTRY# asserted on the CPU clock following AACK#, it asserts the PCI_STOP# signal and the PCI_DEVSEL# signal on the next PCI bus clock to signal a target retry. The PCI device must back off the PCI bus (and retry), and the 654 Controller then grants the 60X CPU host bus to the 60X CPU or the L2 cache for a writeback to system memory. The 654 Controller arbiter resolves simultaneous cache hits in the L1 and L2 caches. (The write-back cache uses a protocol that guarantees coherency.)
CPU_ADDR[0:8], [19], [29:31]	in	CPU address bus. Of the 32 bits on the CPU address bus, the 654 Controller uses CPU_ADDR[0:8], [19], [29:31], where [0] is the most significant bit. By analyzing these bits, the 654 Controller can determine if the CPU is addressing system memory, PCI memory, PCI I/O space, PCI interrupt acknowledge space, the error address register, system ROM, or PCI configuration space. The 654 Controller also monitors changes in the state of CPU_ADDR[19] (driven by the 653 Buffer) to detect the crossing of a DRAM page during PCI bursts to memory. The 653 Buffer uses all 32 bits of the CPU_ADDR address lines.
CPU_CLK	in	CPU bus clock. 40MHz, 50MHz, or 66MHz.
CPU_GNT#	out	CPU bus grant. Bus grant to the 60X CPU, grants control of the host bus to the 60X CPU. The arbiter parks the host bus on the 60X CPU (CPU_GNT# is asserted regardless of CPU_REQ#) when no other masters are requesting a bus.
CPU_REQ#	in	CPU bus request. Bus request from the 60X CPU, indicates that the 60X CPU wants control of the host bus. The 654 Controller arbiter controls the host bus grant through CPU_GNT#.
DPE#	in	Data parity error. The 654 Controller checks DPE# two clocks after each TA# when an L2 cache asserts L2_CLAIM# and provides read data. If DPE# is active, the 654 Controller asserts DPE_ERR# low for two clocks. L2 error addresses are not saved.

Table 3–5. 654 Controller to 60X CPU Bus Interface (Continued)

654 Controller Signal Name	Signal Type	Description
INT_CPU#	out	CPU Interrupt. INT_CPU# is asserted to signal the processor of an external interrupt or under some error conditions like memory or L2 cache parity errors.
SRESET_CPU#	out	During normal mode the SRESET_CPU# signal is a soft reset—the 60X CPU warm boots when SRESET_CPU# is asserted. The 654 Controller asserts SRESET_CPU# when SRESET_REQ# activates. The 654 Controller asserts SRESET_CPU# for two cycles of REFRESH_REQ# (at least 15us) to ensure the required minimum assertion time for SRESET_CPU# to the 60X CPU (10 CPU clocks).
TA#	I/O	CPU bus transfer acknowledge. The 654 Controller asserts TA# to indicate that a data transfer has completed successfully. The 654 Controller asserts TA# for one CPU clock for a single beat data transfer transaction. For a four-beat burst transaction, the 654 Controller asserts TA# at the conclusion of each of the four data transfer cycles.
TBST#	I/O	Transfer burst. TBST# indicates a burst transfer of four 64-bit double-words on the 60X CPU bus. Burst transfers by the 60X CPU are only allowed to system memory, not to the PCI bus. The 650 Bridge does support PCI burst reads and writes of system memory by PCI devices. The 654 Controller drives TBST# inactive during PCI to memory snoop cycles.
TEA#	out	CPU bus transfer error acknowledge. If MASK_TEA# is not asserted, the 654 Controller asserts TEA# for a PCI bus error, a non-maskable interrupt (NMI), transfer type error, transfer size error, or a parity error from system memory. See Section 5.8. If XATS# is asserted for a PIO cycle, the 654 Controller asserts TEA# regardless of the condition of MASK_TEA#.
TS#	I/O	CPU bus transfer start. TS# is asserted by the current bus master when the address on the CPU_ADDR[00:31] lines and the address attribute lines are valid. During PCI memory cycles, the 654 Controller asserts TS# to start snoop cycles on the second CPU clock after PCI_FRAME# is asserted on the PCI bus and then asserts AACK# on the next CPU clock. This initiates L1 and L2 cache snooping of the address of PCI memory cycles. See ARTRY# for an explanation of the snoop hit process. Note: All PCI memory transactions produce snoop cycles on the 60X CPU bus. The memory model for the 650 Bridge only allows system memory to be mapped in the 0 to 2G range, therefore snoops from 2G to 4G are never cache hits.
TSIZ[0:2]	I/O	CPU bus transfer size—number of bytes. The 650 Bridge supports transfers of 1, 2, 3, 4, 8, and 32 bytes. The 654 Controller asserts TEA# for unaligned transfers of 2, 3, or 4 bytes that cross a double-word boundary, and also asserts TEA# for attempted transfers of 5, 6, or 7 bytes. See Section 5.8. TSIZ[0:2] is ignored when TBST# is asserted for 32-byte bursts.
TT[0:3]	I/O	CPU bus transfer type. TT[4] is not used and should be negated—pulled low.
XATS#	in	Extended address transfer start. When asserted this signal indicates that the 60X CPU is performing I/O controller interface (PIO) operations. If the T-bit in a 60X CPU segment register is set, it indicates that addresses in the range of that segment register are I/O controller interface accesses. If XATS# is asserted, the 654 Controller generates a TEA# error to the 60X CPU (regardless of the setting of MASK_TEA#).

3.2.2 654 Controller to PCI Bus Interface Signals

Table 3–6 shows 654 Controller signals that are related to the operation of the PCI bus devices.

Table 3–6. 654 Controller to PCI Bus Interface

654 Controller Signal Name	Signal Type	Description
IO_BRDG_GNT#	out	I/O bridge bus grant. System bus grant to the I/O bridge.
IO_BRDG_REQ#	in	I/O bridge request. PCI bus request line from the I/O bridge subsystem. The I/O bridge request has the highest priority of the PCI initiators. (The arbiter grants the bus to DRAM refresh requests, the 60X CPU, and the L2 cache before any PCI initiators.)
PCI_C/BE[3:0]#	t/s	C (bus command) and BE (byte enable) multiplexed lines. During a PCI address phase this is a bus command. During a PCI data phase PCI_C/BE[3:0]# are byte enables—one bit for each of the four bytes on the PCI bus.
PCI_CLK	in	PCI bus clock. When the CPU bus clock is 40, 50, or 66MHz, PCI_CLK is one-half the frequency of the CPU bus clock. (The maximum PCI bus clock is 33MHz.) When the CPU bus clock is 33MHz or 25MHz, PCI_CLK is equal to the CPU bus clock. The 654 Controller PCI_CLK must be synchronous to CPU_CLK within –0.5 to +4.0 nsecs. The PCI clock at the 653 Buffer may or may not be the same physical signal line as the PCI clock at the 654 Controller. See Section 7.2 for clocking details.
PCI_DEVSEL#	s/t/s	PCI device select. PCI_DEVSEL# is asserted by a device to claim a PCI address. PCI_DEVSEL# can go active within three PCI clocks of PCI_FRAME# (or four if subtractive decode is used). The 650 Bridge asserts PCI_DEVSEL# when it claims a PCI memory cycle—the address is within physical memory and there is no hit in L1 or L2 cache.
PCI_FRAME#	s/t/s	PCI frame. Asserted by the current master to indicate the beginning and duration of a PCI bus access.
PCI_GNT[1:5]#	out	PCI bus grants. Five PCI bus grant lines corresponding to PCI_REQ[1:5]#.
PCI_IRDY#	s/t/s	PCI initiator ready. PCI_IRDY# indicates the bus master is ready to complete the current data phase of a transaction. A data phase is complete on any PCI clock where both PCI_IRDY# and PCI_TRDY# are asserted. During a write, PCI_IRDY# indicates that the master has placed valid data on the PCI_AD bus. During a read PCI_IRDY# indicates that the master is prepared to accept data. During 60X to PCI cycles, this signal is generated independent of the state of PCI_TRDY#.
PCI_PAR	t/s	PCI parity. PCI bus parity bit for PCI_AD[31:00] and PCI_C/BE[3:0]# combined, an even parity bit for the 36 bits of PCI_AD[31:0] and PCI_C/BE[3:0]#. (This bit plus the thirty-six PCI bits equals an even number of bits.) The 653 Buffer calculates the parity for PCI_AD[31:0] and passes it to the 654 Controller as PCI_AD_PAR. When the 650 drives the PCI_AD and PCI_C/BE lines on PCI cycles, the 654 combines PCI_AD_PAR with PCI_C/BE[3:0] to generate PCI_PAR. The 650 Bridge does not check PCI_PAR on incoming transactions.
PCI_REQ[1:5]#	in	PCI bus requests. Five PCI bus request lines. PCI_REQ[1] is the highest priority among the five request lines.

Table 3–6. 654 Controller to PCI Bus Interface (Continued)

654 Controller Signal Name	Signal Type	Description
PCI_STOP#	s/t/s	PCI stop. A PCI target uses this signal to end the current transaction with target retry or target abort. For target retry, PCI_STOP# and PCI_DEVSEL# are asserted together. For target abort, PCI_STOP# is asserted by itself. After a target retry, the initiator can retry the cycle at a later time. The 654 Controller asserts PCI_STOP# and PCI_DEVSEL# (target retry) for a snoop hit on a PCI to system memory transaction to allow a cache write-back operation.
PCI_TRDY#	s/t/s	PCI target ready. A PCI target uses PCI_TRDY# to signal that it is ready for data transfer. A PCI transaction ends when PCI_TRDY# and PCI_IRDY# are asserted together. During a PCI read from system memory, the 654 Controller asserts PCI_TRDY# when memory data is valid. During a PCI write to system memory, the 654 Controller waits for PCI_IRDY# to be asserted and then asserts PCI_TRDY# when it has completed the write cycle.

3.2.3 654 Controller to System Memory (DRAM) Interface Signals

Table 3–7 shows the 654 Controller signals that communicate with the DRAM memory chips.

Table 3–7. 654 Controller to System Memory (DRAM) Interface

654 Controller Signal Name	Signal Type	Description
CAS[7:0]#	I/O	Column address selects. The CAS[7:0]# lines are also used to read and write system logic data during the setup of the 650 Bridge.
RAS[7:0]#	out	Row address selects.
WE[1:0]#	out	DRAM write enables. Two identical signals to meet loading requirements.

3.2.4 654 Controller to ROM (Flash or EPROM) Signals

Table 3–8 shows 654 Controller signals that are related to the operation of the system ROM.

Table 3–8. 654 Controller to ROM or Flash Signals

654 Controller Signal Name	Signal Type	Description
ROM_CS#	out	ROM chip select.
ROM_OE#	out	ROM output enable.
ROM_WE#	out	ROM write enable. Write enables for flash ROM.

3.2.5 654 Controller to L2 Cache Signals

Table 3–9 shows the 654 Controller signals that communicate with the optional L2 cache.

Table 3–9. 654 Controller to L2 Cache Signals

654 Controller Signal Name	Signal Type	Description
L2_CACHE_GNT#	out	L2 cache bus grant. The 654 Controller asserts L2_CACHE_GNT# to grant the 60X bus to the L2 cache for a write-back operation. Note that since the 650 Bridge is designed as a single-bus system, when the L2 is doing a write-back operation, the PCI bus remains idle (parked on the 654 Controller).
L2_CACHE_REQ#	in	L2 cache bus request. Asserted by a write-back L2 cache to perform a write-back operation. This signal is not used by write-through cache designs.
L2_CLAIM#	in	L2 claim. The L2 cache asserts L2_claim# to indicate a read or write hit in the L2 cache. The 650 Bridge backs off of system memory and lets the L2 cache provide or receive data when L2_CLAIM# is asserted. The cache must assert L2_CLAIM# by the second clock period after that in which TS# is asserted, and L2_CLAIM# must be held until AACK# is asserted. A write-through L2 cache only asserts L2_CLAIM# on read hits. A write-back L2 cache can assert L2_CLAIM# on read and write hits.
L2_PRESENT#	in	L2 cache present. Must be continuously asserted to indicate that an L2 cache is present in the system.

3.2.6 654 Controller to Test Signals

Table 3–10 shows the 654 Controller test signals.

Table 3–10. 654 Controller to Test Signals

654 Controller Signal Name	Signal Type	Description
DI#	in	Driver inhibit. (Pull to logic high during normal system operation.) Assert DI# with TEST# to tri-state the outputs.
RI#	in	Receiver inhibit. (Pull to logic high during normal system operation.) Assert RI# to gate off inputs while in test mode.
TEST#	in	Test mode. (Pull to logic high during normal system operation.)Assert TEST# with DI# to tri-state the outputs.

3.2.7 654 Controller to External Logic and System Interface Signals

Table 3–11 shows 654 Controller signals that are connected to host system devices.

Table 3–11. 654 Controller to External Logic and System Interface

654 Controller Signal Name	Signal Type	Description
BE_PAR_EN#	out	Big-endian parity enable. The 654 Controller asserts BE_PAR_EN# when reading memory in big-endian mode. The system can use this signal to enable an external buffer to route big-endian-ordered parity to the 60X data parity signals. (See LE_PAR_EN#.)
DPE_ERR#	out	Data parity error. DPE_ERR# is a qualified data parity error for L2 cache parity errors. It is a pulse two CPU clocks wide. External logic must latch this signal.

Table 3–11. 654 Controller to External Logic and System Interface (Continued)

654 Controller Signal Name	Signal Type	Description
IO_BRDG_HOLD#	in	I/O bridge hold. I/O bridge memory operation request from an ISA bus device. When this signal is asserted, the 654 arbiter will not remove grant from the I/O bridge subsystem unless PCI_FRAME# is asserted or the IO_BRDG_REQ# is removed. IO_BRDG_HOLD# is an asynchronous input.
IO_BRDG_IRQ	in	Interrupt from I/O bridge subsystem. This signal is passed through to the CPU as an interrupt on INT_CPU#. As a result of the assertion of INT_CPU#, the 60X should request a PCI interrupt acknowledge transaction to which the 654 Controller will respond by running a PCI interrupt acknowledge cycle to read an interrupt vector.
ISA_MASTER#	in	ISA master. In response to ISA_MASTER#, the 654 Controller asserts NO_TRANS to the 653 Buffer to allow PCI transactions from ISA masters to directly address system memory from 0 to 16M without the usual address remapping. This function supports ISA bridges that do not allow remapping of ISA address space within PCI address space. ISA_MASTER# is an asynchronous input.
LE_MODE_REQ#	in	Little-endian mode request. External logic asserts and holds LE_MODE_REQ# to request the selection of little-endian mode. In response, the 654 Controller asserts LE_MODE_SEL# to the 653 Buffer to enable little-endian addressing. LE_MODE_SEL# is changed only when the buses are not busy. Note: LE_MODE_REQ# and LE_MODE_SEL# are external to the CPU and are not the same as the internal CPU endian mode bit. LE_MODE_REQ# is an asynchronous input.
LE_PAR_EN#	out	Little-endian parity enable. The 654 Controller asserts LE_PAR_EN# when reading memory in little-endian mode. The system can use this signal to enable an external buffer to route little-endian-ordered parity to the 60X data parity signals. (See BE_PAR_EN#.)
MASK_TEA#	in	Mask TEA#. When external logic asserts MASK_TEA#, all 60X CPU host bus cycles (except XATS# cycles) terminate with TA#, regardless of error conditions. MASK_TEA# can be used for diagnostic purposes. If XATS# is asserted for PIO operations, a TEA# error is always asserted, regardless of the setting of MASK_TEA#. MASK_TEA# is an asynchronous input.
MC_SETUP#	in	Memory controller setup. External logic asserts MC_SETUP# to select setup of the controller registers through the CAS[7:0]# lines. Read and write operations are supported. Data should be gated to or from the CAS# lines when MC_SETUP# is active. MC_SETUP# is an asynchronous input.
MEM_PAR_ERR#	out	Memory parity error. The 654 Controller asserts MEM_PAR_ERR# to indicate that a qualified memory parity error was detected. MEM_PAR_ERR# remains asserted until the conclusion of a 60X bus cycle which the bridge decodes as a read error address. MEM_PAR_ERR# is speculatively asserted during CPU or PCI reads of system memory. The MEM_PAR_ERR# signal becomes valid one CPU clock after TA# is sampled valid or one PCI clock after TRDY# is sampled valid.

Table 3–11. 654 Controller to External Logic and System Interface (Continued)

654 Controller Signal Name	Signal Type	Description
NMI_REQ	in	Non-maskable interrupt request. The 60X CPU does not have a non-maskable interrupt. However, the 654 Controller asserts INT_CPU# when NMI_REQ is asserted. As a result of the assertion of INT_CPU#, the 60X software should issue a byte load instruction at the specific address that the 654 Controller decodes a request for a PCI interrupt acknowledge cycle. The 654 Controller responds by asserting TEA# instead of running a PCI interrupt acknowledge cycle, and the 654 returns FFh as the result of the byte load. NMI_REQ is an asynchronous input.
REFRESH_REQ#	in	DRAM refresh request. External logic asserts REFRESH_REQ# to request a DRAM refresh cycle. The 654 arbiter treats REFRESH_REQ# as the highest priority bus request. The 654 asserts REFRESH_SEL# to the 653 Buffer when the bus is available. REFRESH_REQ# is an asynchronous input.
RESET#	in	System reset. Power good when deasserted (high), power-on-reset (POR) condition when asserted. RESET# must be held low for at least 10us after power is stable and clocks are running normally. RESET# is an asynchronous input.
SRESET_REQ#	in	Soft reset request (warm boot). The 654 Controller asserts SRESET_CPU# to the 60X CPU in response to SRESET_REQ#. To guarantee the minimum assertion time, the 654 Controller asserts SRESET_CPU# for two assertions of REFRESH_REQ#. SRESET_REQ# should be a pulse of from 100ns minimum to 4ms maximum. SRESET_CPU# is an asynchronous input.
TT_ERR#	out	Transfer type error. The 654 Controller asserts TT_ERR# if an unsupported transfer type or alignment is detected or if XATS# is asserted. TT_ERR# remains asserted until the conclusion of a 60X bus cycle which the bridge decodes as a read error address.

3.3 Signals Between the 653 Buffer and 654 Controller

Table 3–12 shows the signals that interconnect the 654 Controller with the 653 Buffer.

Table 3–12. Signals Between the 653 Buffer and the 654 Controller

654 Controller Signal Name	653 Type	654 Type	Description
ADDRHI/DATALO	in	out	Address high/data low. The 654 Controller asserts ADDRHI/DATALO (high) to put the 653 Buffer in a PCI address cycle and negates ADDRHI/DATALO low to put the 653 Buffer in a PCI data cycle. (The PCI_AD[31:00] bus is a multiplexed address and data bus.) This signal has two uses—during 60X CPU initiated cycles to the PCI bus, the negation transition (signalling the end of the address tenure) occurs exactly one PCI clock cycle earlier than when the data tenure begins. The 653 Buffer delays driving data to the PCI_AD bus for one PCI cycle. During PCI-initiated cycles to system memory, the 653 Buffer latches the PCI_AD bus as an address on each PCI_CLK rising edge while ADDRHI/DATALO remains asserted high. When ADDRHI/DATALO is negated, the last address remains in the PCI address latch in the 653 Buffer.

Table 3–12. Signals Between the 653 Buffer and the 654 Controller (Continued)

654 Controller Signal Name	653 Type	654 Type	Description
ALL_ONES_SEL#	in	out	All-ones select, asserted by the 654 Controller to the 653 Buffer to place all one-bits on the 653 Buffer internal data bus. ALL_ONES_SEL# is used during PCI configuration read transactions to return 64 one-bits to the CPU data bus when no PCI device responds and during system memory reads that are out-of-range.
BURST_CLK#	in	out	Burst clock. Based on other control signals, BURST_CLK# clocks the shifts in the ROM shift register, clocks the ROM burst counter, clocks the PCI burst counter during PCI master cycles, or clocks the CPU burst counter (all within the 653 Buffer).
CPU_ADDR_OE#	in	out	CPU address output enable. The 654 Controller asserts CPU_ADDR_OE# to enable the 653 Buffer to assert PCI-initiated addresses on the 60X CPU address bus. The 654 Controller asserts TS# when the address is valid, allowing the L1 and L2 caches to snoop memory cycles.
CPU_ADDR_SEL#	in	out	CPU address select. The 654 Controller asserts CPU_ADDR_SEL# to enable the 653 Buffer to receive addresses from the 60X bus. The 654 Controller asserts this signal during power-on-reset (POR). After power up, this signal must be asserted and deasserted by the 654 Controller before any bus cycles are initiated. This initializes the CPU burst counter within the 653 Buffer.
CPU_DATA_OE#	in	out	CPU data output enable. The 654 Controller asserts CPU_DATA_OE# during CPU read cycles to enable the 653 Buffer to assert data onto the 60X bus.
CPU_DATA_SEL#	in	out	CPU data select. The 654 Controller asserts CPU_DATA_SEL# during CPU write cycles to enable the 653 Buffer to receive data (byte-swapped in little-endian mode) from the 60X bus.
ERR_ADDR_SEL#	in	out	Error address select. The 654 Controller asserts ERR_ADDR_SEL# to enable the 653 Buffer to drive a 32-bit error address from the error address register onto both halves of the 64-bit 60X CPU data bus. MEM_PAR_ERR# and TT_ERR# are deactivated coincidently with the rising edge of this signal.
L_PCI_DATA#	in	out	Latch PCI data. While L_PCI_DATA# is not asserted and the PCI_CLK is low, the PCI data latch is transparent to the PCI_AD bus. When data is required from the PCI bus (during a CPU to PCI read or a PCI bus master to system memory write), the 654 Controller asserts this signal following the rising edge of the PCI_CLK for the current data phase. This latches the current data phase data into the PCI data latch. The 4-byte data is then placed on both halves of the 8-byte 653 Buffer internal data bus.
LE_MODE_SEL#	in	out	Little-endian mode select. In response to LE_MODE_REQ#, the 654 Controller asserts LE_MODE_SEL# to set the 653 Buffer to little-endian mode of operation. This signal is switched between bus cycles.
MEM_DATA_OE#	in	out	Memory data output enable. While MEM_DATA_OE# is asserted, the 653 Buffer drives the 64-bit internal data bus and its eight parity signals onto the memory data bus and the memory parity bus. MEM_DATA_OE# is asserted by the 654 Controller during memory write cycles.
MEM_DATA_SEL#	in	out	Memory data select. MEM_DATA_SEL# is asserted by the 654 Controller during a memory read transaction. When MEM_DATA_SEL# is asserted, the 653 Buffer uses the memory data bus as the source for the current transaction.

Table 3–12. Signals Between the 653 Buffer and the 654 Controller (Continued)

654 Controller Signal Name	653 Type	654 Type	Description
MEM_PAGE_HIT#	out	in	Memory page hit. The 653 Buffer asserts MEM_PAGE_HIT# to indicate an equal compare on the RAS address. This signal is valid one CPU_CLK after the assertion of TS#. MEM_PAGE_HIT# is not asserted for forced page hits that occur after refresh cycles and PCI I/O cycles. The 650 Bridge internally sets a forced page condition for these situations.
MEM_PAR_GOOD	out	in	Memory parity good. Negated by the 653 Buffer to indicate a parity error on a read of system memory. This is an unqualified decode of the 64 memory data lines. The 654 Controller samples this line appropriately.
NO_TRANS	in	out	No translation mode. The 654 Controller asserts NO_TRANS high when a memory read or write cycle runs on the PCI bus on behalf of an ISA master. NO_TRANS disables the address remapping within the 653 Buffer so that ISA masters that cannot remap the 0 to 16M address range can directly access memory in the 0 to 16M address range. When NO_TRANS is asserted, all address remapping in the 653 Buffer is disabled.
PCI_AD_PAR	out	in	PCI address and data parity. The 653 Buffer generates an even parity bit across the PCI_AD[31:0] lines. This is an unqualified signal that is only valid when the PCI_AD bus is valid. The 654 Controller combines PCI_AD_PAR with PCI_C/BE[3:0] to generate PCI_PAR, the PCI parity bit.
PCI_OE#	in	out	PCI output enable. While PCI_OE# is asserted, the 653 Buffer drives the internal address or data buses onto the PCI_AD bus. PCI_OE# is asserted whenever the CPU or L2 is the bus master except during the data phase of reads from the PCI. See the ADDRHI/DATALO signal.
PCI_SEL#	in	out	PCI select. The 654 Controller asserts PCI_SEL# to enable the 653 Buffer to receive addresses and data from the PCI bus during PCI master cycles to system memory or CPU reads from the PCI. PCI_SEL# is asserted for the duration of the cycle.
RASHI/CASLO	in	out	RAS or CAS select. The 654 Controller asserts RASHI/CASLO high for a RAS cycle and negates RASHI/CASLO low for a CAS cycle. This signal is asserted during any PCI I/O and configuration cycles, following a DRAM memory page miss, and during DRAM refresh cycles.
REFRESH_SEL#	in	out	DRAM refresh selection. After REFRESH_REQ# is asserted externally, the 654 Controller asserts REFRESH_SEL# as soon as the current CPU or PCI bus cycle concludes in order to initiate a refresh cycle. In response to REFRESH_SEL#, the 653 Buffer places a refresh address on the memory address bus. The 653 Buffer increments its internal row address on the rising edge of this signal.
ROM_SEL#	in	out	ROM select. The 654 Controller asserts ROM_SEL# to signal the 653 Buffer that a ROM cycle is in progress.

Table 4–1. 650 Bridge Mapping of 60X CPU Bus Addresses

60X CPU Address Range	Other Conditions	Target Cycle Decoded	Target Cycle Address Range	Comment
0 to 2G		System Memory	0 to 2G	Cacheable by L1 and L2
2G to 2G + 8M	CONTIG_IO deasserted	PCI I/O Cycle (ISA, EISA, or MicroChannel)	0 to 64K (64K to 8M not accessible)	Non-contiguous I/O. 32 bytes of each 4K memory page in this 8M address space are mapped to 32 bytes in the 64K PC I/O space. See Section 4.1.1 and Section 4.1.2
	CONTIG_IO asserted	PCI I/O Cycle (ISA, EISA, or MicroChannel)	0 to 8M	Contiguous I/O. CONTIG_IO is a pin on the 653 Buffer chip. See Section 4.1.1 and 4.1.2
2G + 8M to 2G + 16M		PCI Configuration Cycle	8M to 16M	PCI_AD[1:0] forced to 00b
2G + 16M to 3G – 8M		PCI I/O Cycle	16M to 1G – 8M	PCI_AD[1:0] flow through
3G – 8M to 3G	CPU_ADDR[19] = 0	Read Error Address Register	None	No PCI cycle
	CPU_ADDR[19] = 1	PCI Interrupt Acknowledge	1G – 8M to 1G	PCI_AD[1:0] forced to 00b
3G to 4G – 8M		PCI Memory Cycle	0 to 1G – 8M	PCI_AD[1:0] forced to 00b Note: CPU space 4G – 16M to 4G – 8M is reserved in PowerPC Reference Platform Specification.
4G – 8M to 4G	Read cycle	ROM Read	1G – 8M to 1G (0 to 8M in ROM)	System ROM (Can be EPROM, EEPROM, or flash ROM)
	Write cycle (CPU_ADDR[31] = 0)	ROM Write Port	N/A	Flash ROM write port (address coded in data field)
	Write cycle (CPU_ADDR[31] = 1)	Flash ROM Lock-Out Port	N/A	Write to this range of addresses locks out flash ROM writes until RESET# (No PCI cycle)

Section 4
650 Bridge Theory of Operation

This section describes the theory of operation of the 650 Bridge. This section includes basic timing diagrams with narrative descriptions.

Section 4.1 describes the memory and device mapping that the 650 Bridge applies to 60X bus addresses. The 60X CPU can address system memory (DRAM), PCI devices, and other functions by loading (reading) or storing (writing) data to specific address ranges.

Section 4.2 describes the mapping the 650 Bridge applies to memory reads and writes initiated by PCI devices.

Section 4.3 contains timing diagrams and descriptions of basic cycles that can be generated by the 60X CPU and PCI devices. Section 7 contains comprehensive, detailed timing diagrams.

4.1 650 Bridge Mapping of 60X CPU Bus Addresses

The 650 Bridge maps the 60X address space as shown in Table 4–1. The 654 Controller decodes the nine most-significant bits of the 60X bus address (CPU_ADDR[0:8]) to determine the basic type of transaction the 60X CPU is requesting.

In general, the range of the CPU_ADDR[0:8] signals serves to identify the target of a 60X CPU transaction. Other conditions can modify the type of cycle within an address range as follows:

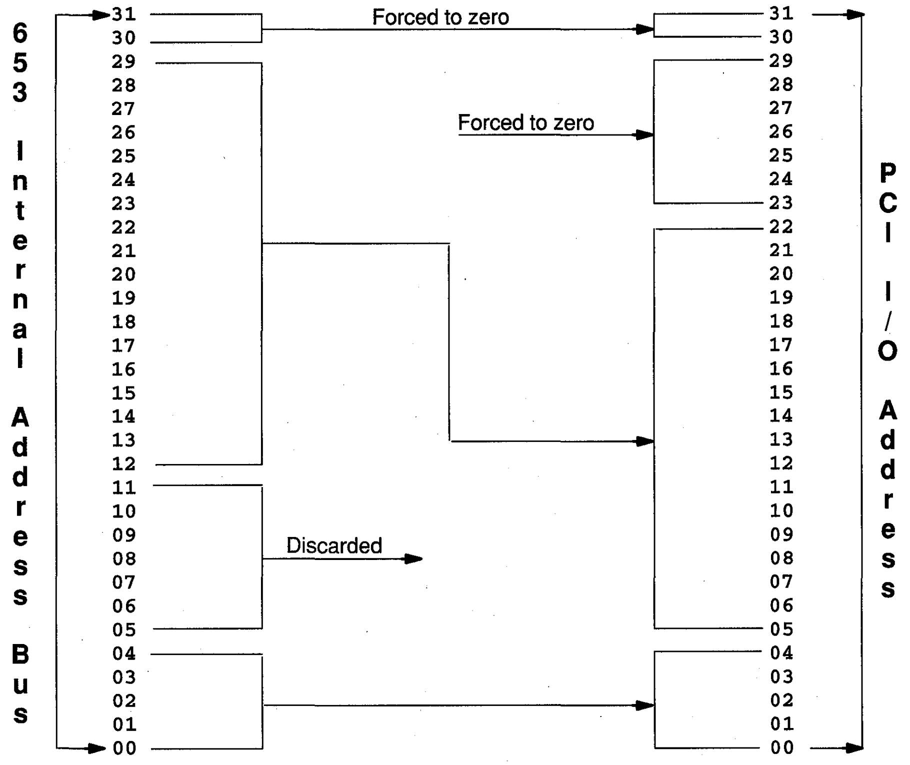
- External logic asserts or deasserts the CONTIG_IO signal to select contiguous or non-contiguous PCI I/O addressing in the first 8M of PCI I/O addresses. Sections 4.1.1 and 4.1.2 discuss non-contiguous and contiguous PCI I/O addressing.
- For mapping purposes, CPU_ADDR[19] is used to differentiate between PCI interrupt acknowledge transactions and error address register read transactions.
- For mapping purposes, CPU_ADDR[31] determines whether a ROM write cycle is addressed to the ROM write port or the ROM write lock-out port.

4.1.1 Address Mapping for Non-Contiguous I/O

Figure 4–1 illustrates the address mapping the 650 Bridge performs in non-contiguous mode (CONTIG_IO is deasserted) for addresses from 2G to 2G + 8M. After the 60X bus address reaches the internal bus of the 653 Buffer, the 0 to 8M addresss space is compressed into 64K of PCI addresses from 0 to 64K.

If LE_MODE_SEL# is asserted, CPU_ADDR[29:31] are unmunged when they reach [2:0] on the 653 internal address bus. Note that the 653 internal bus is numbered in little-endian order. Munging and unmunging are described in Section 5.3 along with other endian-related operations.

If they are not claimed by a PCI agent, all PCI I/O transactions with PCI addresses from 0 to 64K are claimed by the I/O bridge. In non-contiguous I/O mode the 60X CPU cannot create PCI I/O addresses from 64K to 8M.



A31 to A30 are forced to 00b. A29 to A12 are shifted to A22-A5. A11 to A5 are discarded. A4 to A0 pass through unchanged. (On the input side A2 to A0 are unmunged in LE mode.) A29 to A23 are set to zero.

Figure 4–1. Non-Contiguous PCI I/O Address Transformation

In non–contiguous I/O mode, the 650 Bridge partitions the 2G to 2G + 8M address space so that the first 32 bytes of each 4K page are remapped into the 0 to 64K ISA port address space. Therefore, 60X CPU protection attributes can be assigned to any of the 4K pages. This provides a flexible mechanism to lock the I/O from change by user-state code. This partitioning spreads the ISA I/O address locations over 8M of 60X CPU address space.

In non-contiguous mode, the unused byte addresses within each 4K page are not available. Each of the 32 contiguous port addresses in each 4K page has the same protection attributes in the 60X CPU.

For example, 60X CPU addresses 8000 0000h to 8000 001Fh are converted to I/O bridge port addresses 0000h through 001Fh. I/O bridge port 0020h starts in the next 4K page at 60X CPU address 8000 1000h.

4.1.2 Address Mapping for Contiguous I/O

In contiguous I/O mode (CONTIG_IO asserted), a 60X CPU address from 2G to 2G + 8M causes a PCI I/O cycle to run on the PCI bus with PCI_AD[29:00] unchanged except for the unmunging of the three low-order address bits. If not claimed by another PCI agent, the addresses from 0 to 64K may be claimed by the I/O bridge.

4.1.3 PCI Final Address Formation

The 650 Bridge maps 60X bus addresses from 2G to 4G as PCI transactions, error address register reads, or ROM reads and writes. The 650 Bridge manipulates 60X bus addresses from 2G to 4G to generate PCI addresses as follows:

- PCI_AD[31:30] are set to zero.
- PCI_AD[2:0] are unmunged if LE_MODE_SEL# is asserted. See Section 5.3.
- After unmunging, PCI_AD[1:0] are set to 00b for all PCI cycles except PCI I/O cycles.

4.2 650 Bridge Mapping of PCI Device Addresses

Table 4–2 shows the mapping of memory read and write cycles from the PCI bus to system memory and PCI memory. Of the transactions that can be initiated by PCI masters, the 650 Bridge only recognizes PCI memory reads and writes. The 650 Bridge ignores PCI I/O, PCI configuration, and PCI interrupt acknowledge transactions that are initiated by PCI devices on the PCI bus.

The 650 Bridge broadcasts the remapped address of all PCI memory read and write transactions so that they can be snooped by the L1 and L2 caches. By definition, the PowerPC Reference Platform Specification maps system memory only from 0 to 2G, therefore memory addresses from 2G to 4G must not be cacheable and will not cause snoop hits even though they are broadcast.

When the ISA_MASTER# signal is asserted, the 650 Bridge maps PCI memory reads and writes from 0 to 16M directly to system memory at 0 to 16M. If the ISA_MASTER# signal is not asserted, PCI memory reads and writes in this address range are ignored by the 650 Bridge.

The 60X CPU can generate PCI memory reads and writes in the range of 0 to 1G – 8M. The best range of addresses to locate PCI memory that can be addressed by both PCI devices and the 60X CPU is from 16M to 1G – 16M.

Table 4–2. 650 Bridge Mapping of PCI Device Addresses

PCI Cycles And Addresses		650 Bridge Maps as:		
Type Of PCI Cycle	**PCI Bus Address**	**Target Of Address**	**60X Bus Address**	**Comments**
Memory	0 to 16M	System Memory	0 to 16M	ISA masters only (ISA_MASTER# asserted). Snooped.
		PCI Memory	0 to 16M	ISA_MASTER# not asserted. Bridge ignores except for snoop.
	16M to 1G – 16M	PCI Memory	2G + 16M to 3G – 16M	Bridge ignores except for snoop.
	1G – 16M to 1G – 8M	Reserved	3G – 16M to 3G – 8M	Architecture reserves this area. Bridge ignores except for snoop.
	1G – 8M to 2G	Unavailable	3G – 8M to 4G	650 Bridge cannot create these memory addresses on the PCI bus. Do not map PCI memory here unless the 60X CPU will never access it. Bridge ignores except for snoop.
	2G to 4G	System Memory	0 to 2G	Snooped.

4.3 650 Bridge Bus Transactions

The following timing diagrams show examples of bus transactions in a 601 system where the CPU bus clock (CPU_CLK) is running at 66MHz, synchronous to and in phase with the 601 internal clock (P_CLOCK).

The CPU_REQ# signal is not generally shown in the timing diagrams. When the bus is not being used by another device or being requested by a higher priority device, the 650 Bridge arbiter responds to the CPU_REQ# signal from the 60X CPU by asserting CPU_GNT#.

Unless shown separately, TSIZ[0:2], TT[0:4], and TBST# are asserted and negated with CPU_ADDR.

Section 7 contains detailed timing diagrams and timing conventions. Section 7 also contains timing diagrams for many different varieties of the basic transactions shown in this section.

Final determination of the exact operation of the 650 Bridge should be made from the detailed timing diagrams in Section 7.

4.3.1 CPU to Memory Read—Single-Beat, Page Hit, XCAS = 0

Figure 4–2 shows a single-beat system memory (DRAM) read with a page hit and XCAS = 0. (See Section 5.2.2.2.) The 60X CPU can initiate a read of system memory by executing a *load* instruction with an address range of 0G to 2G. See Table 4–1. The 650 Bridge arbiter sends CPU_GNT# low in cycle 0 to grant the bus to the 60X CPU. The state of this signal during the rest of this transfer has no effect on this transfer, and so is shown as unknown. Likewise, the state of TBST#, the CPU_ADDR group (address and attributes), TS#, MEM_PAGE_HIT#, and MEM_ADDR are not initially known.

In cycle 1, the 60X CPU asserts TS#, CPU_ADDR[0:31], TT[0:3], and TSIZ[0:2], and drives TBST# inactive. In response, the 650 Bridge evaluates the address, transfer type, and TBST# signal to determine that the CPU is requesting a single-beat read of system memory.

During cycle 2, the 653 Buffer asserts MEM_PAGE_HIT# to indicate that the row address of the memory read matches the previous row address. As a result of this signal, the 654 Controller leaves RASHI/CASLO low, therefore RAS# stays low and the 650 Bridge does not update the row address in the DRAM. Also during cycle 2, the 654 Controller asserts CPU_ADDR_SEL# to select the CPU address for use during this transfer. This address is processed inside the 653 Buffer, and propagates through to the memory controller, which selects the column address and drives it onto the MEM_ADDR lines during cycle 3. AACK# and TA# stay tri-stated until cycle 4 to avoid contentions with an L2 cache in the event of a cache hit.

During cycle 4, the 654 Controller asserts the CAS[7:0]# lines to begin a CAS# read access, asserts MEM_DATA_SEL# to select the memory data bus as the source of the data for this transfer, and asserts CPU_DATA_OE# to enable the 653 Buffer to drive the data onto the CPU data bus. The 654 Controller also asserts BE_PAR_EN# (if the system is in big-endian mode) or LE_PAR_EN# (in little-endian mode) to enable one of the external parity buffers.

The MEM_DATA signals become valid and propagate through the 653 Buffer to the CPU_DATA lines. Since XCAS = 0, the CAS# lines are asserted for a total of three CPU_CLK cycles. They are active for two cycles before TA# is asserted, then the 654 Controller asserts TA# and AACK# during cycle 6, and the 60X completes the transaction. The 654 then negates the remaining control outputs.

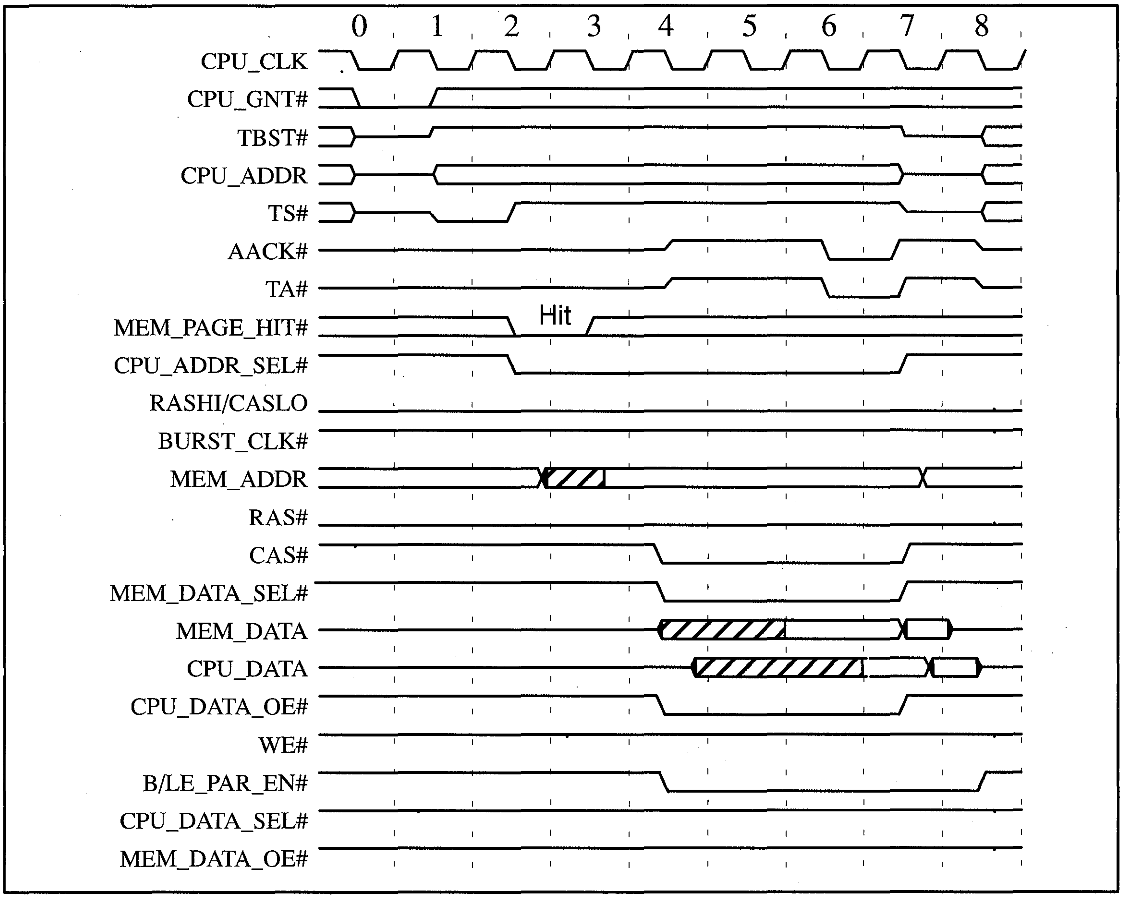


Figure 4–2. CPU to Memory Read, Single-Beat, Page Hit, XCAS = 0 Timing Diagram

4.3.2 CPU to Memory Read—Single-Beat, Page Hit, XCAS = 1

Figure 4–3 shows a single-beat system memory (DRAM) read with a page hit and XCAS = 1. (See Section 5.2.2.2.) This transfer is identical to the XCAS = 0 transfer shown in Figure 4–2 during cycles 0 through 5. Since XCAS = 1, the CAS[7:0]# signals are extended for one clock and all subsequent signals occur one clock later. This could also be thought of as adding an additional fifth cycle to the CAS# read access.

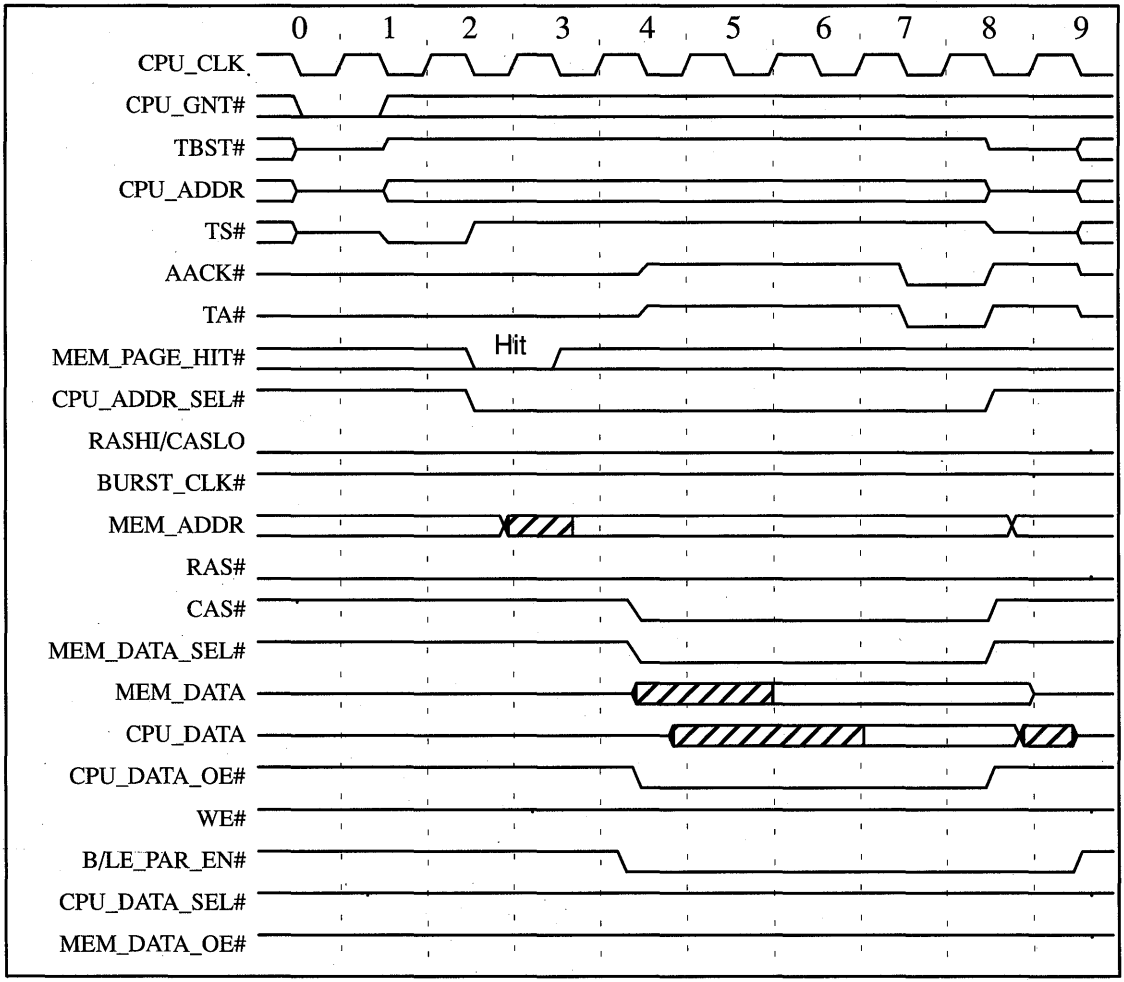


Figure 4–3. CPU to Memory Read, Single-Beat, Page Hit, XCAS = 1 Timing Diagram

4.3.3 CPU to Memory Read—Single-Beat, Page Miss, XCAS = 1

Figure 4–4 shows a single-beat system memory read with a page miss and XCAS equals 1. This transfer is identical to the page hit transfer shown in Figure 4–3 during cycles 0 through 2. During cycle 2, the 653 Buffer negates MEM_PAGE_HIT# to indicate that the row address of the memory read does not match the previous row address.

Beginning in cycle 3, the 650 Bridge inserts a RAS# access before the CAS# read access that begins in cycle 4 during a page hit. The 654 Controller sends RASHI/CASLO high to cause the 653 Buffer to select the row address to drive onto the MEM_ADDR lines. The 654 Controller sends RAS# high for 4 cycles, and then asserts a RAS# line to latch the row address into the DRAM.

In cycle 8, the 654 Controller sends RASHI/CASLO low to cause the 653 Buffer to drive the column address onto the MEM_ADDR lines. The address propagates to the DRAM during cycle 9.

In cycle 10, the 654 Controller asserts the selected CAS# lines to begin the CAS# read access. The rest of the transfer follows the page hit timing found in Figure 4–3.

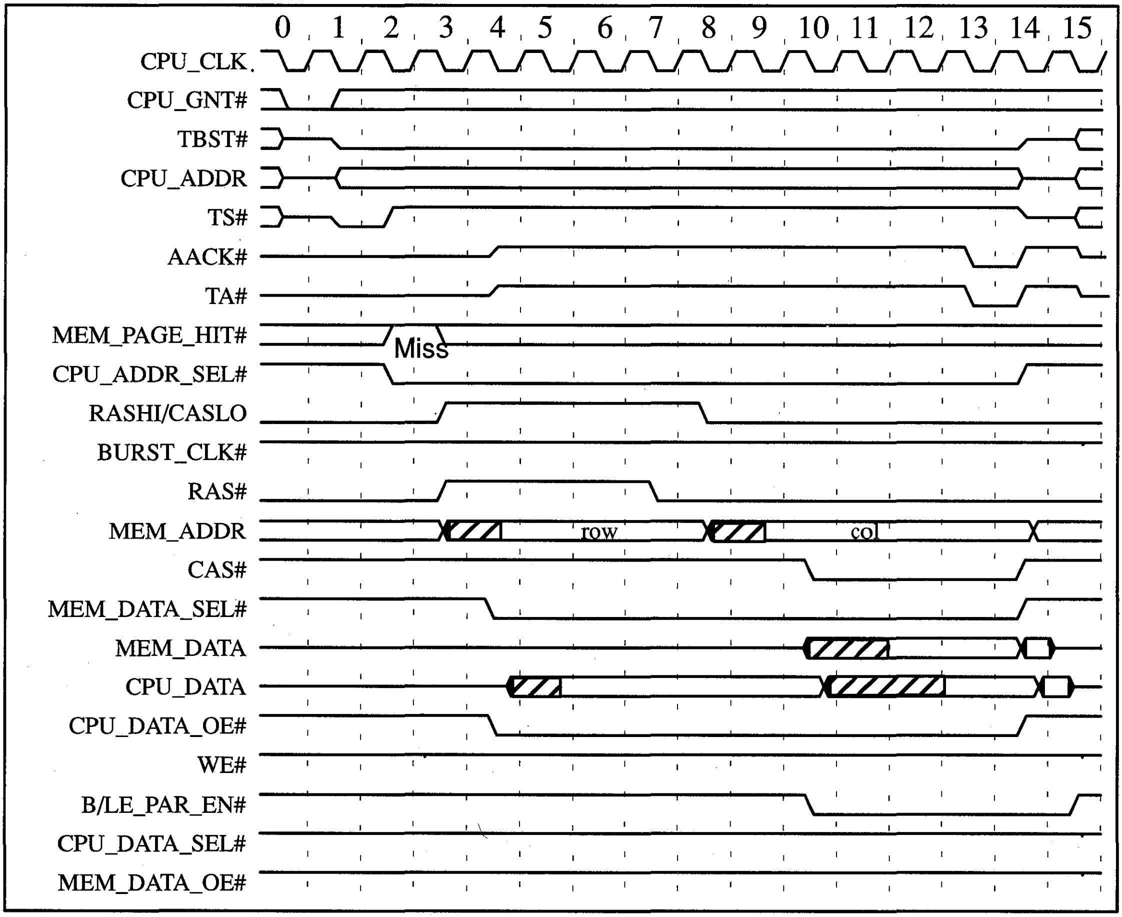


Figure 4–4. CPU to Memory Read, Single-Beat, Page Miss, XCAS = 1 Timing Diagram

4.3.4 CPU to Memory Read—Burst, Page Miss, XCAS = 1

Figure 4–5 shows a burst read from system memory with an initial page miss and XCAS = 1. The operations performed by the 650 Bridge are identical to those in Figure 4–4 during cycles 0 through 11. The 654 Controller leaves the data and address path control signals (CPU_ADDR_SEL#, MEM_DATA_SEL#, and CPU_DATA_OE#) asserted, effectively stretching the transfer to accomodate the three extra beats of the burst.

In cycle 12, the 654 Controller begins a burst read, which is executed three times, once each for beats 1, 2, and 3, of the 4-beat burst (the beats are numbered 0 through 3). In this read, the 654 Controller asserts BURST_CLK# for one cycle to increment the column address presented to the DRAM (via the CPU Address Counter in the 653 Buffer) from the beat 0 (initial) address to the beat 1 address. At this time, the DRAM data from the beat 0 column address is still becoming valid on the MEM_DATA lines and propagating through the 653 Buffer to the CPU_DATA lines.

In cycle 13 the 654 Controller asserts TA# for one cycle, and the 60X CPU latches in the data. In cycle 14, the 654 Controller deasserts CAS# for one cycle. As CAS# is sent low again, the beat 1 address is latched into the DRAM, and the burst cycle begins over again.

Cycles 26 through 30 are identical to Figure 4–4 cycles 11 through 15.

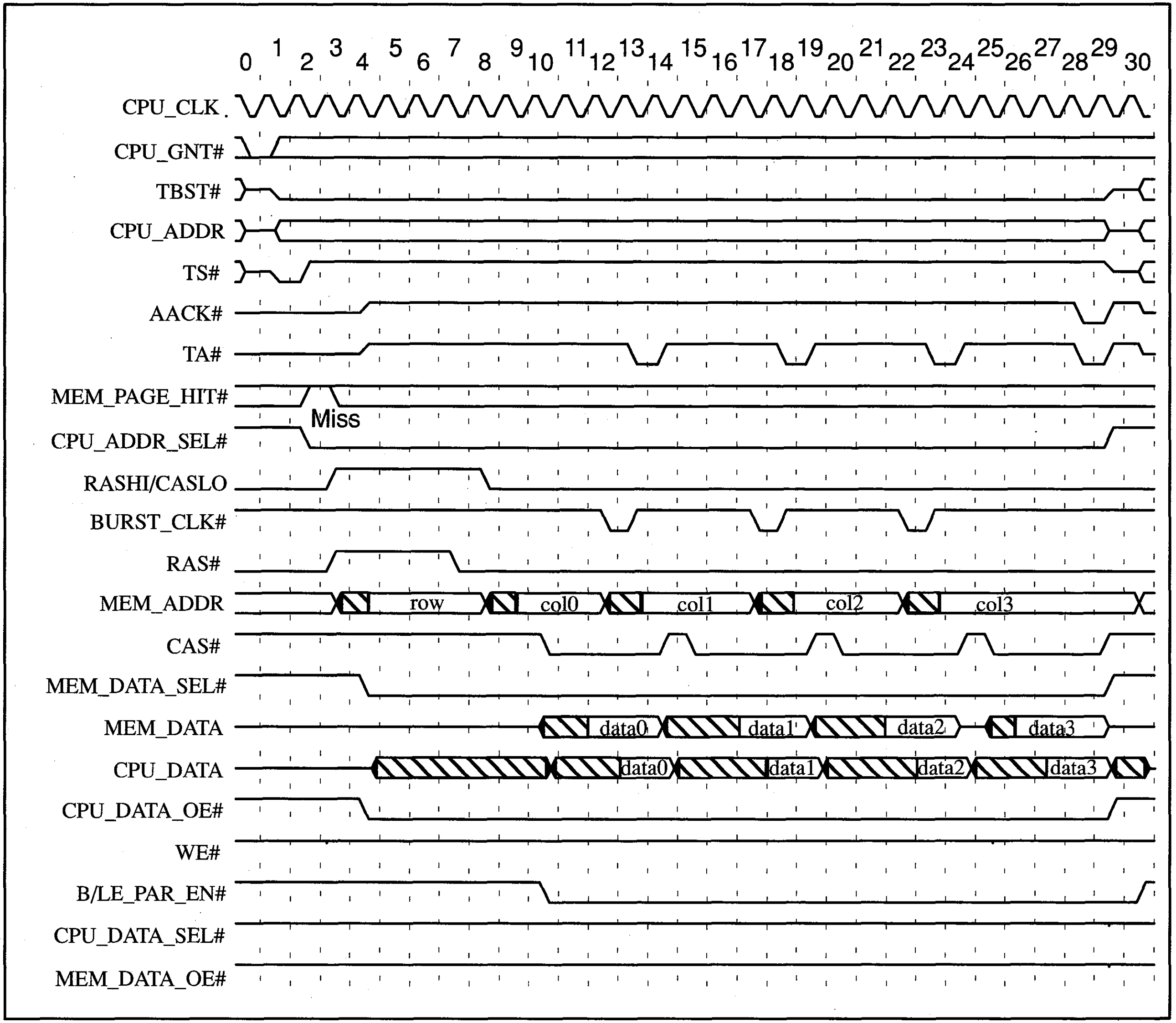


Figure 4–5. CPU to Memory Read, Burst, Page Miss, XCAS = 1 Timing Diagram

4.3.5 CPU to Memory Write—Single-Beat, Page Hit, XCAS = 0

The 60X CPU can initiate a write of system memory by executing a *store* instruction with an address range of 0G to 2G. See Table 4–1.

Figure 4–6 shows a single-beat system memory write with a page hit and XCAS = 0. The initial operations performed by the 650 Bridge during cycles 0 and 1 are identical to those in Figure 4–2. Note that the state of CPU_DATA_SEL# can not be determined from this transfer alone—it may have been negated by the previous transaction or it may be under asynchronous control of TT[1]. See Section 7.4.3.

In cycle 1, the 60X CPU asserts TS#, CPU_ADDR[0:31], TT[0:3], and TSIZ[0:2], and drives TBST# inactive. In response, the 650 Bridge evaluates the address, transfer type, and TBST# signal to determine that the CPU is requesting a single-beat write of system memory.

During cycle 2, the 653 Buffer asserts MEM_PAGE_HIT# to indicate that the row address of the memory read matches the previous row address. As a result of this signal, the 654 Controller leaves RASHI/CASLO low, RAS# stays low, and the 650 Bridge does not update the row address in the DRAM.

Also during cycle 2, the 654 Controller asserts CPU_ADDR_SEL#, CPU_DATA_SEL# (if it is not already asserted), MEM_DATA_OE#, and WE# to select the CPU as the source of the address and data for the transfer, enable the memory data bus drivers, and prepare the DRAM for a write cycle. The 653 Buffer processes the address and propagates it to the memory controller which selects the column address and drives it to the MEM_ADDR lines during cycle 3.

The CPU_DATA is expected to become valid at least by cycle 2, and is then propagated through the 653 Buffer to the MEM_ADDR lines during cycle 3.

AACK# and TA# stay tri-stated until cycle 4 to avoid contentions with an L2 cache in the event of a cache hit.

During cycle 4, the 654 Controller asserts the CAS[7:0]# lines to begin a CAS# write access.

During cycle 6, the 654 Controller asserts TA# and AACK# for one cycle, and the 60X CPU completes the memory write. The 654 Controller nagates the various control lines during cycle 7, with the exception of CPU_DATA_SEL#, which is still under asynchronous control as long as the 60X CPU has the bus grant.

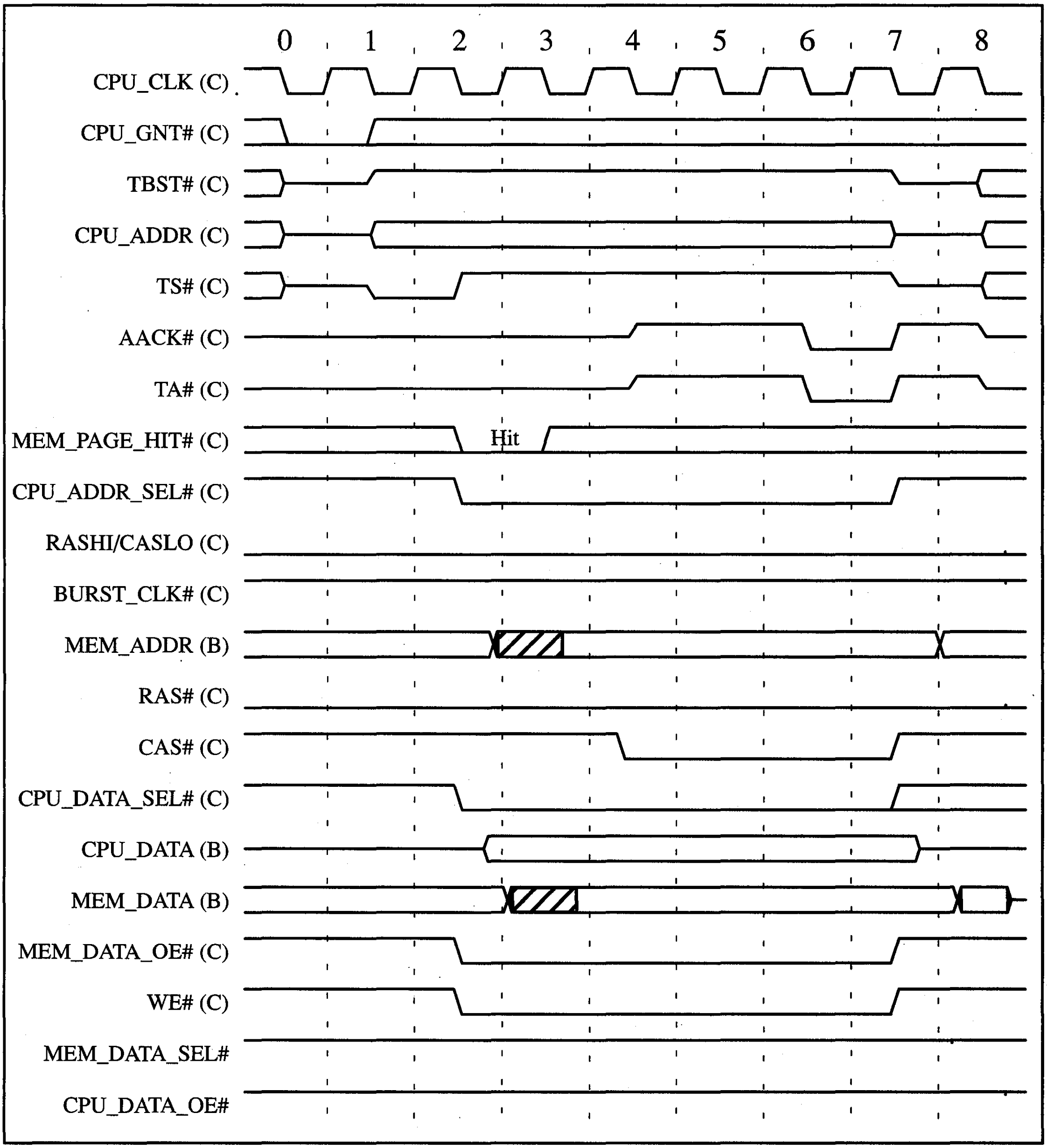


Figure 4–6. CPU to Memory Write, Single-Beat, Page Hit, XCAS = 0 Timing Diagram

4.3.6 CPU to Memory Write—Single-Beat, Page Hit, XCAS = 1

Figure 4–7 shows a single-beat system memory write with a page hit and XCAS = 1.

The operations performed by the 650 Bridge are identical to those in Figure 4–6 during cycles 0 through 3, and as usual AACK# and TA# stay tri-stated until cycle 4.

However, since XCAS = 1, the 654 Controller does not assert the CAS[7:0]# lines to begin a CAS# access until cycle 5. The CAS# write begins one cycle later than it does when XCAS = 0 but it is otherwise unchanged. Cycles 6 through 9 of this XCAS = 1 transfer are identical to cycles 5 through 8 of the XCAS = 0 transfer.

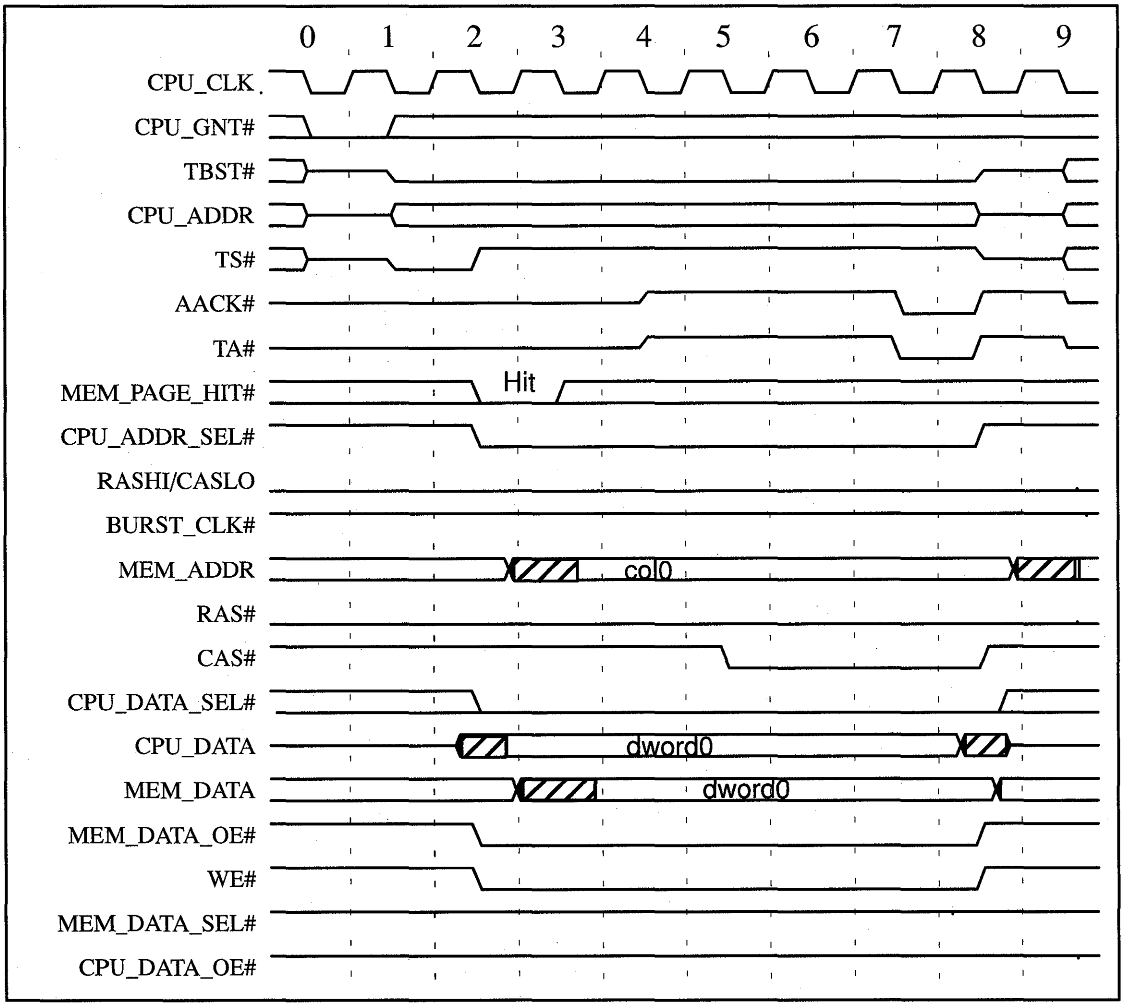


Figure 4–7. CPU to Memory Write, Single-Beat, Page Hit, XCAS = 1 Timing Diagram

4.3.7 CPU to Memory Write—Single-Beat, Page Miss, XCAS = 1

Figure 4–8 shows a single-beat system memory write with a page miss and XCAS = 1. This transfer is identical to the one shown in Figure 4–7 during cycles 0 through 2—AACK# and TA# stay tri-stated until cycle 4. During cycle 2, the 653 Buffer negates MEM_PAGE_HIT# to indicate that the row address of the memory write does not match the previous row address.

Beginning in cycle 3, the 650 Bridge inserts a RAS# access before the CAS# write access that would begin in cycle 4 during a page hit. This RAS# access is the same as the one executed during read miss transfers. The 654 Controller sends RASHI/CASLO high to cause the 653 Buffer to select the row address to drive onto the MEM_ADDR lines. The 654 Controller sends RAS# high for 4 cycles, and then asserts a RAS# line to latch the row address into the DRAM.

In cycle 8, the 654 Controller sends RASHI/CASLO low to cause the 653 Buffer to select the column address to drive onto the MEM_ADDR lines. This address propagates through to the DRAM during cycle 9.

In cycle 10, the 654 Controller asserts the selected CAS# lines to begin the CAS# write access. The rest of the transfer follows the page hit timing found in Figure 4–7.

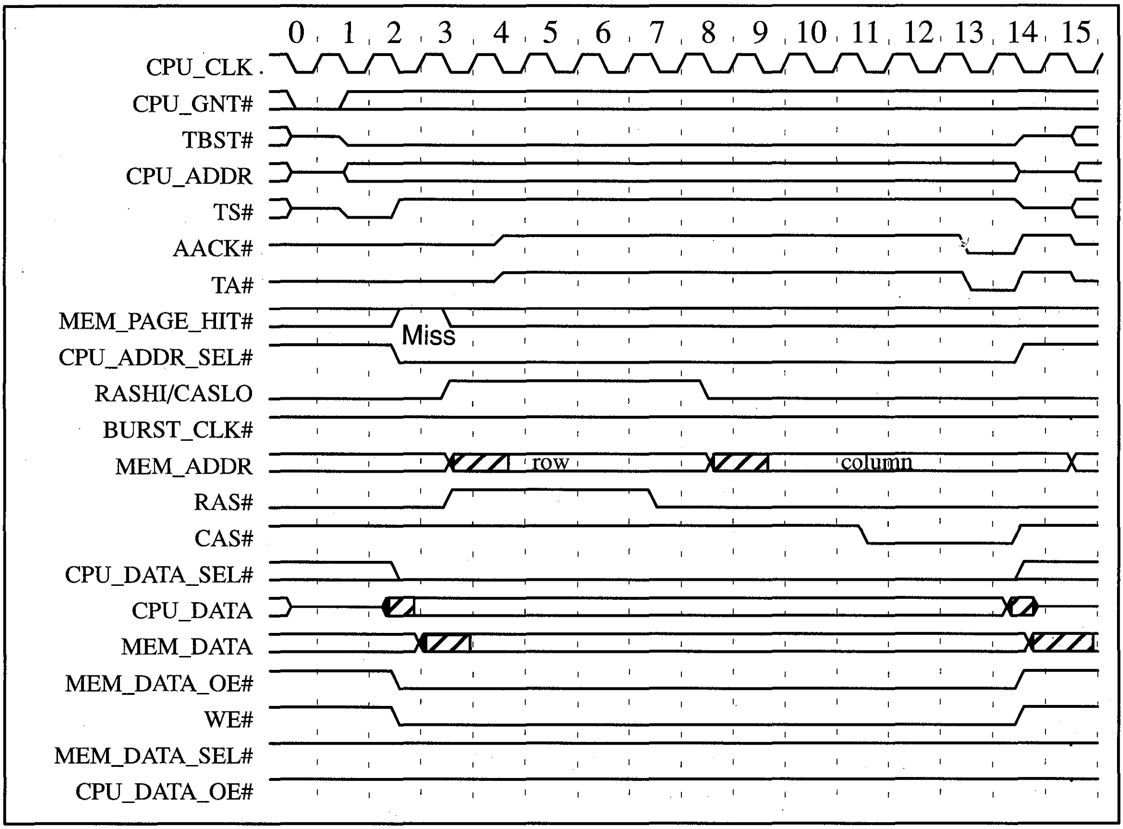


Figure 4–8. CPU to Memory Write, Single-Beat, Page Miss, XCAS = 1 Timing Diagram

4.3.8 CPU to Memory Write—Burst, Page Miss, XCAS = 1

Figure 4–9 shows a burst write to system memory with an initial page miss and XCAS = 1. The operations performed by the 650 Bridge are identical to those in Figure 4–8 during cycles 0 through 12. The 654 Controller leaves the data and address path control signals asserted, effectively stretching the transfer to accomodate the three extra beats of the burst.

In cycle 13, the 654 Controller begins a burst write, which is executed three times, once each for beats 1, 2, and 3, of the 4-beat burst (the beats are numbered 0 through 3). In this write, the 654 Controller asserts BURST_CLK# for one cycle to increment the column address presented to the DRAM (via the CPU Address Counter in the 653 Buffer) from the beat 0 (initial) address to the beat 1 address. The 654 Controller also asserts TA# for one cycle to signal the 60X CPU that the previous data has been written.

In cycle 14, the 654 Controller deasserts CAS# for two cycles. As CAS# is sent low again, the beat 1 address is latched into the DRAM, and the burst write cycle begins over again.

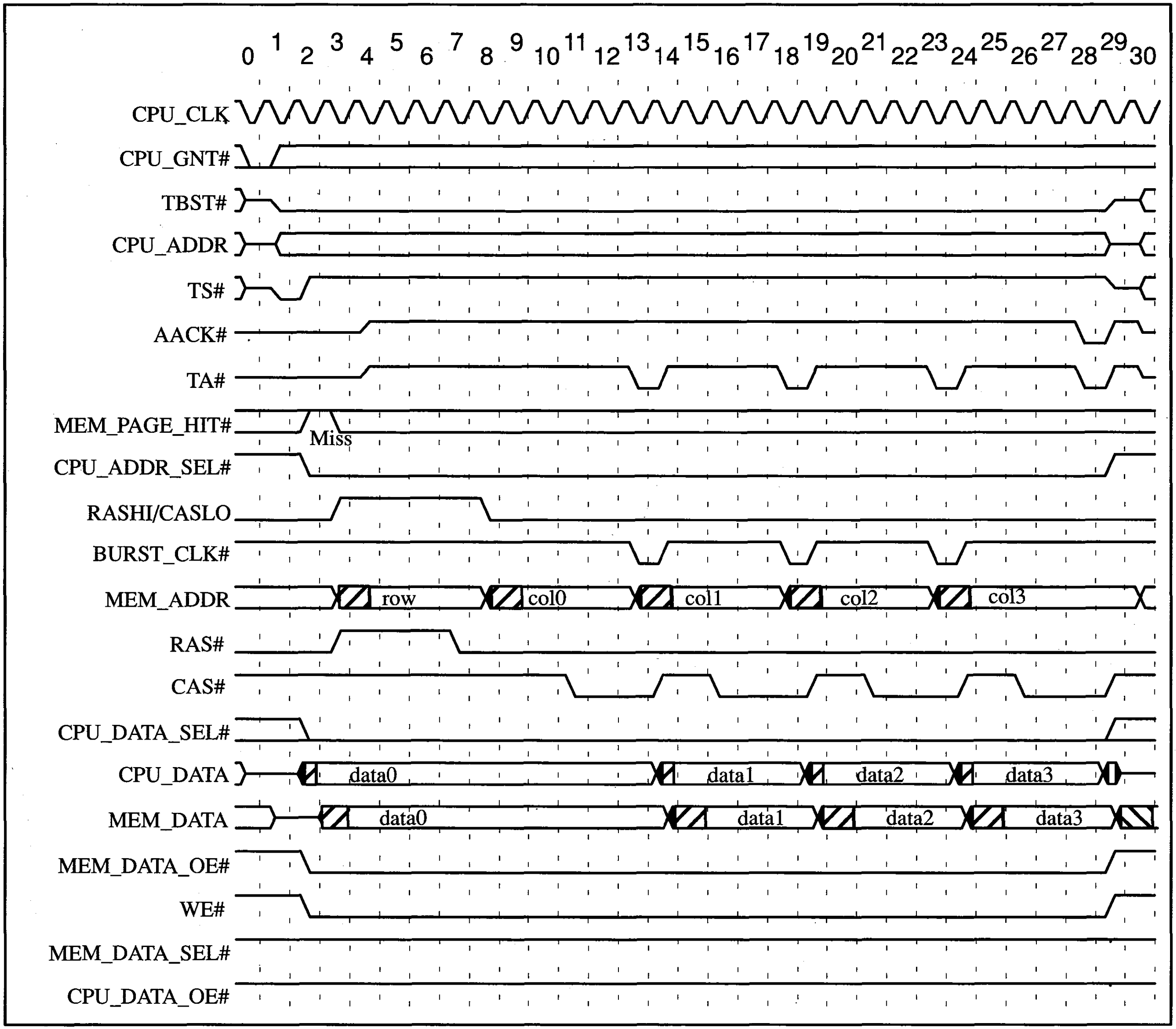


Figure 4–9. CPU to Memory Write, Burst, Page Miss, XCAS = 1 Timing Diagram

4.3.9 CPU to PCI Write—XADIO = 1

The 60X CPU can initiate a PCI write transaction by executing a *store* instruction with an address range of 2G to 4G. The exact type of PCI transaction is determined by the specific address range within 2G to 4G. See Table 4–1.

Figure 4–11 shows a CPU to PCI bus write transaction while XADIO = 1. During CPU to PCI transactions, the logic that controls the CPU interface operates in substantially the same manner as it does during CPU to memory transfers.

References to cycle x refer to the CPU_CLK cycle labled x. The following are specific notes for Figure 4–11 and following figures as applicable:

1. PCI_OE# is clocked by the rising edge of CPU_CLK (see Figure 4–10). PCI_OE# only changes state on a rising edge of the CPU_CLK on which the PCI_CLK is also rising. The signals (PCI_signal) that the PCI Specification defines relative to the PCI clock are handled the same way.

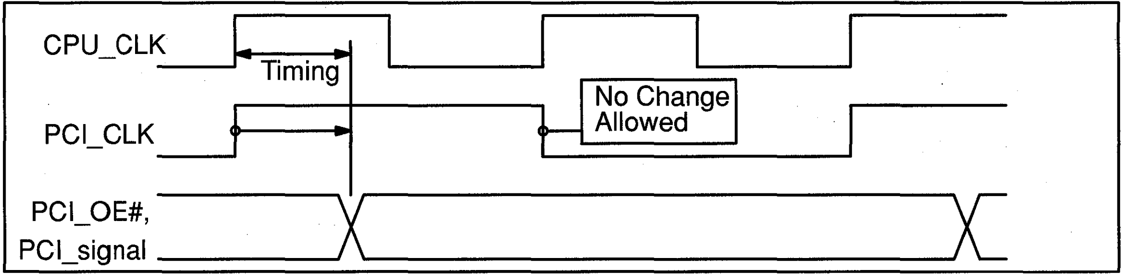


Figure 4–10. Timing of PCI_OE#

2. PCI_OE# enters this transaction deasserted if a PCI bus master has been in control of the PCI bus and the 650 is transferring PCI bus mastership to the CPU bus for this transaction. If the CPU bus mastered the previous transaction (or the bus was idle), then PCI_OE# has been asserted and is still asserted during PCI_CLK 0.
3. PCI_OE# is asserted while the 650 needs to drive address or data onto the PCI_AD bus. This occurs during CPU bus to PCI transaction address phases, CPU to PCI write transaction data phases, and while no PCI bus master is driving the PCI_AD bus (but not during turn-around cycles). Thus during this write transaction PCI_OE# is negated by the rising edge of PCI_CLK 6 only if the 650 grants the bus to a PCI bus master at the conclusion of this transaction. If the bus is not immediately granted to a PCI bus master, PCI_OE# remains asserted.
4. PCI_FRAME#, PCI_IRDY#, and PCI_C/BE[3:0]# enter this transaction tri-stated if a PCI bus master has been in control of the PCI bus and the 650 is transferring bus mastership to the CPU bus for this transaction. In this case, PCI_FRAME# and PCI_IRDY# are output enabled and driven high during CPU_CLK cycle two. If the CPU bus mastered the previous transaction (or the bus was idle), then these signals enter this transaction already output enabled and driven high.
5. If the arbiter transfers bus mastership from the 60X to a PCI bus master at the conclusion of this transaction, then the 650 tri-states PCI_FRAME#, PCI_IRDY#, and PCI_C/BE[3:0]# on the rising edge of PCI_CLK on which PCI_TRDY# (or PCI_STOP#) are sampled valid. If the bus is not immediately granted to a PCI bus master, these signals remain driven (output enabled) by the 650.

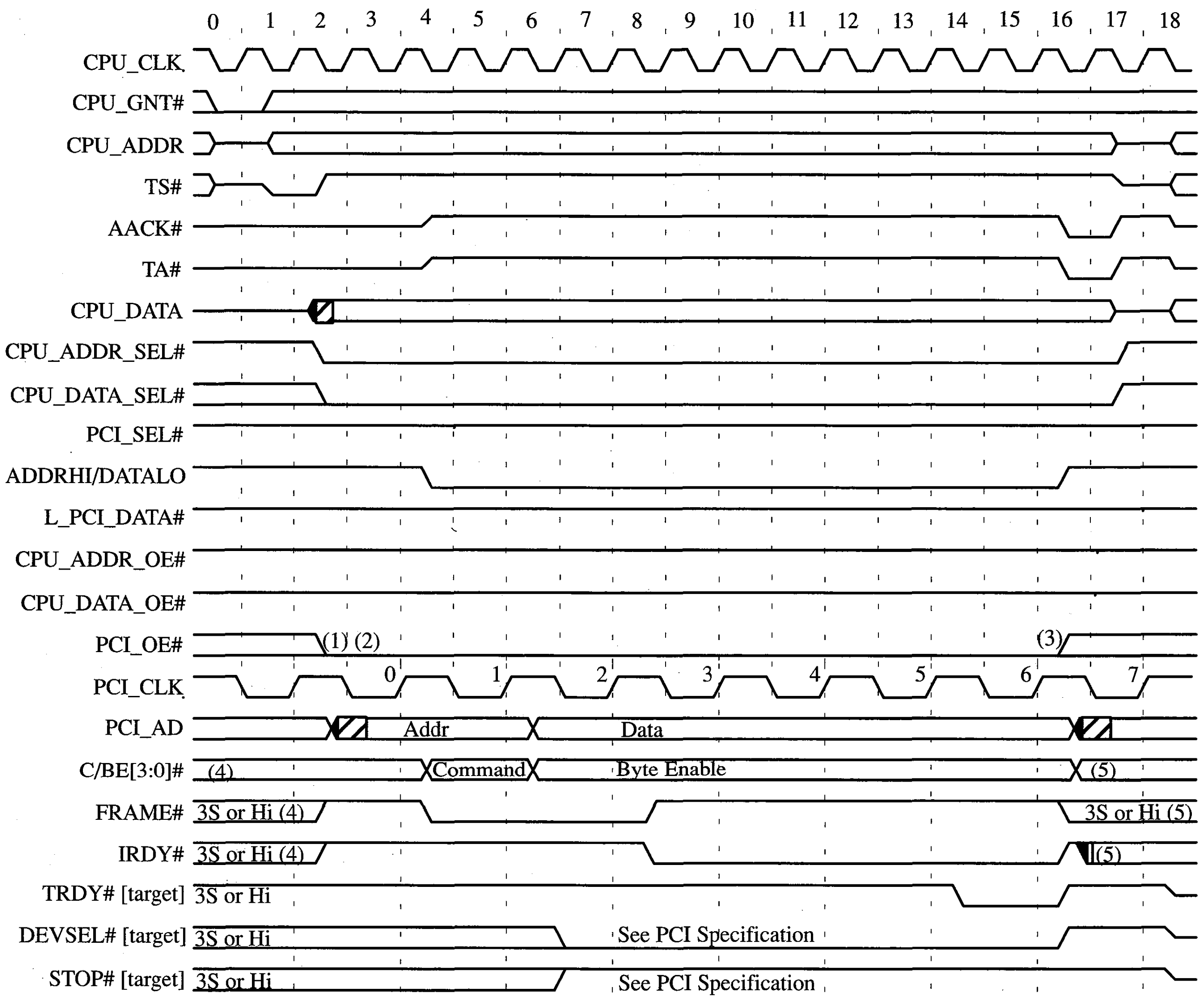


Figure 4–11. CPU to PCI Write, XADIO = 1 Timing Diagram

4.3.10 CPU to PCI Write Additional Timing Examples

Figure 4–12 shows a 60X CPU to PCI write with XADIO = 0, during which TS# is asserted across the falling edge of PCI_CLK rather than across the rising edge of PCI_CLK. Most of the timing diagrams show TS# asserted across a rising edge of PCI_CLK, but it is equally likely that TS# will be asserted across a falling edge of PCI_CLK. When this happens, the 650 Bridge responds by stretching the transaction, effectively adding a CPU_CLK wait cycle after CPU_CLK 3 to synchronize the transaction to PCI_CLK. The rest of the transaction remains unchanged.

Figure 4–13 shows a 60X CPU to PCI write with XADIO = 1, during which the target device asserts a target retry (PCI_STOP# and PCI_DEVSEL# asserted together). A PCI device can target retry a transaction for various reasons. Following a target retry, the initiating device can retry the transaction.

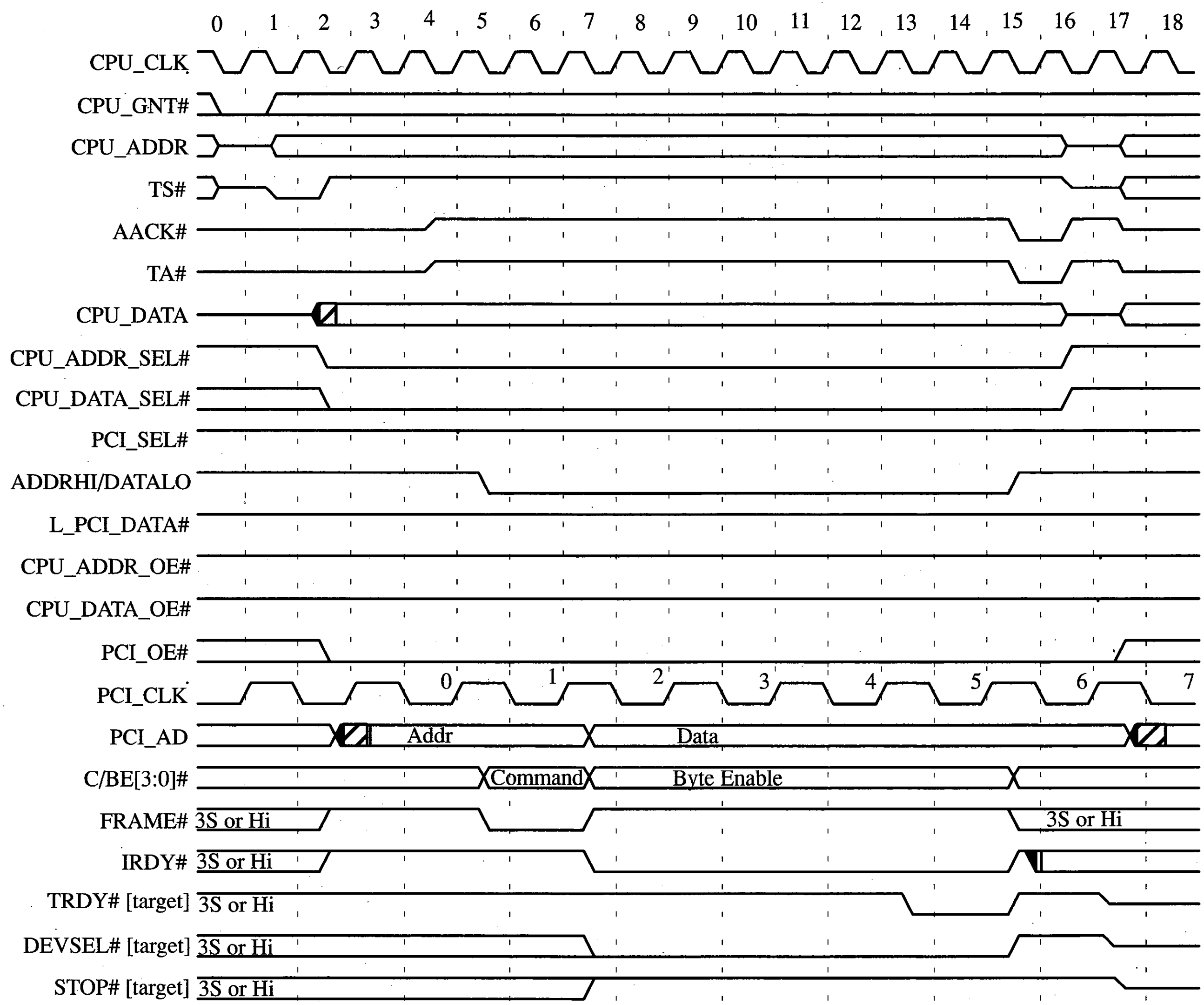


Figure 4–12. CPU to PCI Write, XADIO = 0, Offbeat TS# Timing Diagram

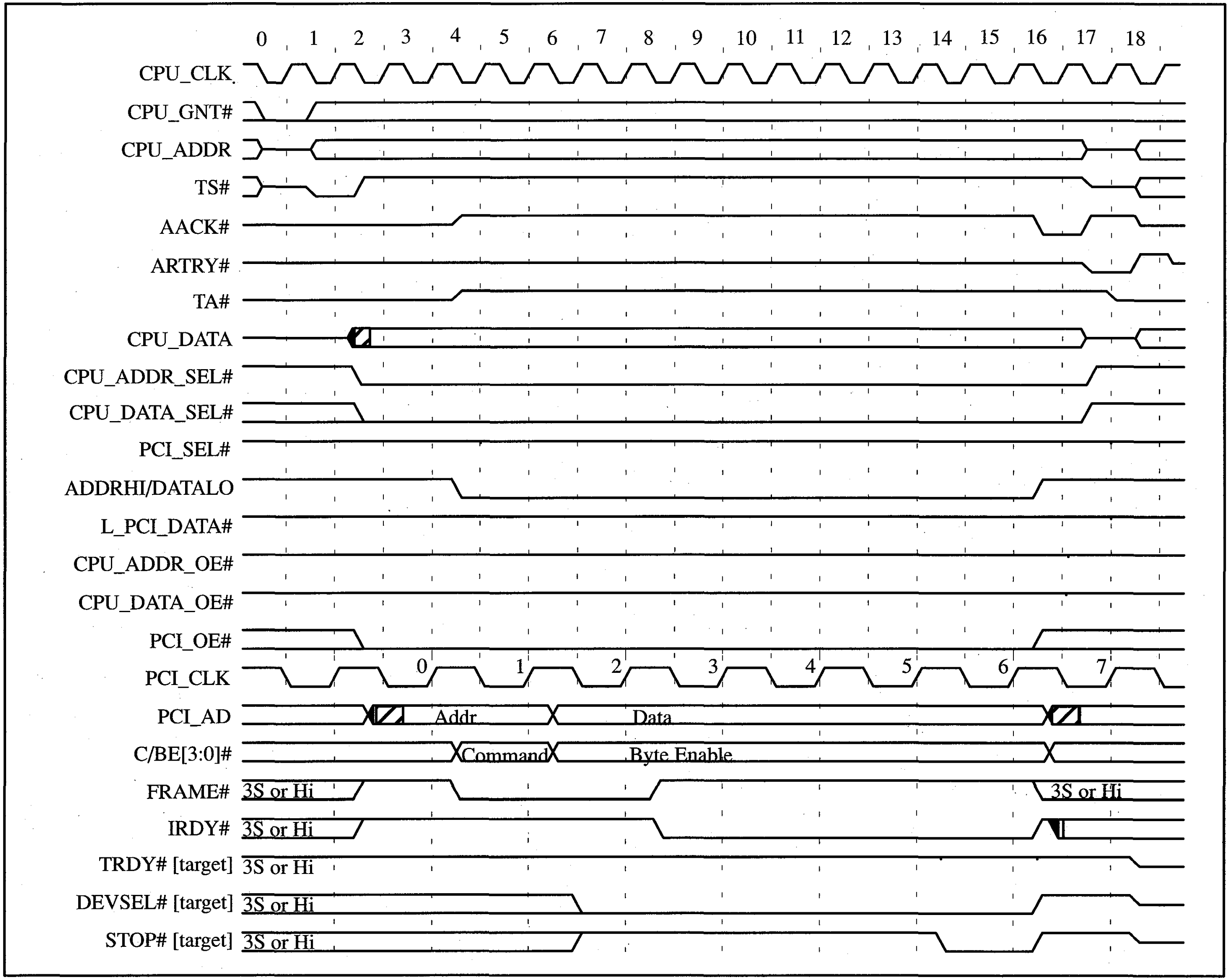


Figure 4–13. CPU to PCI Write, XADIO = 1, Target Retry Timing Diagram

4.3.11 CPU to PCI Read

Figure 4–14 shows a 60X CPU to PCI read. The 60X CPU can initiate a PCI read transaction by executing a *load* instruction with an address range of 2G to 4G. The exact type of PCI transaction is determined by the specific address range within 2G to 4G. See Table 4–1. Note that the 654 deasserts PCI_SEL# one PCI_CLK after it samples PCI_TRDY# or PCI_STOP# asserted.

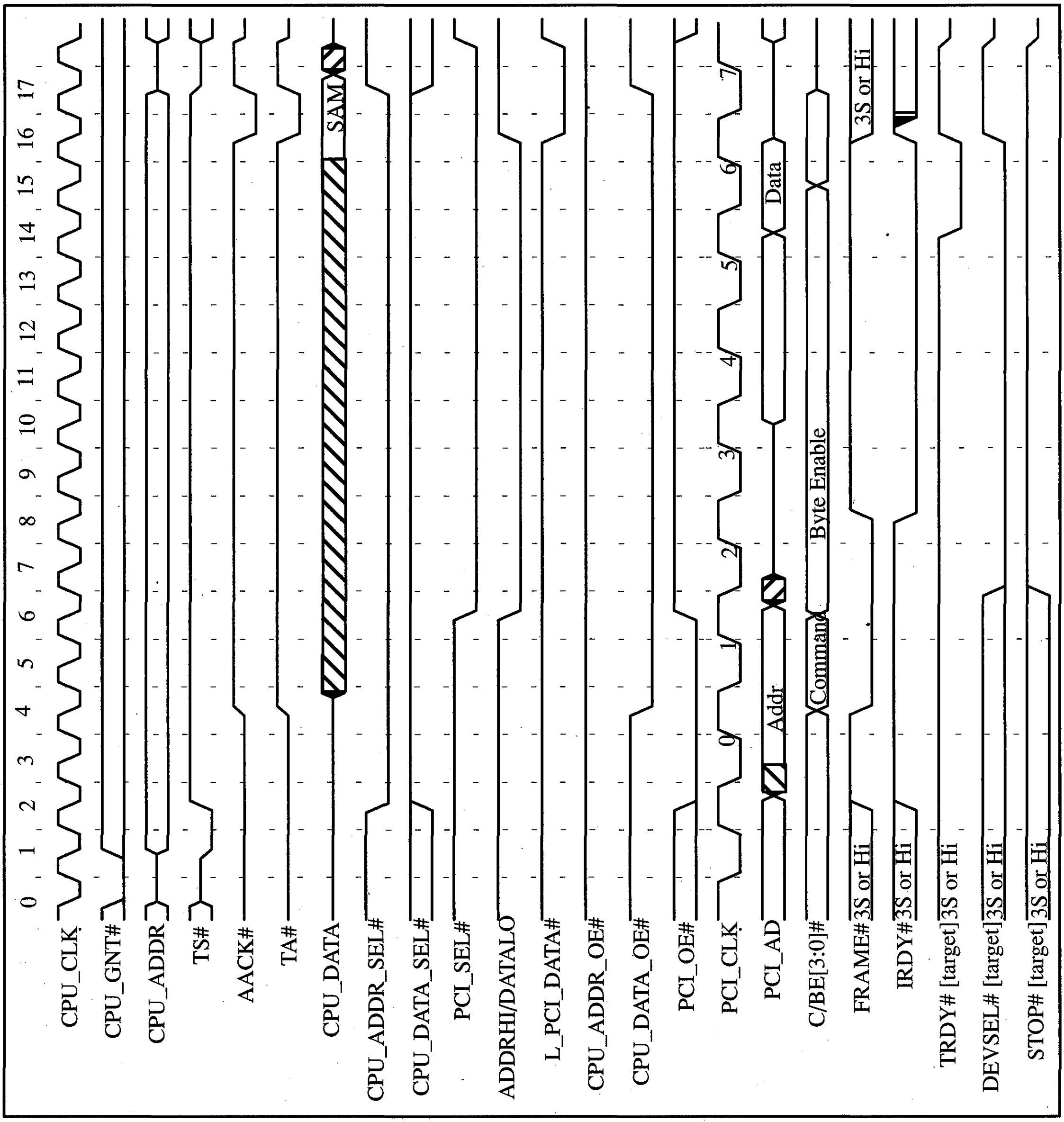


Figure 4–14. CPU to PCI Read Timing Diagram

4.3.12 PCI to Memory Read—Single-Beat, Page Hit

During PCI to memory transactions, the 650 Bridge updates the PCI address latch in the 653 Buffer on each rising edge of the PCI_CLK while ADDRHI/DATALO is high, so in Figure 4–15, the PCI address latch is updated on PCI_CLK 1. Also on PCI_CLK 1, the 650 Bridge samples PCI_FRAME# active, which starts the 650 Bridge PCI target cycle (assuming that the PCI bus master is addressing system memory).

The 654 Controller sends ADDRHI/DATALO low on PCI_CLK 1 to hold the PCI address in the latch. PCI_TRDY#, PCI_DEVSEL#, and PCI_STOP# have been tri-stated since the beginning of the cycle; on PCI_CLK 2, the 650 asserts PCI_DEVSEL# to claim the transaction, and drives PCI_STOP# and PCI_TRDY# high. The 654 Controller asserts PCI_OE# on PCI_CLK 2 to enable the PCI_AD drivers in the 653 Buffer (the cycle between PCI_CLKs 1 and 2 is a turn–around cycle (TAC) for the PCI_AD lines, and some control lines).

The 654 Controller begins a CAS# read to the memory. This CAS# read is similar to that used when the CPU is reading system memory. The 653 Buffer drives valid data onto the PCI_AD lines in time to meet the required PCI data setup times for PCI_CLK 5, so the 654 Controller asserts PCI_TRDY# on PCI_CLK 4. The 654 Controller then negates PCI_TRDY#, PCI_DEVSEL#, and PCI_STOP#, and negates PCI_OE# to tri-state the PCI_AD lines. The 654 Controller tri-states PCI_TRDY#, PCI_DEVSEL#, and PCI_STOP# on PCI_CLK 6 (see notes 1 and 2).

The 650 Bridge generates a snoop cycle on the 60X CPU bus for each PCI to system memory transaction. In this transaction (Figure 4–15), CPU_ADDR_OE# has been asserted (see note 1), so the 653 Buffer is driving the (translated) PCI address onto the CPU address lines. The 654 Controller asserts TS# for one CPU_CLK cycle, followed by asserting AACK# for one CPU_CLK cycle, in compliance with 60X CPU bus snoop cycle requirements. Should either the L1 or L2 caches detect a cache hit, it must assert ARTRY# so that it is sampled valid at least by the second CPU_CLK after it samples TS# valid, or it is not recognized.

These notes refer to Figure 4–15 and to following figures as appropriate.

1. During PCI to memory transactions, the 654 Controller drives PCI_SEL#, CPU_ADDR_OE#, and AACK# depending on two factors—the state of the transaction engine and the state of the arbiter engine. Depending on the status of the system on the rising edge of the PCI_CLK on which the 654 tri-states PCI_TRDY# (in this case PCI_CLK 6), the arbiter either removes the grant from the current PCI bus master or that bus master retains mastership of the system. If the PCI bus master retains the grant, the 654 leaves PCI_SEL# and CPU_ADDR_OE# low, and continues to drive AACK# high into the next cycle. If the PCI bus master is losing the grant, then (on the rising edge of the PCI_CLK on which the 654 tri-states PCI_TRDY#) the 654 drives PCI_SEL# and CPU_ADDR_OE# high, and tri-states AACK#. See Table 4–3.

Table 4–3. Effects of Arbiter on Three Signals

Signal	PCI Retains System Mastership	PCI Loses System Mastership
PCI_SEL#	Remains driven low.	Is driven high.
CPU_ADDR_OE#	Remains driven low.	Is driven high.
AACK#	Remains driven high.	Is tri-stated.

2. During PCI to memory read transactions, ADDRHI/DATALO is deasserted on the PCI_CLK that the 654 Controller tri-states PCI_TRDY# (in this case PCI_CLK 6). During PCI to memory writes, the 654 deasserts ADDRHI/DATALO one PCI_CLK earlier.
3. ISA master devices can access system memory from 0 to 16M with a direct address of 0 to 16M. See Section 5.6.1.2 and Section 5.6.1.3.

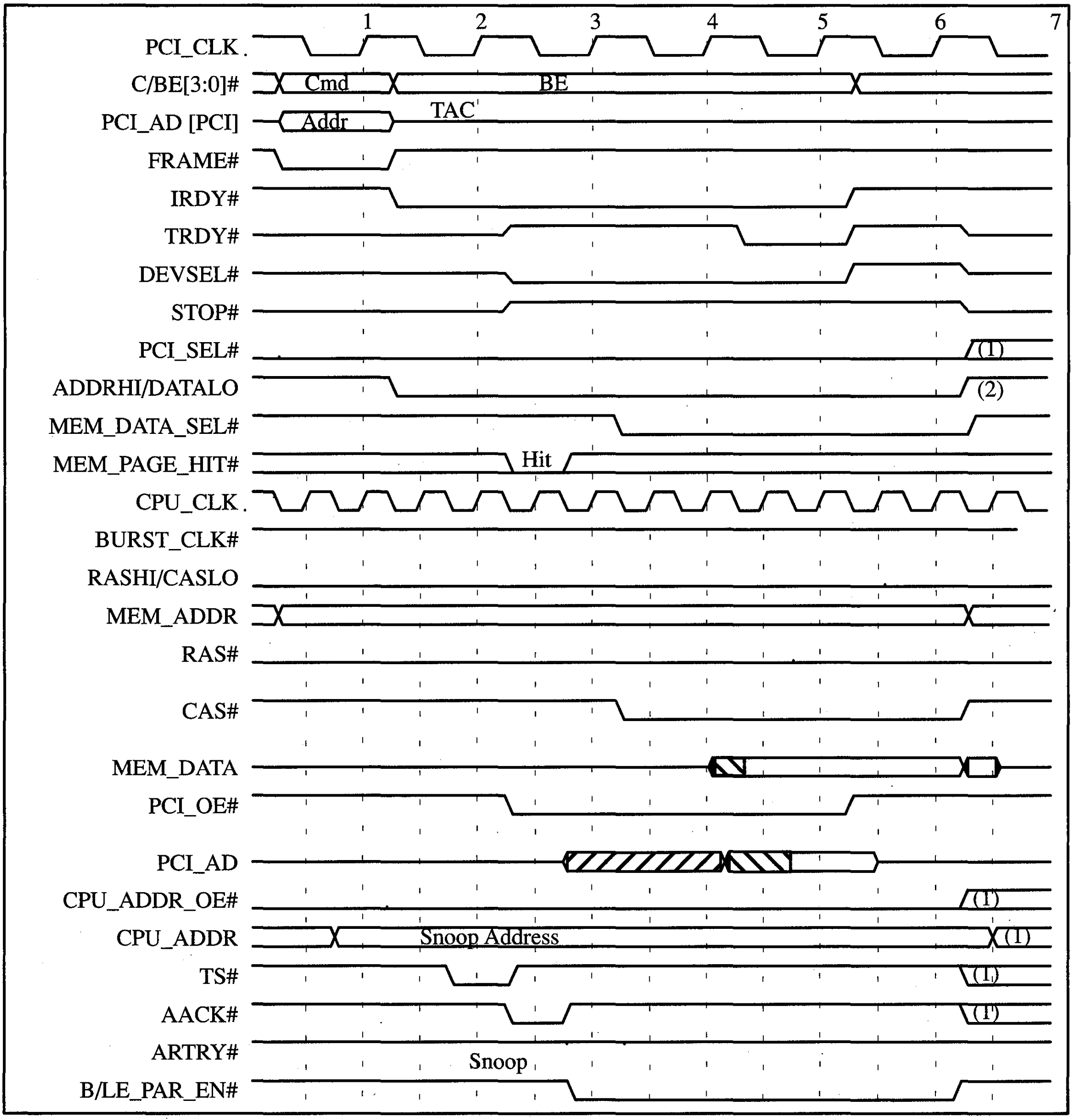


Figure 4–15. PCI to Memory Read, Single-Beat, Page Hit Timing Diagram

4.3.13 PCI to Memory Read—Burst, Page Hit Then Miss

When a PCI bus master is reading from system memory in burst-mode, a page miss can occur at any point in the transaction. Figure 4–17 shows this type of page hit and page miss activity. During PCI to memory reads, each data phase requests up to four bytes from the 650 bridge, but the 650 Bridge always reads eight bytes from the memory subsystem (see Figure 4–16). During burst reads that start on an eight–byte boundary (PCI_AD[2] = 0 during the address phase), the 650 Bridge performs the following steps:

1. Reads eight bytes from memory (and generates a snoop cycle to the 60X bus),
2. Delivers the lower four bytes to the PCI for the first data phase,
3. Delivers the upper four bytes to the PCI for the second data phase,
4. Reads another eight bytes from memory (and generates a snoop cycle),
5. Delivers the lower four bytes to the PCI for the third data phase,
6. Delivers the upper four bytes to the PCI for the fourth data phase,
7. Repeats steps 4., 5., and 6. as required.

With a 2:1 CPU bus to PCI bus clocking mode, this process yields burst read performance (assuming no page misses) of 5–4–3–4–3–4–3–, etc.

During burst reads that start on a four-byte but not an eight–byte boundary (PCI_AD[2] = 1 during the address phase), the 650 Bridge performs the following steps:

1. Reads eight bytes from memory (and generates a snoop cycle to the 60X bus),
2. Delivers the upper four bytes to the PCI for the first data phase,
3. Reads another eight bytes from memory (and generates a snoop cycle),
4. Delivers the lower four bytes to the PCI for the second data phase,
5. Delivers the upper four bytes to the PCI for the third data phase,
6. Repeats steps 3., 4., and 5. as required.

With a 2:1 CPU bus to PCI bus clocking mode, this process yields burst read performance (assuming no page misses) of 5–4–4–3–4–3–4–3–, etc.

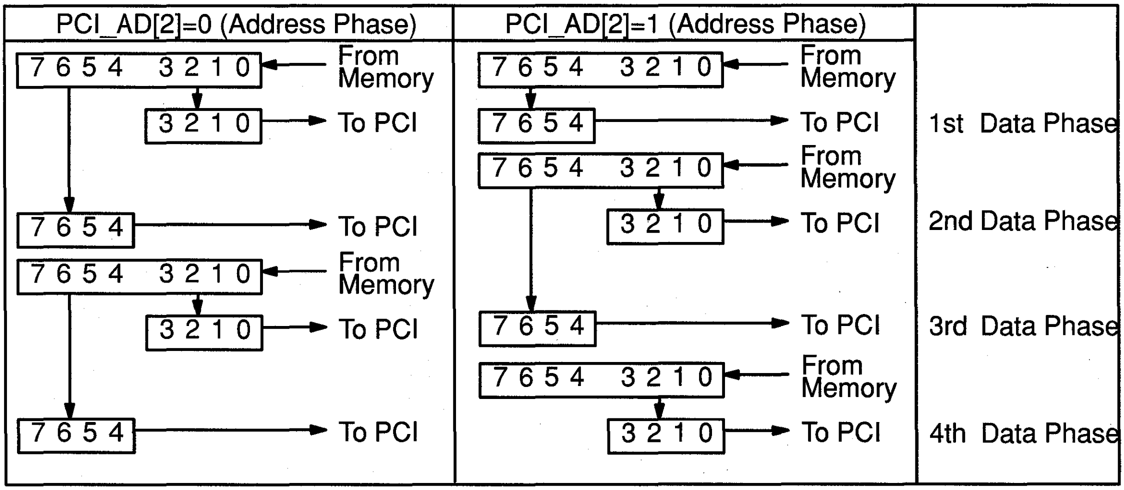


Figure 4–16. PCI To Memory Burst Read Transaction

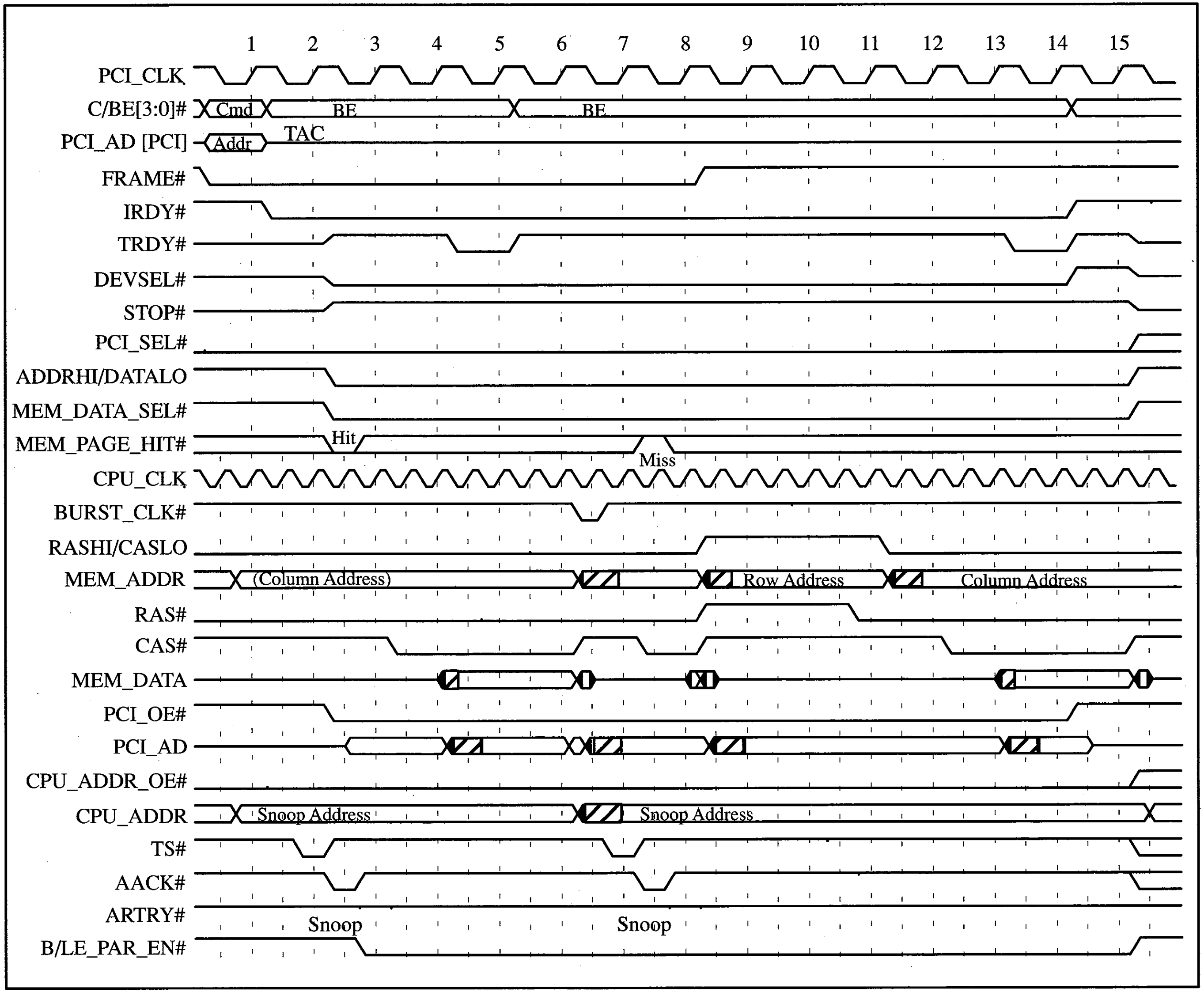


Figure 4–17. PCI to Memory Read, Burst, Page Hit Then Miss Timing Diagram

4.3.14 PCI to Memory Write—Burst, Page Miss Then Hit

During PCI to memory writes, the 650 Bridge asserts WE# on PCI_CLK 2 to begin the DRAM write operation, and asserts MEM_DATA_OE# on PCI_CLK 2 to enable the 653 Buffer to drive data onto the MEM_DATA bus.

In Figure 4–18, the 653 Buffer negates MEM_PAGE_HIT# on PCI_CLK 2 to signal a page miss to the 654 Controller, which then begins a RAS# access. This RAS# access is similar to the one that the 654 Controller executes during a 60X CPU to memory page miss. After the RAS# access completes, the 654 Controller executes CAS# writes, which are also similar to those executed by the 654 during 60X CPU to memory writes.

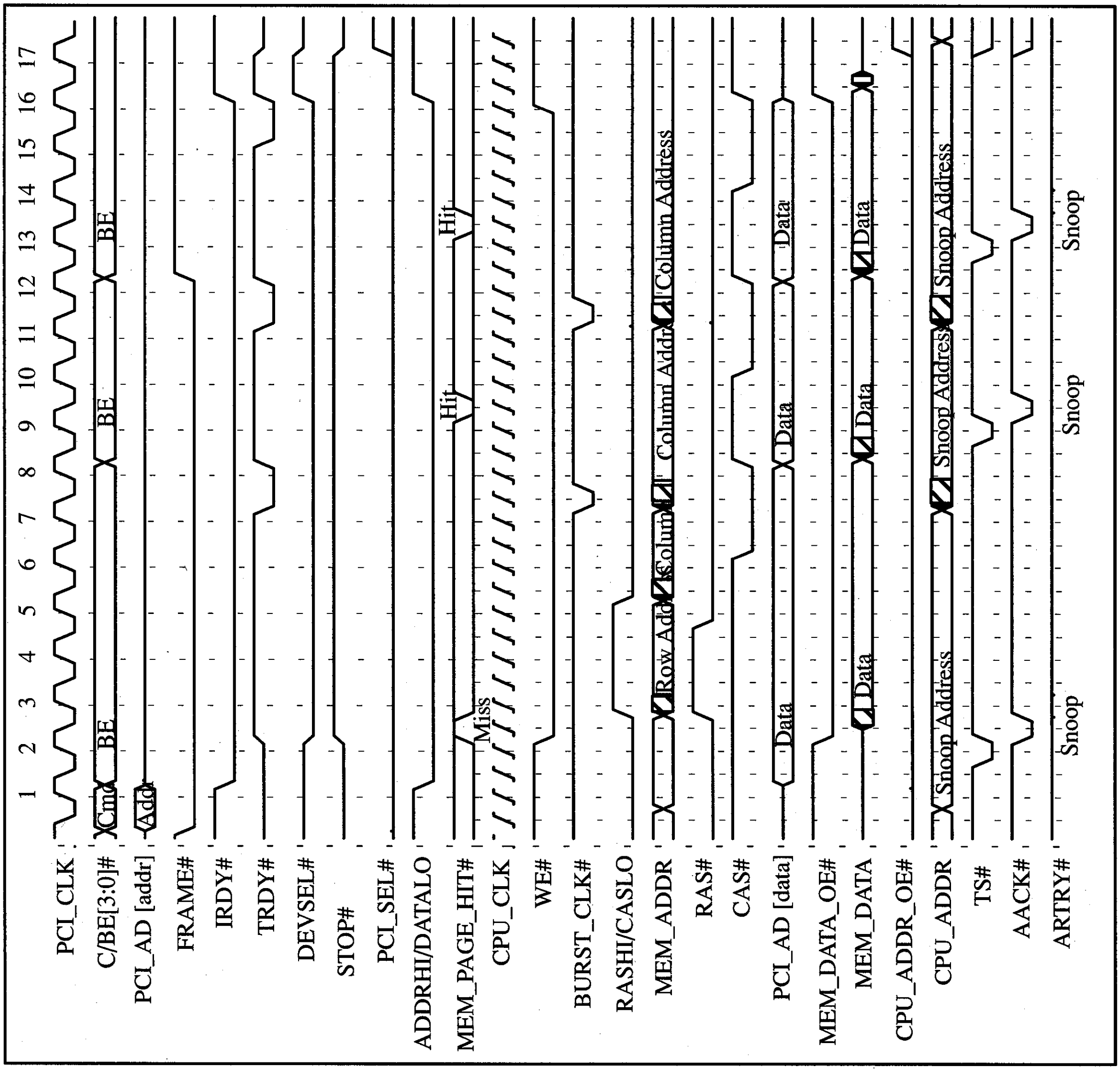


Figure 4–18. PCI to Memory Write, Burst, Page Miss Then Hit Timing Diagram

Section 5
The 650 Bridge Functional Description

This section describes in detail all the possible operations the 650 Bridge can perform in its role as PCI bridge, bus arbiter, memory controller, and system resource manager.

Section 5.1 describes how the 650 Bridge controls the PCI local bus and the 60X CPU host bus by means of an arbiter that allocates bus access based on priority and fairness algorithms.

Section 5.2 describes the programmability of the 650 Bridge, including the SIMM mapping registers and the system setup register.

Section 5.3 describes little-endian and big-endian addressing theory and the implementation of this theory in the 650 Bridge.

Section 5.4 describes the operation of the memory controller including RAS# and CAS# logic for both big-endian and little-endian addressing modes.

Section 5.5 describes the 60X CPU bus transactions that the 650 Bridge can service.

Section 5.6 describes the PCI to 650 Bridge transactions.

Section 5.7 describes the operation of the optional L2 cache.

Section 5.8 describes system errors and the methods that can be used to access and report errors and exceptions.

5.1 The 650 Bridge Arbiter

The 650 Bridge arbiter allocates 60X CPU bus and PCI local bus cycles. If two or more masters are requesting the bus, their requests are latched and then granted in the following order:

- DRAM refresh requests (up to three refresh requests can be queued, in addition to one refresh in progress.)
- 60X CPU bus requests (instruction fetches, system memory, ROM, and PCI)
- L2 cache to 60X CPU bus requests (snoop hits on write-back cache or castouts)
- I/O Bridge (a special PCI device)
- PCI bus requests (up to five additional request and grant lines)

Once a bus has been granted to a device, the 650 Bridge evaluates the bus transaction and generates the responses required for each type of bus transaction.

5.1.1 Arbitration Description

The 650 Bridge provides control for the PCI local bus as a PCI initiator and as a PCI target. The Bridge also interacts with the 60X CPU bus as both a slave and to master snoop cycles. For the purpose of arbitration, the system is treated as a single bus system. Arbitration is designed to ensure that only one master may control the buses at any time, with fairness as well as a timeout counter that assists in maintaining the PCI bus latency.

The 654 Controller provides the arbitration for the 60X CPU and PCI bus as a single bus system. Either the 60X CPU bus or the PCI bus can execute a cycle at any given time.

5.1.2 The Arbitration Fairness Mechanism

The 650 Bridge uses a bus request queue to implement a fixed priority with fairness algorithm that minimizes access latency on the PCI bus. When the bus request queue is empty, the 650 Bridge queues all the currently active bus requests. The arbiter grants all the queued requests in priority order before reloading the queue again. The bus request queue mechanism ensures that lower-priority devices are not locked out by very busy high-priority devices.

DRAM refresh is an exception to fairness. See Section 5.1.4.

5.1.3 The Timeout Counter

All masters must be given access to the bus within the maximum latency limits of the system. A 6-bit timeout counter assists in meeting the latency limits. When the bus is granted and another master requests the bus, the timeout counter is started (based on PCI_CLK). The timeout counter operation is suspended when IO_BRDG_HOLD# is active until the I/O bridge device drives PCI_FRAME#.

If the timeout counter counts out before a device completes its tenure on the bus, the 650 Bridge removes the bus grant for the device.

5.1.4 Support for System Memory (DRAM) Refresh

System memory refresh can only occur when the two system buses are idle. The REFRESH_REQ# signal from an external device acts as a high-level bus request, removing the grant from the current master. When the current master deasserts PCI_FRAME# and completes its PCI cycle, the 650 Bridge initiates refresh cycles. When refresh has been completed, all pending bus requests are granted in the same order as before the refresh was started. Up to seven refreshes can be queued in addition to the currently operating refresh cycle.

Each time the REFRESH_REQ# signal activates, a request counter within the 650 Bridge is incremented. As soon as the current bus master releases control, the 650 Bridge generates refresh cycles until the request counter is decremented to zero. Refresh cycles take approximately 16 CPU clock cycles each and their timing is independent of the duration of the REFRESH_REQ# signal. No other bus cycles are initiated while the refresh cycles are active. The maximum count of the request counter is seven. Refresh does not obey the fairness rule. Refresh is always granted ahead of any other requests as soon as the bus is available.

The device that generates REFRESH_REQ# must be programmed for a refresh interval appropriate to the DRAM used in the system.

5.1.5 Support for Cache Snooping

For PCI to system memory cycles, the 654 Controller masters the 60X CPU bus to run a snoop cycle (to maintain cache coherency in the L1 cache in the 60X CPU and the optional L2 cache).

If the 60X CPU or the L2 cache asserts ARTRY# because of a snoop hit, the 654 Controller terminates the current PCI operation with target retry (PCI_STOP# and PCI_DEVSEL# are asserted) and the current master of the PCI bus immediately relinquishes the bus grant. The 654 Controller then grants the bus to the 60X CPU or the L2 cache to do a write-back to system memory. When the write-back is completed, normal priority scheduling resumes, starting with the master that lost the bus grant due to the snoop hit (if it is requesting the bus).

5.1.6 Bus Parking

During cycles when the system buses are idle and no masters are requesting either bus, the 654 Controller arbiter parks the bus on the 60X CPU (asserts CPU_GNT#) and enables the PCI_AD, PCI_C/BE[3:0]#, and PCI_PAR drivers (in conformance with the *PCI Specification*, revision 2.0).

Parking the bus on the 60X CPU allows the 60X CPU to drive the bus with zero clock delays, as described in the PCI Specification.

5.2 650 Bridge Programmability

The 650 Bridge has internal registers that are programmable. The 650 Bridge memory controller requires programming for memory configuration for any memory array other than the 8M default. The 650 Bridge setup register can also be programmed for various optional modes of operation.

The memory registers and system setup register cannot be programmed or read without corrupting system memory (DRAM). For this reason, these registers should be programmed before system memory is initialized.

5.2.1 Programming the 650 Bridge Memory Controller

The 650 Bridge directly supports 8M and 32M, 70ns page-mode parity 168-pin SIMMs. The 650 Bridge also supports, with buffers, 4M, 8M, 16M, and 32M 72-pin industry-standard parity SIMMs. A maximum of eight SIMMs can be configured for a maximum system memory of 256M. Fast page mode is supported. One RAS per logical SIMM is supported. The width of the memory is eight bytes.

System memory (DRAM) is configured within the 650 Bridge by eight SIMM registers—seven SIMM mapping registers and one SIMM top-of-memory register.

5.2.1.1 Memory Controller Configuration

External logic asserts MC_SETUP# during a PCI I/O read or write cycle for memory controller configuration. Register selection and configuration data are accessed through the CAS# lines. See Section 5.2.3.

5.2.1.2 SIMM Mapping Registers

The memory controller within the 654 Controller contains seven SIMM mapping registers and a SIMM top-of-memory register. Each SIMM mapping register indicates the starting address of that SIMM modulo 8M. The SIMM mapping register information controls the assertion of the RAS# line corresponding to a memory address. Each SIMM mapping register consists of the following eight bits:

> MR[7:5]—The encoded address bits, the three most-significant bits in the byte
>
> MR[4:0]—The starting address bits, the five low-order bits in the byte

The encoded address bits MR[7:5] indicate the address of the register to be accessed. Each of these addresses corresponds to a SIMM register as shown in Table 5–1.

SIMM register 000b is the top-of-memory register. Each of the other seven SIMM registers is a SIMM mapping register, associated with the like-numbered SIMM slot.

The numbering of SIMM slots begins with slot 0. Since slot 0 starts with memory address 00000000h, there is no necessity for a SIMM mapping register to indicate the starting address for the first SIMM slot.

Table 5–1. SIMM Mapping Register Selection

Address Bits MR[7:5]	Register Name
000b	Top-of-Memory Register
001b	SIMM Mapping Register 1
010b	SIMM Mapping Register 2
011b	SIMM Mapping Register 3
100b	SIMM Mapping Register 4
101b	SIMM Mapping Register 5
110b	SIMM Mapping Register 6
111b	SIMM Mapping Register 7

5.2.1.3 SIMM Starting Address Registers

Beginning with SIMM slot 1 (the second SIMM slot), each of the SIMM mapping registers contains the starting address for its SIMM slot, modulo 8M. For example, the value of MR[4:0] in SIMM mapping register 1 is 00001b if there is an 8M SIMM in slot 0 (see Table 5–2). If there is a 32M SIMM in slot 0, the value of MR[4:0] in SIMM mapping register 1 is 00100b.

Table 5–2. SIMM Mapping Register Starting Addresses

MR[4:0]	Starting Address	MR[4:0]	Starting Address
00000	0M	10000	128M
00001	8M	10001	136M
00010	16M	10010	144M
00011	24M	10011	152M
00100	32M	10100	160M
00101	40M	10101	168M
00110	48M	10110	176M
00111	56M	10111	184M
01000	64M	11000	192M
01001	72M	11001	200M
01010	80M	11010	208M
01011	88M	11011	216M

Table 5–2. SIMM Mapping Register Starting Addresses (Continued)

MR[4:0]	Starting Address	MR[4:0]	Starting Address
01100	96M	11100	224M
01101	104M	11101	232M
01110	112M	11110	240M
01111	120M	11111	248M

5.2.1.4 SIMM Starting Address Rules

The starting address for a slot is used with the starting address for the next slot to determine the SIMM to be activated. The formula for activating RAS[n] is as follows:

$$RAS_n = SA_n \leq Address < SA_{n+1}$$

where:

- RAS[n] is the RAS signal for SIMM slot n.
- SA_n is the contents of the SIMM starting address register for SIMM slot n.
- SA_{n+1} is the contents of the SIMM starting address register for SIMM slot n+1.
- Address is the input address of the memory to be accessed.
- One and only one RAS[7:0]# line can be asserted for a memory read or write cycle.

5.2.1.5 SIMM Top-of-Memory Logic

The eight bits of the SIMM top-of-memory register are encoded like the SIMM starting address registers. The MR[7:5] address bits for the SIMM top-of-memory register are set to 000b to address the top of memory register. See Table 5–1.

Program register bits MR[4:0] to represent the address of the top of system memory minus 8M. Any memory access with an effective address above the address plus 8M generates an out-of-range memory access. An out-of-range memory read asserts ALL_ONES_SEL# to the 653 Buffer to output 64 one-bits on the data bus with normal TA# termination. For PCI bus transactions, an out-of-range memory error terminates the current PCI transaction with a target abort (PCI_DEVSEL# deasserted and PCI_STOP# asserted).

Any address greater than or equal to 256M is hard decoded by the 654 Controller as out-of-range.

Notice that the SIMM top-of-memory register serves as SIMM mapping register 8 for SIMM mapping register 7. The SIMM top-of-memory register provides the next slot comparison that is necessary to activate SIMM slot 7.

5.2.1.6 SIMM Register Programming Rules

SIMM starting address registers must be programmed in ascending order. For example, do not program slot 3 with a lower starting address than slot 2.

Missing or defective SIMMs are programmed out by making the start address of the missing or defective slot the same as the next slot. For example, if SIMM starting address registers 2 and 3 both have 01001b (72M) then SIMM slot 2 is inactive.

The seven SIMM registers are cleared to all zeros on power-on-reset . The SIMM top-of-memory register is cleared on power-on-reset to all zeros, indicating one 8M SIMM installed.

5.2.1.7 Reading the SIMM Registers

When the SIMM registers are read, the MR[7:5] bits indicate the register that has been decoded by the current read cycle. The MR[7:5] bits are incremented sequentially on each read, but software must not expect the first read to access register address 000b.

5.2.1.8 SIMM Starting Address Example #1

Table 5–3 shows the values of the SIMM registers if there is a 32M SIMM in slot 0 and two 8M SIMMs in slots 1 and 2.

Using Table 5–3 and following the rule from Section 5.2.1.4, an application address of 42M is greater than the starting address in mapping register 1, but it is also greater than the address in mapping register 2, so mapping register 1 is not a hit. Then 42M is greater than the starting address in mapping register 2 and less than the starting address in mapping register 3, so SIMM slot 2 (mapping register 2) is selected.

Table 5–3. Example #1 SIMM Mapping Register Setup

Mapping Register	Value of MR[4:0]	Comment
Mapping register 1	00100b	Starting address 32M, SIMM slot 0 is 32M.
Mapping register 2	00101b	Starting address 40M, SIMM slot 1 is 8M.
Mapping register 3	00110b	Starting address 48M, SIMM slot 2 is 8M.
Mapping register 4	00110b	Starting address 48M, SIMM slot 3 is empty.
Mapping register 5	00110b	Starting address 48M, SIMM slot 4 is empty.
Mapping register 6	00110b	Starting address 48M, SIMM slot 5 is empty.
Mapping register 7	00110b	Starting address 48M, SIMM slot 6 is empty.
Top-of-memory register	00101b	40M, top-of-memory minus 8M

5.2.1.9 SIMM Starting Address Example #2

Table 5–4 shows the value of the SIMM registers for a configuration with an 8M SIMM in slot 0, a 32M SIMM in slot1, an 8M SIMM in slot 2, no SIMM in slot 3, and a 32M SIMM in slot 4.

Using Table 5–4 and following the rule from Section 5.2.1.4, an address of 60M is greater than the starting address in mapping register 1, but it is also greater than the address in mapping register 2, so mapping register 1 is not a hit. Mapping registers 2 and 3 are disqualified based on the same logic. Then 60M is greater than the starting address in mapping register 4 and less than the starting address in register 5, so SIMM slot 4 (mapping register 4) is selected and RAS[4]# is asserted.

Table 5–4. Example #2 SIMM Mapping Register Setup

Mapping Register	Value of MR[4:0]	Comment
Mapping register 1	00001b	Starting address 8M, SIMM slot 0 is 8M.
Mapping register 2	00101b	Starting address 40M, SIMM slot 1 is 32M.
Mapping register 3	00110b	Starting address 48M, SIMM slot 2 is 8M.

Table 5–4. Example #2 SIMM Mapping Register Setup (Continued)

Mapping Register	Value of MR[4:0]	Comment
Mapping register 4	00110b	Starting address 48M, SIMM slot 3 is empty.
Mapping register 5	01010b	Starting address 80M, SIMM slot 4 is 32M.
Mapping register 6	01010b	Starting address 80M, SIMM slot 5 is empty.
Mapping register 7	01010b	Starting address 80M, SIMM slot 6 is empty.
Top-of-memory register	01001b	72M, top-of-memory minus 8M

5.2.2 Programming The System Setup Register

Figure 5–1 shows the system setup register. Access to the system setup register uses the same technique and data paths as accessing the SIMM mapping registers (See Section 5.2.3), by asserting MC_SETUP#, but with CPU_ADDR[31] high.

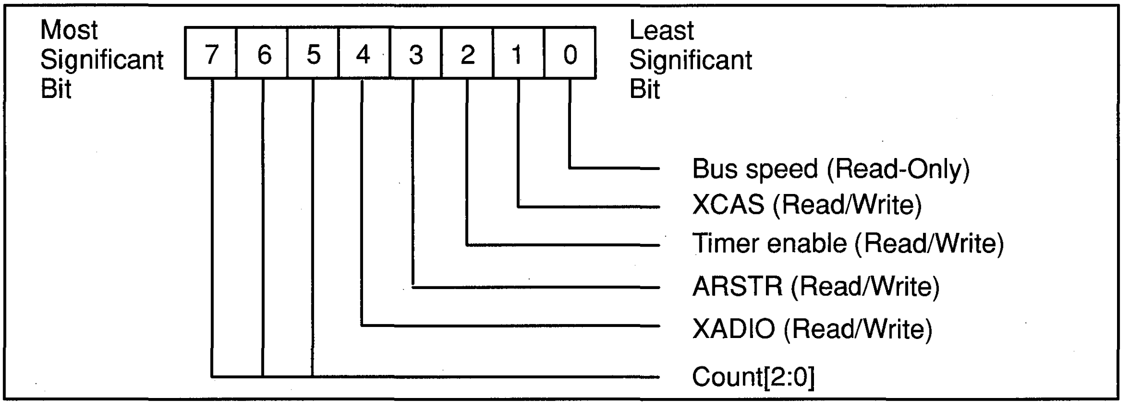


Figure 5–1. The System Setup Register

5.2.2.1 The Bus Speed Setting in the System Setup register

Bus speed is used to indicate the speed difference between the local and PCI buses. This bit is a read-only bit. It is set two processor clocks after power-on-reset (POR). If the processor clock is twice as fast as the PCI clock, this bit is set high. If the processor clock is equal to the PCI clock, it is low.

5.2.2.2 The XCAS (Extended CAS#) Setting in the System Setup register

XCAS extends the timing of the CAS[7:0]# lines by one additional 60X CPU bus clock cycle in order to propagate memory data through the 653 Buffer to the 60X CPU data bus at 66MHz with worst-case delays. This bit is read-write and set to 1 (extended) at POR. The XCAS bit can be programmed to 0 for systems with slower CPU bus speeds.

5.2.2.3 The Timer Enable Setting in the System Setup register

The timer enable bit controls the internal 60us timeout counter used for master abort on PCI cycles where the bus hangs. This bit is read-write and set to 1 (timer enabled) following power-on-reset.

5.2.2.4 The ARSTR Setting in the System Setup register

ARSTR enables the 654 to precharge ARTRY# (drive it high for one CPU_CLK cycle before trisataing it) on the CPU bus after a snoop hit event has occurred. This bit is read-write and set to 1 (precharge enabled) at power-on-reset. Normally only one CPU bus device is allowed to precharge ARTRY#.

5.2.2.5 The XADIO Setting in the System Setup register

XADIO is used to delay asserting PCI_IRDY# by one PCI clock during CPU write access to PCI. This bit is read-write and set to 1 (delay PCI_IRDY#) at power-on-reset. Some systems may be able to program this bit to 0 for slightly better performance.

5.2.2.6 The Count[2:0] Counter in the System Setup register

Count[2:0] is a 3-bit internal counter that changes state for each access to the SIMM mapping registers. These bits are read-only. The power-on state is 000b.

5.2.2.7 Bus Speed and XCAS Settings in the System Setup register

Table 5–5 shows the possible settings for BusSpeed and recommended XCAS for various clock speeds. (Using 70ns, 168-pin SIMMs.)

Table 5–5. System Setup Register Settings

60X CPU Local Bus	PCI Bus	Bus Speed	XCAS
25MHz	25MHz	0	don't care
33MHz	33MHz	0	don't care
40MHz	20MHz	1	0
50MHz	25MHz	1	0
66MHz	33MHz	1	1

5.2.3 Accessing the SIMM Registers and the System Setup Register

The SIMM mapping registers and the system setup register are located inside the 654 Controller, which is not connected to any data bus. Read and write data is passed to and from the 654 Controller registers over the CAS[7:0]# lines. Figure 5–2 shows how this data path is implemented in the example system, and Figure 5–3 shows the data paths and steering logic inside the 654. (Note that there is a minimum time delay required from any change in MC_SETUP# to the initiation of any memory or PCI bus transaction.)

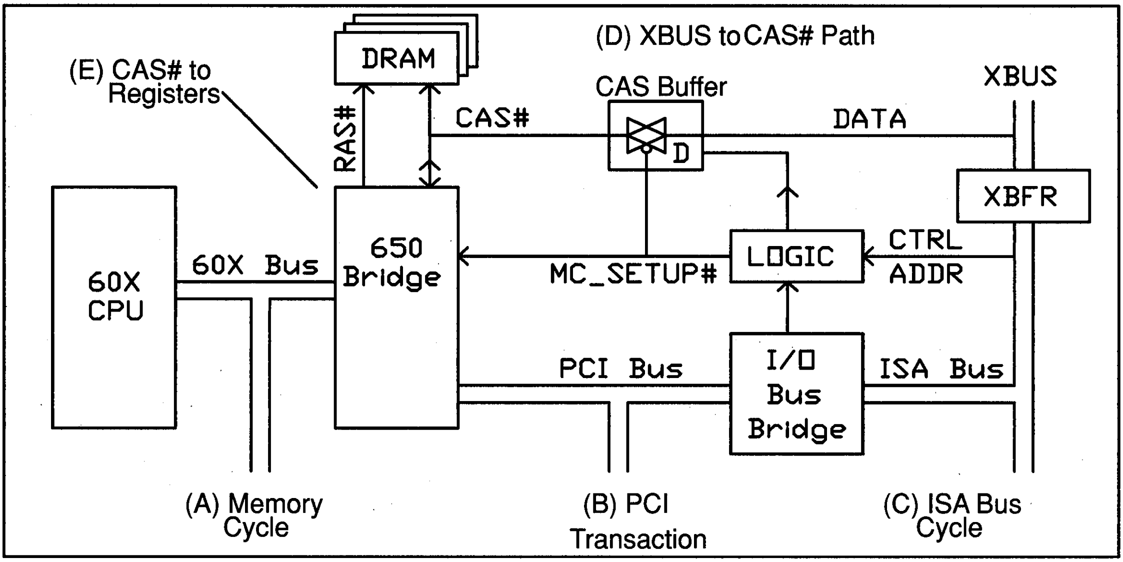


Figure 5–2. 650 Register Access Pathway in the Example System

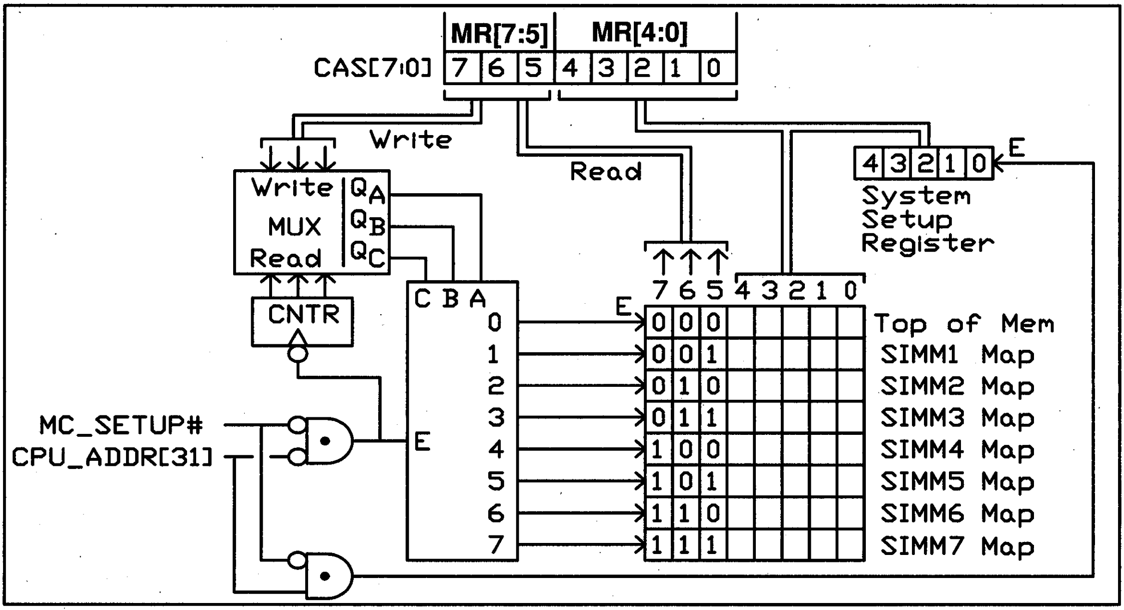


Figure 5–3. 654 Setup Register Data Paths and Steering Logic

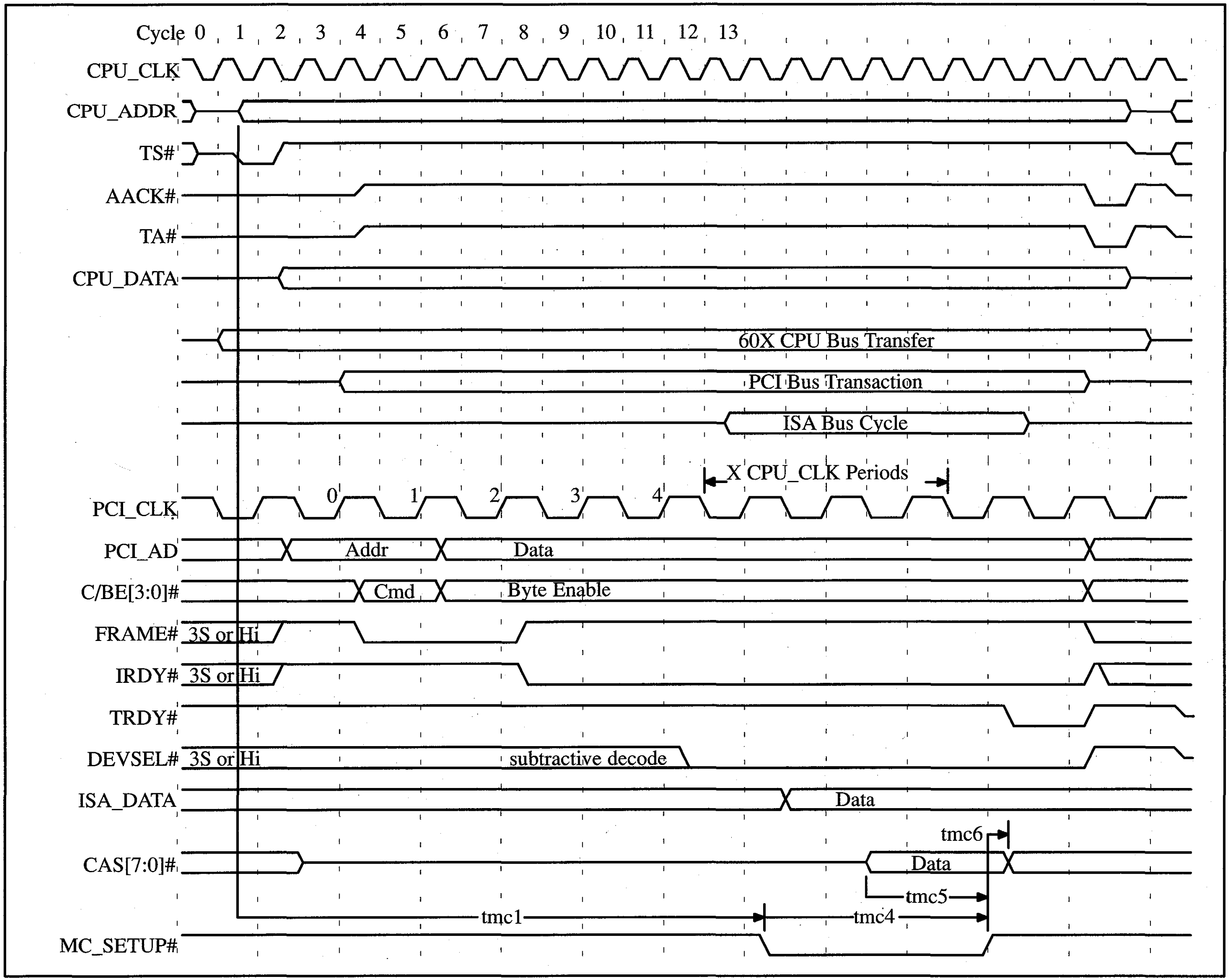


Figure 5–4. 650 Register Write Timing Diagram

5.2.3.1 SIMM Register and Setup Register Writes

In the example system, writing to the top of memory register requires the following steps (example system specific information is shown in italics);

1. In Figure 5–4 cycles 0 and 1, the 60X begins a store byte instruction to the correct address, with the register data in the low-order byte (see Figure 5–2 (A)). The 650 decodes the transfer and begins pacing the 60X CPU via the CPU bus. The 654 also tri-states the CAS# lines, sends RASHI/CASLO high and deasserts the RAS# lines.
2. In Figure 5–4 cycle 4, the 650 begins a PCI bus single-beat memory write transaction, to the I/O bus bridge. The I/O bus bridge decodes and claims the transaction (PCI subtractive decode protocol) in cycle 12 and paces the 650 via the PCI bus (see Figure 5–2 (B)).
3. The I/O bus bridge begins an ISA bus memory write cycle, and controls the XBFR buffers (see Figure 5–2 (C)).
4. The external logic decodes the ISA bus cycle, sets the direction of the CAS# buffer, and asserts MC_SETUP# (see Figure 5–2 (D)), which also causes the CAS# buffer to drive the data onto CAS[7:0]#.
5. Inside the 650 (see Figure 5–3), decode logic causes the read/write MUX to pass CAS[7:5]# to the register selector which is enabled by MC_SETUP# and CPU_ADDR[31] both being low. This enables the selected register to latch in the data from CAS[4:0]#. The data is latched on the rising edge of MC_SETUP#.
6. The I/O bus bridge then completes the ISA bus cycle with no wait states. The deassertion of MC_SETUP# disables the 654 setup register steering logic and turns off the CAS# buffer. Note that the memory controller leaves RAS#[7:0] high at the end of this operation.
7. The I/O bus bridge then asserts TRDY# to complete the PCI bus transaction.
8. The 650 completes the PCI bus transaction and signals AACK# and TA#.
9. The 60X CPU then completes the 60X CPU bus transfer.

Table 5–6 contains the timing information referenced by the timing diagrams.

Table 5–6. SIMM Register Access Timing Chart

Symbol	Description	Value
tmc1	Setup, CPU_ADDR valid to MC_SETUP# fall	0 Min
tmc2	Delay, MC_SETUP# fall to CAS[7:0]# valid	3 CPU_CLK Max
tmc3	Output hold, MC_SETUP# rise to CAS[7:0]# invalid	2 CPU_CLK Max
tmc4	Minimum pulse width, MC_SETUP#	5 CPU_CLK Min
tmc5	Setup, CAS[7:0]# data valid to MC_SETUP# fall	4 CPU_CLK Min
tmc6	Input hold, MC_SETUP# rise to CAS[7:0]# invalid	0 Min

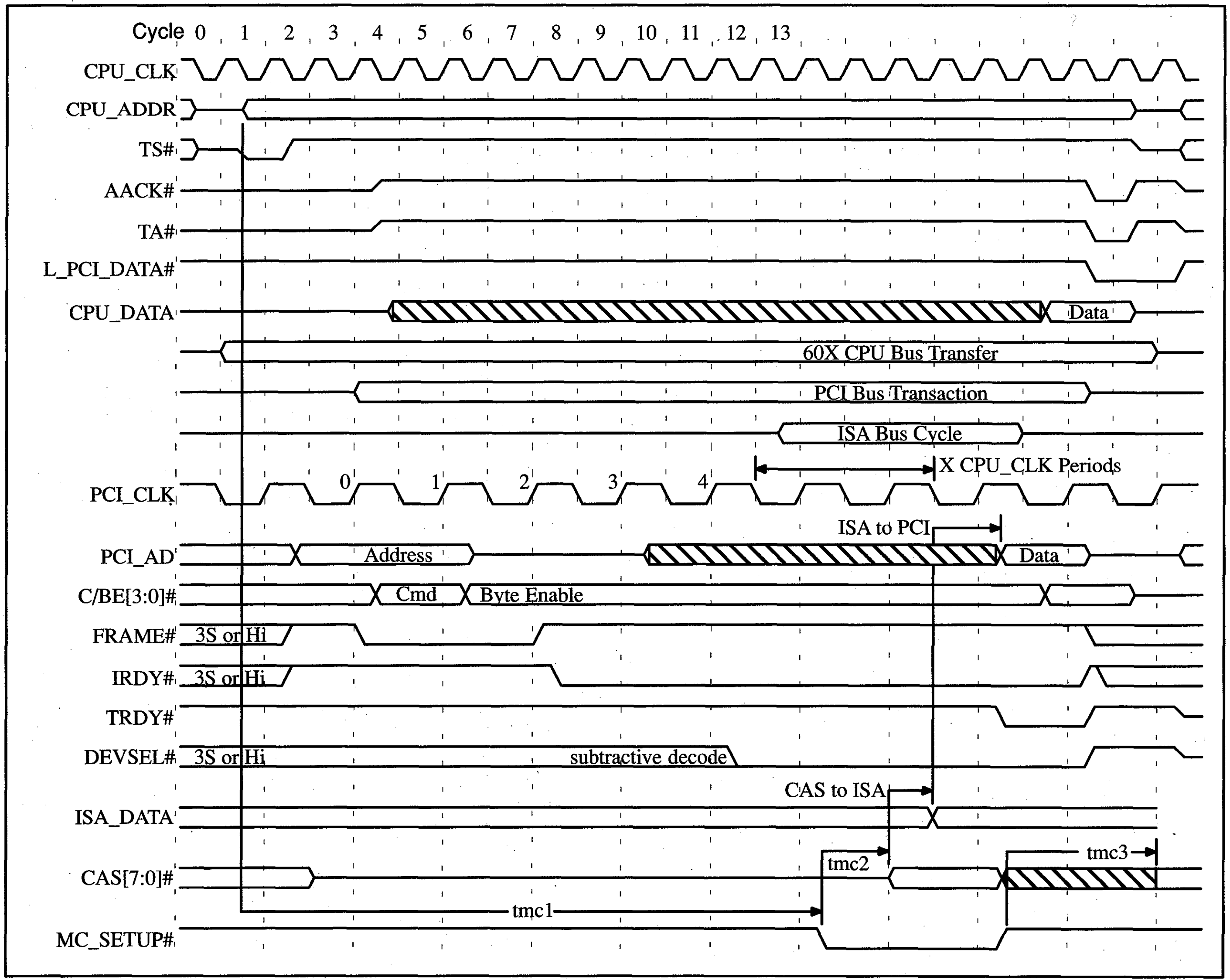


Figure 5–5. 650 Register Read Timing Diagram

5.2.3.2 SIMM Register and Setup Register Reads

Reading the 654 Controller registers is similar to writing to them. There are two major differences:

- The buffers are turned around to transmit data to the CPU.
- During writes to the SIMM registers (and all accesses to the system setup register), the accessed register is uniquely specified. When reading the SIMM registers, a 3-bit counter identifies the accessed register.

In Figure 5–3, the read/write MUX is shown passing the output of the 3 bit counter to the register selector during read operations. The value of the counter determines which register is selected. The state of the counter can not be set directly.

The identity of the register is hardwired into the upper three bits of each register. When the register is read, three of the bits identify the register, and the other five bits contain the data. The 3-bit counter is incremented at the end of each register read transaction. Performing eight reads from the registers yields the data from all of the memory registers. This counter is set to zero during power-on reset.

In the example system, reading a SIMM register requires the following steps:

1. In Figure 5–5 cycles 0 and 1, the 60X CPU begins a load byte instruction to the correct address (see Figure 5–2 (A)). The 650 decodes the transfer and begins pacing the 60X CPU via the CPU bus. The 654 Controller tri-states the CAS# lines, sends RASHI/CASLO high and deasserts the RAS# lines.
2. In Figure 5–5 cycle 4, the 650 Bridge begins a PCI bus single-beat memory read transaction to the I/O bus bridge. The I/O bus bridge decodes and claims the transaction (PCI subtractive decode protocol) in cycle 12 and paces the 650 via the PCI bus (see Figure 5–2 (B)).
3. The I/O bus bridge begins an ISA bus memory read cycle, and controls the XBFR buffers (see Figure 5–2 (C)).
4. The external logic decodes the ISA bus cycle, points the CAS# buffer toward the XBUS, and asserts MC_SETUP# (see Figure 5–2 (D)), which also causes the CAS# buffer to drive the data onto the XBUS.
5. Inside the 650 (see Figure 5–3) decode logic causes the read/write MUX to pass the output of the 3-bit counter to the register selector which is enabled by MC_SETUP# and CPU_ADDR[31] both being low. This selects and enables one of the registers to drive its contents onto CAS[7:0]# (see Figure 5–5 delay tmc2). Note that CAS[7:5]# contain the register ID bits, and CAS[4:0]# contain the register data.
6. The contents of CAS[7:0]# now flow thru the CAS Buffer and the XBFR (see delay CAS to ISA), and onto the ISA bus data lines. The I/O bus bridge then latches the data and completes the ISA bus cycle with no wait states.
7. The I/O bus bridge then places the data on the PCI_AD lines (see delay ISA to PCI), and signals TRDY# to the 650 Bridge. External logic negates MC_SETUP#, disabling the 654 setup register steering logic and turning off the CAS# buffer. Note that the memory controller leaves RAS#[7:0] high at the end of this operation.
8. The 650 completes the PCI bus transaction, supplies the data to the 60X CPU, and signals AACK# and TA#.
9. The 60X CPU then completes the 60X CPU bus transfer.

5.2.3.3 Register Reads in the Example System

In the example system, reading a SIMM register starts with a 60X CPU load byte operation (TT[0:3] = 0101, TSIZ[0:2] = 001) to 60X bus address 8000 0820h. This produces a PCI bus single-beat I/O read transaction (C/BE#[3:0] = 0010), to the I/O bus bridge (PCI address = 0000 0820h), which produces an ISA bus I/O read cycle to ISA bus address 0820h.

5.2.3.4 Register Writes in the Example System

In the example system, writing to a SIMM register starts with a 60X CPU store byte operation (TT[0:3] = 0001, TSIZ[0:2] = 001) to 60X bus address 8000_0820h. This produces a PCI bus single-beat I/O write transaction (C/BE#[3:0] = 0011), to the I/O bus bridge (PCI address = 0000_0820h), which produces an ISA bus I/O write cycle to ISA bus address 0820h. (Note that the example system address of the System Setup Register is 8000 0821h.)

5.2.4 Programming the Flash ROM Lock-Out Bit (W/O)

Writing to an address in the range of 4G – 8M to 4G (FF80 0001h to FFFF FFFFh) with the low-order bit of the CPU address set to 1 turns on the FLASH lock-out bit. Once this bit is set (to 1), subsequent ROM write attempts are locked out and TA# is asserted to terminate the cycles. No error indication is given. This bit can only be cleared with a power-on-reset. The initial state of the lockout bit is unlocked (0).

5.3 Little-Endian and Big-Endian Addressing Considerations

Internally, the 60X CPU always operates with big-endian addresses, data, and instructions. A mode bit can be set in the 60X CPU that enables a little-endian addressing mode for CPU bus activity. The 650 Bridge works with the little-endian mode addresses on the 60X CPU bus to produce a true little-endian memory and I/O map.

In big-endian mode the most-significant byte of a data field is stored in the lowest numbered address of the field. In little-endian mode the most-significant byte of a data field is stored in the highest numbered address of the field. The 650 Bridge supports both big-endian and little-endian addressing modes. Munging in the 60X CPU combined with byte swapping and unmunging in the 650 Bridge allows data addressing in main memory and on the PCI bus in true little-endian format.

When the 60X CPU is attempting to access system memory (DRAM), the 654 Controller decodes TBST#, TSIZ[0:2], CPU_ADDR[29:31], and LE_MODE_SEL# to determine the proper CAS# lines to assert for the memory transfer. For 60X CPU cycles to a PCI target, the value of PCI_C/BE[3:0]# is based on TSIZ[0:2], CPU_ADDR[29:31], and LE_MODE_SEL# to determine the PCI byte enables to be asserted.

The *PowerPC 601 RISC Microprocessor User's Manual, MPR601UMU-02,* contains a discussion of the implications of endian modes from the perspective of the 60X CPU.

5.3.1 60X CPU Addressing in Big-Endian Mode

When the 60X CPU is operating in big-endian mode, all addresses and data pass through the 650 Bridge without byte swapping or unmunging. The system memory representation and the PCI bus representation of data is big-endian.

5.3.2 60X CPU Address Munging in Little-Endian Mode

When the 60X CPU is operating in little-endian mode, CPU_ADDR[29:31] is munged as shown in Table 5–7. A different XOR value is used for one-byte, two-byte, and four-byte transfers. Eight-byte transfers do not munge or unmunge CPU_ADDR[29:31].

The combinations in Table 5–7 that are marked n/a are unaligned transfers that cause alignment exceptions in the 60X CPU and therefore do not generate 60X bus cycles.

Table 5–7. CPU_ADDR[29:31] Munging for Little-Endian Mode

CPU_ADDR[29:31] before munge	1-byte XOR 111	2-bytes XOR 110	4-bytes XOR 100	8-bytes (no change)
000	111	110	100	000
001	110	n/a	n/a	n/a
010	101	100	n/a	n/a
011	100	n/a	n/a	n/a
100	011	010	000	n/a
101	010	n/a	n/a	n/a
110	001	000	n/a	n/a
111	000	n/a	n/a	n/a

5.3.3 650 Bridge Address Unmunging in Little-Endian Mode

The 653 Buffer unmunges the address produced by the 60X processor as shown in Table 5–8.

Note that the unmunge of the three low-order CPU address lines is the same when the CPU addresses the PCI as it is when the CPU addresses system memory or ROM. In the cases of memory and ROM the transform has no effect in the 653 Buffer. A similar transform in the 654 Controller determines which bytes are addressed during memory writes and which byte enables are asserted during PCI transactions.

Table 5–8. Three Low-Order Address Bit Unmunge

TSIZ[0:2]	Big-Endian Mode	Little-Endian Mode
000	none	none
001	none	XOR 3 low-order bits with 111
010	none	XOR 3 low-order bits with 110
011	none	N/A
100	none	XOR 3 low-order bits with 100
101	none	N/A
110	none	N/A
111	none	N/A

5.3.4 Byte Swapping for Endian Compatibility

The 653 Buffer uses a byte swapper to reverse the order of bytes read or written by the CPU when the 650 is in little-endian mode. The action of the byte swapper combined with the unmunging of the low-order bits of the effective address, results in data storage in system memory in true little-endian order. (Also see Section 5.3.8.)

The storage location of single byte loads and stores is unaffected by the endian selection. A single byte written or read to address 0000 1013h always goes to that memory location, regardless of the current endian mode.

In little-endian mode, transfers of half-words, words, and double-words result in a reversal of the bytes within the half-word, word, or double-word. Table 5–9 illustrates this byte swapping. As shown in the table, the bits within individual bytes are not swapped. The byte swapper examples in Table 5–9 are reversible—output from the 60X CPU (store instructions) is exactly reversed or swapped back for input (load instructions).

Table 5–9. Endian Formats from the Byte Swapper

Data in the 60X CPU	Big-Endian Output	Little-Endian Output
ABCDh	ABCDh	CDABh
1234 5678h	1234 5678h	7856 3412h
1234 5678 9ABC DEF0h	1234 5678 9ABC DEF0h	F0DE BC9A 7856 3412h

5.3.5 Unmunging and Byte Swapping for System Memory or PCI Writes

The 650 Bridge is designed to implement a memory model that stores big-endian and little-endian data in system memory or to the PCI bus in an exact representation of the required endian mode. Little-endian data is stored in little-endian mode and big-endian data is stored in big-endian mode. Therefore, data read from or written to external media, like disk drives, does not require any extra manipulation. The following examples in this section illustrate the process of little-endian data manipulation.

The 60X CPU and the 650 Bridge cooperate through munging, byte swapping, and unmunging to organize the system memory in little-endian mode. In little-endian mode, the 60X CPU munges the three low-order address bits to send the bytes to the correct byte lanes in the byte swapper in the 650 Bridge. The byte swapper then swaps the eight-byte CPU data bus. The byte swap restores the data to the byte lanes where it was prior to the munge, and the unmunge restores the correct address for the memory write.

The sequence of operations is as follows:

1. The address is munged by the 60X CPU to place the data in the correct byte lanes for the byte swapper.
2. The byte swapper swaps the data, placing the reversed bytes back at their original address range.
3. The unmunger restores the address to its original value so that the swapped bytes can be accessed from the output side of the byte swapper.

5.3.5.1 An Example of a One-Byte Little-Endian Store Instruction

In the following example, the addresses of data only refer to the low-order three bits of an address because byte swaps in little-endian mode can only occur within an eight-byte double word. If the system is in little-endian mode, a one-byte store instruction executed by the 60X CPU can cause the following steps to occur:

1. The 60X CPU executes a store byte instruction—`store 31h to 010b.`
2. The effective address is XOR'd with 111b to—`store 31h to 101b.`
3. The 650 Bridge swaps the bytes as shown in Figure 5–6.
4. The 650 Bridge XOR's the effective address with 111b to `010b.`
5. The 654 Controller asserts CAS[2]# (see Table 5–12) or PCI_C/BE[2]# (see Table 5–18 in Section 5.5.3.6).
6. The byte 31h is written to 010b.

Note that the same instruction executed in big-endian mode also writes to 010b. Single byte reads and writes are stored in exactly the same addresses in big-endian and little-endian modes.

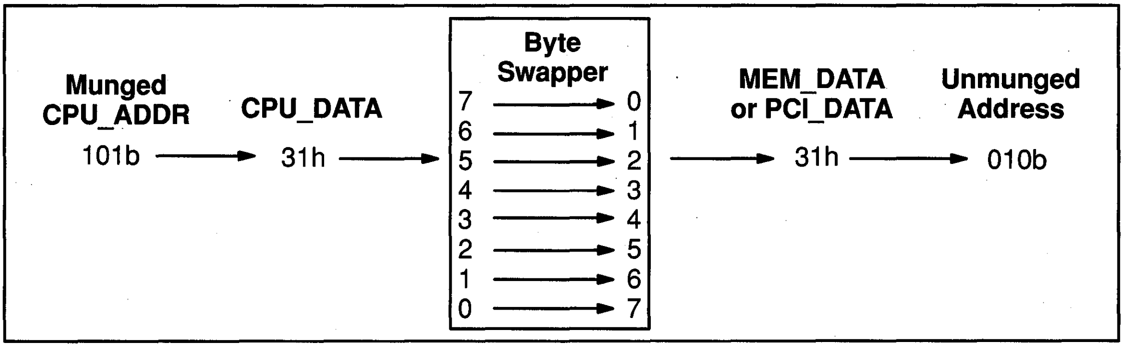


Figure 5–6. Byte Swapper Operation for Example of a Store Byte Instruction

5.3.5.2 An Example of a Two-Byte Little-Endian Store Instruction

In the following example, the addresses of data only refer to the low-order three bits of an address because byte swaps in little-endian mode can only occur within an eight-byte double word. If the system is in little-endian mode, a two-byte store instruction executed by the 60X CPU can cause the following steps to occur:

1. The 60X CPU executes a store half-word instruction—`store 3132h to 010b.`
2. The effective address is XOR'd with 110b to—`store 3132h to 100b.`
3. The 650 Bridge swaps the bytes as shown in Figure 5–7.
4. The 650 Bridge XOR's the effective address with 110b to `010b.`
5. The 654 Controller asserts CAS[2]# and CAS[3]# (see Table 5–12) or PCI_C/BE[3]# and PCI_C/BE[2]# (see Table 5–18 in Section 5.5.3.6).
6. The two bytes 32h and 31h are written to 010b and 011b respectively.

Note that the same instruction executed in big-endian mode also writes to 010b and 011b, but the two bytes are written in big-endian mode—31h and 32h respectively.

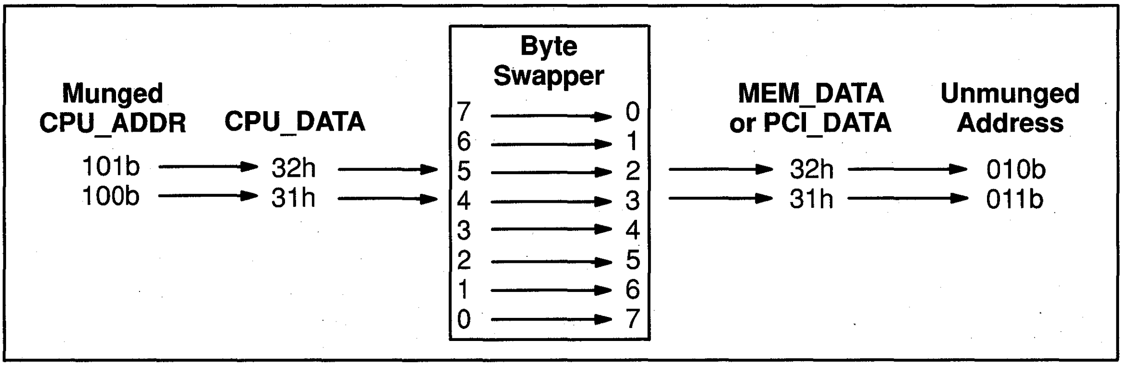


Figure 5–7. Byte Swapper Operation for Example of a Store Half-Word Instruction

5.3.5.3 An Example of a Four-Byte Little-Endian Store Instruction

In the following example, the addresses of data only refer to the low-order three bits of an address because byte swaps in little-endian mode can only occur within an eight-byte double word. If the system is in little-endian mode, a four-byte store instruction executed by the 60X CPU can cause the following steps to occur:

1. The 60X CPU executes a store word instruction—`store 31323334h to 100b.`
2. The effective address is XOR'd with 100b to—`store 31323334h to 000b.`
3. The 650 Bridge swaps the bytes as shown in Figure 5–8.
4. The 650 Bridge XOR's the effective address with 100b to `100b`.
5. The 654 Controller asserts CAS[4]#, CAS[5]#, CAS[6]# and CAS[7]# (see Table 5–12) or PCI_C/BE[3]# through PCI_C/BE[0]# (see Table 5–18 in Section 5.5.3.6).
6. The four bytes 34h, 33h, 32h, and 31h are written to 100b, 101b, 110b, and 111b respectively.

Note: The data doubler within the 653 Buffer places the four-byte output on both halves of the 64-bit output bus so that the PCI_C/BE[3:0] gets the data regardless of which four-byte word is addressed. See Appendix C.

Note that the same instruction executed in big-endian mode also writes to 100b through 111b, but the four bytes are written in big-endian mode—31h, 32h, 33h, and 34h respectively.

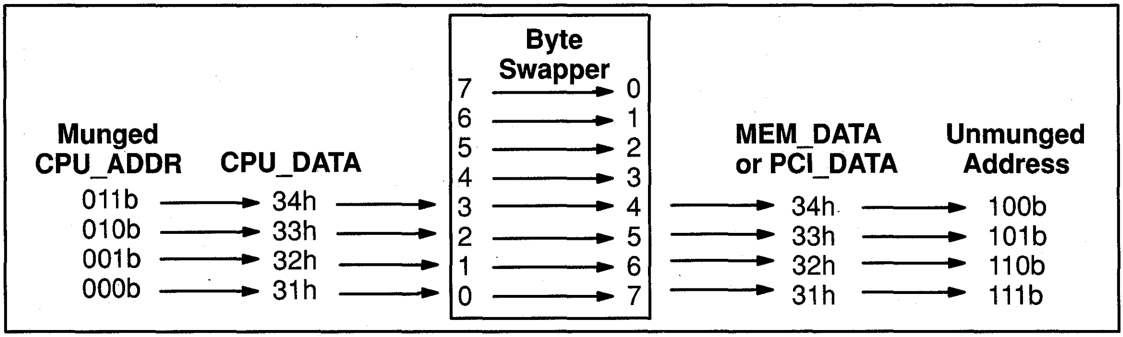


Figure 5–8. Byte Swapper Operation for Example of a Store Word Instruction

5.3.5.4 An Example of an Eight-Byte Little-Endian Store Instruction

In the following example, the addresses of data only refer to the low-order three bits of an address because byte swaps in little-endian mode can only occur within an eight-byte double word. If the system is in little-endian mode, an eight-byte store instruction executed by the 60X CPU can cause the following steps to occur:

1. The 60X CPU executes a store floating-point double instruction—
   `store 31323334 35363738h to 000b.`
2. The effective address is not XOR'd for an eight byte store—
   `store 31323334 35363738h to 000b.`
3. The 650 Bridge swaps the bytes as shown in Figure 5–9.
4. The 650 Bridge does not XOR the effective address—`000b.`
5. The 654 Controller asserts all of CAS[7:0]# (see Table 5–12). This transaction cannot occur on the PCI bus because PCI can only accept up to four-byte transfers.
6. The eight bytes 38h, 37h, 36h, 35h, 34h, 33h, 32h, and 31h are written to 000b through 111b respectively.

Note that the same instruction executed in big-endian mode also writes to 000b through 111b, but the eight bytes are written in big-endian mode—31h, 32h, 33h, 34h, 35h, 36h, 37h, and 38h respectively.

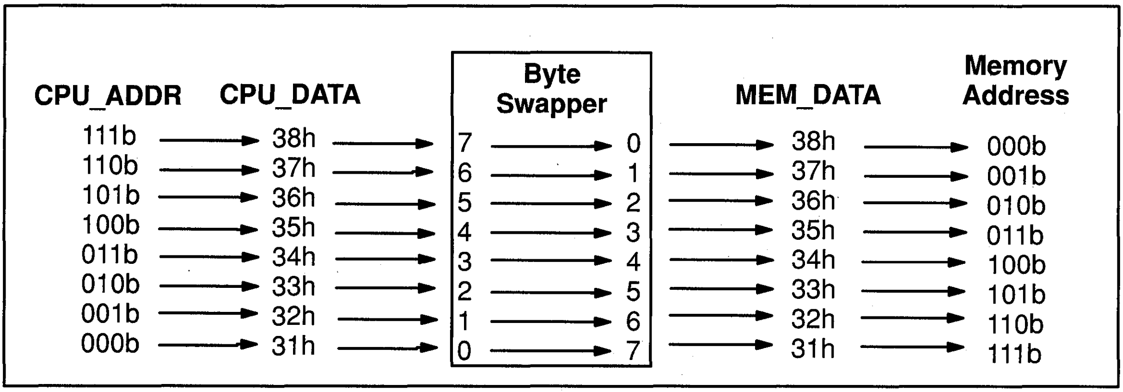

CPU_ADDR
CPU_DATA
Byte Swapper
MEM_DATA
Memory Address
111b → 38h → 7 → 0 → 38h → 000b
110b → 37h → 6 → 1 → 37h → 001b
101b → 36h → 5 → 2 → 36h → 010b
100b → 35h → 4 → 3 → 35h → 011b
011b → 34h → 3 → 4 → 34h → 100b
010b → 33h → 2 → 5 → 33h → 101b
001b → 32h → 1 → 6 → 32h → 110b
000b → 31h → 0 → 7 → 31h → 111b

Figure 5–9. Byte Swapper Operation for a Store Floating-Point Double Instruction

5.3.6 Unmunging and Byte Swapping for System Memory and PCI Reads

For 60X CPU system memory reads in little-endian mode, the munging and byte swapping occur exactly as they do for system memory writes. (See Section 5.3.5.) The 650 Bridge reads eight bytes from system memory regardless of the size of the transfer, therefore CAS[7:0]# is always 0000 0000b for a 60X CPU system memory read. PCI reads are a maximum of four bytes aligned in a word (the The data doubler within the 653 Buffer places the four-bytes on both halves of the 64-bit bus (see Appendix C). The TSIZ[0:2] and CPU_ADDR[29:31] signals determine the byte lanes that are accessed by the CPU.

The following examples apply equally whether the read data is directly from system memory or from a cache or from PCI. In each case, the bytes are swapped before they reach the 60X CPU. In the case of cached data, the bytes were swapped at the time the data cache was originally stored in the cache. This means that cached data is byte-swapped, including instruction fetches.

5.3.6.1 An Example of a Two-Byte Little-Endian Load Instruction

In the following example, the addresses of data only refer to the low-order three bits of an address because byte swaps in little-endian mode can only occur within an eight-byte double word. If the system is in little-endian mode, a two-byte load instruction executed by the 60X CPU can cause the following steps to occur:

1. The 60X CPU executes a two-byte load instruction—`load half-word at 010b.`
2. The effective address is XOR'd with 110b to—`load half-word at 100b.`
3. The system memory is read based on CPU_ADDR[0:28]. CAS[7:0]# is all asserted.
4. The 650 Bridge swaps the bytes as shown in Figure 5–10.
5. The two bytes are in big-endian order as 31h and 32h.

Note that the same instruction executed in big-endian mode also reads addresses 010b and 011b, but the two bytes are read unswapped from 010b and 011b in big-endian mode—31h and 32h respectively—because the data is stored in memory in big-endian mode (the byte swapper is not active when the 650 Bridge is in big-endian mode). Cached data is read correctly because the 64-bit double-words are byte swapped as they are loaded into the cache.

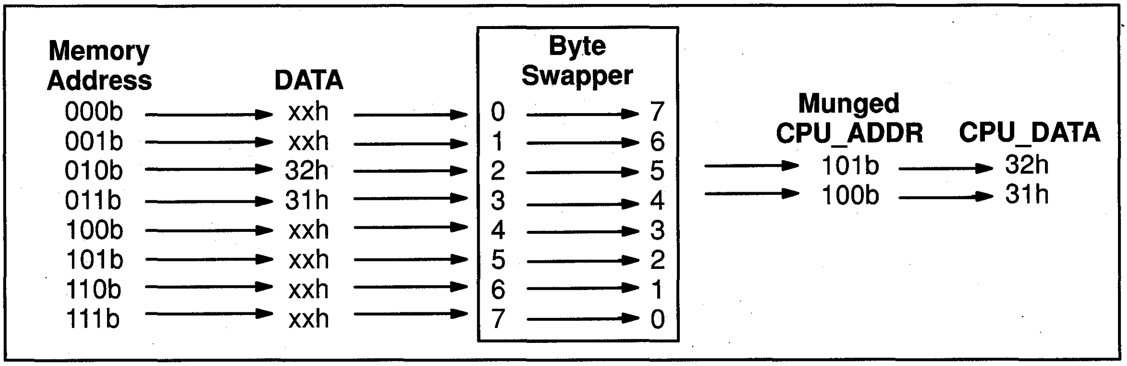


Figure 5–10. Byte Swapper Operation for Example of a Load Half-Word Instruction

5.3.6.2 An Example of a Four-Byte Little-Endian Load Instruction

In the following example, the addresses of data only refer to the low-order three bits of an address because byte swaps in little-endian mode can only occur within an eight-byte double word. If the system is in little-endian mode, a four-byte load instruction executed by the 60X CPU can cause the following steps to occur:

1. The 60X CPU executes a four-byte load instruction—`load word at 100b.`
2. The effective address is XOR'd with 100b to—`load word at 000b.`
3. The system memory is read based on CPU_ADDR[0:28]. CAS[7:0]# is all asserted.
4. The 650 Bridge swaps the bytes as shown in Figure 5–11.
5. The four bytes are in big-endian order as 31h, 32h, 33h, and 34h.

Note that the same instruction executed in big-endian mode also reads 100b through 111b but the bytes are stored and therefore read back in big-endian order.

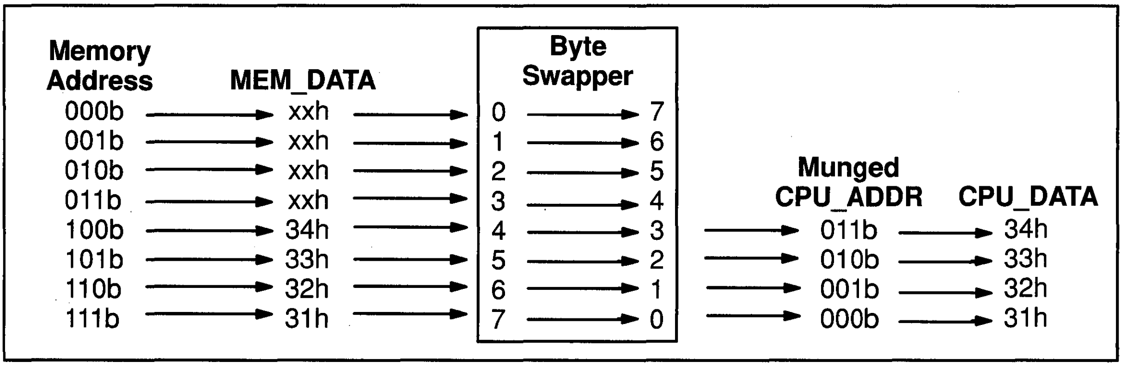


Figure 5–11. Byte Swapper Operation for Example of a Load Word Instruction

5.3.6.3 An Example of an Eight-Byte Little-Endian Load Instruction

In the following example, the addresses of data only refer to the low-order three bits of an address because byte swaps in little-endian mode can only occur within an eight-byte double word. If the system is in little-endian mode, an eight-byte load instruction executed by the 60X CPU can cause the following steps to occur:

1. The 60X CPU executes an eight-byte load floating-point double instruction— `load double word at 000b.`
2. The effective address is not XOR'd—`load double word at 000b.`
3. The system memory is read based on CPU_ADDR[0:28]. CAS[7:0]# is all asserted.
4. The 650 Bridge swaps the bytes as shown in Figure 5–12.
5. The eight bytes are in big-endian order as 31h, 32h, 33h, 34h, 35h, 36h, 37h, 38h.

Note that the same instruction executed in big-endian mode also reads 000b through 111b but the data bytes do not move throught the swapper.

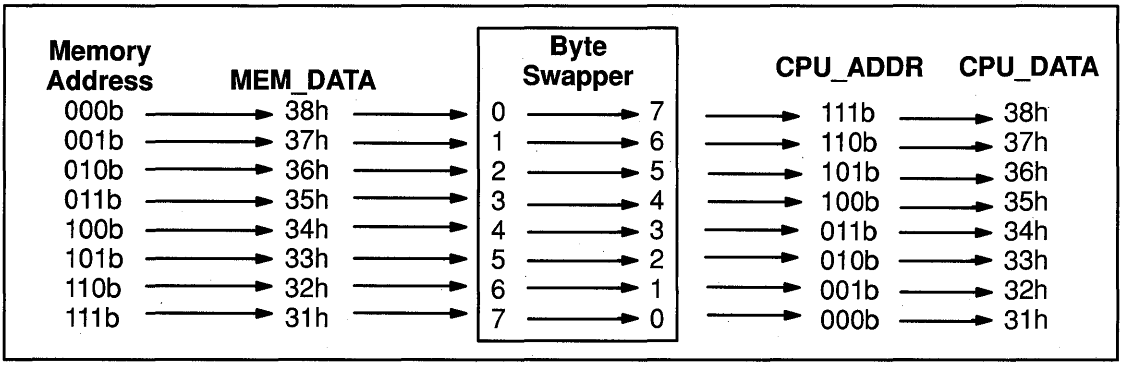

Memory Address
MEM_DATA
Byte Swapper
CPU_ADDR
CPU_DATA
000b → 38h → 0 → 7 → 111b → 38h
001b → 37h → 1 → 6 → 110b → 37h
010b → 36h → 2 → 5 → 101b → 36h
011b → 35h → 3 → 4 → 100b → 35h
100b → 34h → 4 → 3 → 011b → 34h
101b → 33h → 5 → 2 → 010b → 33h
110b → 32h → 6 → 1 → 001b → 32h
111b → 31h → 7 → 0 → 000b → 31h

Figure 5–12. Byte Swapper Operation for Example of Load Floating-Point Instruction

5.3.7 Instruction Fetches in Little-Endian Mode

Instruction fetches in little-endian mode work transparently to byte swap instruction words as the 60X CPU requires.

5.3.8 LE_MODE_REQ# Assertion on the 654 Controller

It is the responsibility of the system designer and programmer to ensure that the endian mode of the processor is synchronized with the endian mode of the 650 Bridge. The system designer must provide a means for the programmer to assert LE_MODE_REQ# to the 654 Controller so the 654 Controller can assert LE_MODE_SEL# to the 653 Buffer.

The 654 Controller samples LE_MODE_REQ# continuously, but it changes LE_MODE_SEL# only between bus transactions, while the bus is idle. This allows the LE_MODE_REQ# signal to be the output of an I/O port and guarantees that the endian selection will not change during the bus cycle that writes to the port.

The programmer must perform the code steps necessary to cause LE_MODE_REQ# to be asserted when the 60X CPU is switched to little-endian mode. LE_MODE_SEL# is switched in response to LE_MODE_REQ# when both the 60X bus and PCI bus are idle.

5.3.9 Exceptions in Little-Endian Mode

In little-endian mode, the 60X CPU does not support a number of instructions and data alignments that are allowed in big-endian mode. When the 60X CPU encounters one of these instructions in little-endian mode, it takes an internal alignment exception and does not produce an external bus cycle.

Some of the instructions that may not be supported in little-endian mode are as follows:

Unaligned loads and stores
LMW instruction
STMW instruction
Move assist instructions (*LSWI, LSWX, STSWI, STWX*)

Check the documentation for your 60X CPU to determine the instructions that are not supported in little-endian mode on your machine.

5.4 Memory Controller Operation

The memory controller supports the 60X CPU and PCI devices in both single-cycle and burst-mode accesses. The access time to system memory varies based on the setting of the bus speed and XCAS bits of the system setup register (see Section 5.2.2).

5.4.1 System Memory Timing

Table 5–10 shows the bus clock cycles for a variety of page hit and page miss scenarios. The first number in each column is the number of cycles for a single 64-bit read or write cycle counted from the assertion of TS# or PCI_FRAME#. The second, third, and fourth numbers are the number of cycles for each phase of a burst transaction. Processor to memory performance is measured in processor clocks. PCI to memory performance is measured in terms of the PCI clock.

In the column titled *Bus Speed and Extended CAS*, the bus is either 100% for when the CPU and PCI buses are running at the same rate, or 50% for when the PCI bus is running at half the rate of the CPU bus. The X under XCAS means that XCAS has no effect when the bus speed is 100%. See Sections 5.2.2.1 and 5.2.2.2.

A page miss to memory always occurs after a DRAM refresh cycle, after a PCI I/O or PCI configuration cycle, after the RAS timeout, and after a memory access outside the current 4K page boundary. During PCI burst accesses to system memory, the 654 Controller samples CPU_ADDR[19] two 60X CPU clocks after asserting BURST_CLK# to the 653 Buffer to determine whether a page miss has occurred.

Table 5–10. DRAM Memory Timings

Bus Speed and Extended CAS		CPU to Memory (in CPU clocks)				PCI to Memory (in PCI clocks)			
		DRAM Page Hit		DRAM Page Miss		DRAM Page Hit		DRAM Page Miss	
Bus	**XCAS**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**
100%	X	5–3–3–3	5–3–3–3	10–3–3–3	10–3–3–3	8–7–7–7	8–3–6–3	12–7–7–7	12–3–6–3
50%	0	6–4–4–4	6–4–4–4	12–4–4–4	12–4–4–4	5–4–4–4	5–3–4–3	8–4–4–4	8–3–4–3
50%	1	7–5–5–5	7–5–5–5	13–5–5–5	13–5–5–5	5–4–4–4	5–3–4–3	8–4–4–4	8–3–4–3

5.4.2 60X CPU to System Memory Burst-Mode Counting

60X CPU bursts to and from system memory use a linear count within a 32-byte cache sector. These bursts are initiated by the L1 cache for the purpose of filling or writing a 32-byte cache sector. The memory cycle can begin with any 8-byte double-word within a 32-byte aligned cache sector. (A cache sector always begins with an address that is a multiple of thirty-two.) After the first cycle of the burst, subsequent 8-byte double-words are transferred within a circular address range for the 32-byte cache sector. For example:

1. The 60X CPU requests a burst read beginning at address 0000 0010h.
2. The 650 Bridge transfers eight bytes from system memory address 0000 0010h.
3. The 650 Bridge increments the address to 0000 0018h and transfers eight bytes.
4. The 650 Bridge increments the address to 0000 0000h and transfers eight bytes.
5. The 650 Bridge increments the address to 0000 0008h and transfers eight bytes.

Notice that in step 4 the 650 Bridge incremented the address within the 32-byte sector. The 60X CPU expects burst transfers to and from system memory to follow this logic.

5.4.3 PCI to System Memory Burst Mode Transfers

PCI bursts to system memory count sequentially from the beginning address of the burst and can continue indefinitely.

PCI bursts to and from system memory (DRAM) are supported without special restrictions. PCI bursts can start at any byte address and end at any byte address. The 650 Bridge arbitration logic ensures that the PCI device does not hog the bus.

The memory controller monitors CPU_ADDR[29] and the byte enable signals on PCI_C/BE[3:0]# to determine the bytes to transfer. The memory controller samples CPU_ADDR[19] two 60X CPU clocks after asserting BURST_CLK# to the 653 Buffer to determine whether a page miss has occurred. The 653 Buffer places the translated PCI address on the 60X address bus during this operation.

The PCI specification allows the PCI_C/BE[3:0]# byte enables to change on each data phase. PCI devices use this feature of the PCI specification on the first or last transfer of a burst. The memory address increments by four on each beat of the PCI burst, therefore all intermediate beats of a burst contain four bytes of data.

5.4.4 System Memory Parity Generation and Checking

The 653 Buffer continuously generates parity for the memory data lines. The parity bits are latched and written when the 653 Buffer writes data to memory.

The 653 Buffer continuously checks parity on the memory data lines, driving MEM_PAR_GOOD continuously based on the current data. The 654 Controller samples MEM_PAR_GOOD at the appropriate time in the memory read cycle to verify parity and assert MEM_PAR_ERR# if a valid error occurs.

The parity of data read from the L2 cache is checked by means of the DPE# signal from the 60X CPU. See Section 5.7.3.

5.4.5 RAS# and CAS# Address Assignments

When the 654 Controller asserts CPU-ADDR_SEL# to the 653 Buffer, the address presented to the memory address output pins depends on the input signals DRAMX9HI/X10LO and RASHI/CASLO. If RASHI/CASLO is low then a CAS# address is presented and if it is high a RAS# address is presented. Table 5–11 shows the MEM_ADDR[12:0] values that are asserted from the 653 Buffer internal address bus (which is numbered in little-endian order), depending on the value of DRAMX9HI/X10LO and RASHI/CASLO.

DRAMX9HI/X10LO provides support for X9 and X10 memory SIMMs. This signal comes from system logic or a strapping pin. The 650 Bridge does not control or dynamically switch this signal.

Table 5–11. RAS and CAS Address Assignments

DRAMX9HI/ X10LO	RASHI/ CASLO	Cycle Type	Internal address gated to MEM_ADDR[12:0]
1	1	RAS	ADDR[23:12]
1	0	CAS	0, 0, 0 ADDR[11:3]
0	1	RAS	ADDR[24:13]
0	0	CAS	0, ADDR[24], ADDR[12:3]

5.4.6 RAS[7:0]# Line Selection of SIMM Slots

Section 5.2.1.2 explains how the SIMM memory registers are configured to control assertion of the RAS[7:0]# lines. Each SIMM slot has a corresponding RAS[7:0]# line. RAS[7]# corresponds to SIMM slot 7. RAS[0]# corresponds to SIMM slot 0, etc.

Section 5.2.1.4 discusses the SIMM starting address rules. See Section 5.2.1.8 for examples of how the SIMM slots are selected based on the SIMM starting address registers.

5.4.7 RAS Timeout Counter

The 654 Controller has an internal counter that controls the maximum time that any RAS# line is active. Each time any RAS# line is asserted, the counter is reset and begins to count. When the timeout is reached, the memory controller deasserts the active RAS# between cycles. The timeout periods are as follows:

- When the PCI and CPU clock periods are the same—224 CPU bus clocks
- When the PCI clock period is twice the CPU clock period—400 CPU bus clocks

If the system clocks are to be run more slowly, it is necessary to consider the maximum RAS active time specification for the DRAMs used in the system.

5.4.8 60X CPU to System Memory CAS[7:0]# Generation

Table 5–12 shows the CAS[7:0]# lines that are asserted for a 60X CPU write to memory based on TSIZ[0:2], LE_MODE_SEL#, and CPU_ADDR[29:31]. A 60X CPU read from system memory is always eight bytes, therefore CAS[7:0]# is always 0000 0000b for a 60X CPU memory read.

In Table 5–12 the column titled *CPU_ADDR[29:31] Before Unmunge* is the address that comes from the CPU before any unmunging by the 650 Bridge. In little-endian mode, the column titled *Internal ADDR[2:0] After Unmunge* is the result in the 653 Buffer of the unmunging operation, and the CAS# assertion in little-endian mode matches the unmunged address (see Section 5.3).

The following notes apply to Table 5–12.

1. Does not occur on 60X bus because 60X bus cycles never span a double word.
2. Causes alignment exception internally in the 60X and does not occur on the 60X bus.
3. Not supported by the 650 Bridge—causes a transfer type error.

All entries that do not contain values in Table 5–12 are non-word-aligned transfers that generate transfer type errors or internal CPU exceptions. For example, TSIZ[0:2] settings of 101b, 110b, and 111b (five, six, and seven bytes) are not allowed. Transfers of five, six, or seven bytes can only come from non-double-word aligned double floating-point instructions, so non-double-word aligned double floating-point instructions are not supported. Note that floating-point load and store instructions must be word-aligned in 603 and 604 CPUs as specified in the PowerPC Architecture.

Without exception all alignments of word or half-word loads and stores as well as all move multiple and string instructions to memory are supported in big-endian mode. Programmers should note that unaligned move multiple instructions are not supported on 603 or 604 CPUs.

All transfers must be at natural alignments in little-endian mode or the 60X CPU generates internal alignment exceptions. Also, move multiple and string instructions are not supported in the 60X CPU in little-endian mode.

Note that most instruction execution is from the 60X CPU internal cache, and cache misses always cause memory to be read or written in burst mode. Therefore, alignment restrictions only apply to non–cached data. As long as the data and instructions are in cached pages, any alignments which the 60X CPU supports are allowed by the 650 Bridge.

Table 5–12. CAS[7:0]# Assertion for 60X CPU Writes to System Memory

TSIZ[0:2]	CPU_ADDR[29:31] Before Unmunge	Big-Endian CAS[7:0]#	Internal ADDR[2:0] After Unmunge	Little-Endian CAS[7:0]#
001b one byte (XOR 111b)	000	11111110	111	01111111
	001	11111101	110	10111111
	010	11111011	101	11011111
	011	11110111	100	11101111
	100	11101111	011	11110111
	101	11011111	010	11111011
	110	10111111	001	11111101
	111	01111111	000	11111110
010b two bytes (XOR 110b)	000	11111100	110	00111111
	001	11111001		(2)
	010	11110011	100	11001111
	011	11100111		(2)
	100	11001111	010	11110011
	101	10011111		(2)
	110	00111111	000	11111100
	111	(1)		(2)
011b three bytes	000	11111000		(2)
	001	11110001		(2)
	010	11100011		(2)
	011	11000111		(2)
	100	10001111		(2)
	101	00011111		(2)
	110	(1)		(2)
	111	(1)		(2)
100b four bytes (XOR 100b)	000	11110000	100	00001111
	001	11100001		(2)
	010	11000011		(2)
	011	10000111		(2)
	100	00001111	000	11110000
	101	(1)		(2)
	110	(1)		(2)
	111	(1)		(2)
101b	xxx	(3)		(2)
110b	xxx	(3)		(2)
111b	xxx	(3)		(2)
000b eight bytes	xxx not applicable	00000000	xxx not applicable	00000000

5.4.9 PCI to System Memory CAS[7:0]# Generation

The PCI specification allows the PCI_C/BE[3:0]# byte enables to change on each data phase. The memory controller asserts the CAS[7:0]# signals based on CPU_ADDR[29] (PCI_AD[2]) and PCI_C/BE[3:0]# during each data phase of the PCI burst access to system memory.

PCI reads and writes to system memory are independent of big-endian and little-endian mode. Neither munging or unmunging or byte swapping have any effect on PCI to system memory transactions. Therefore, the endian mode of the PCI device is preserved in the memory representation of the data from that device.

For a PCI burst transfer to system memory, the memory controller detects crossings of DRAM page boundaries and initiates the proper RAS and CAS memory cycles.

5.4.9.1 PCI Read from System Memory

A PCI read from system memory always reads eight bytes even though the PCI device may be reading less than eight bytes, therefore CAS[7:0]# is always 00000000b for a PCI read from system memory. PCI_AD[2] determines whether the high or low 32-bit word of the memory data bus is transferred to the PCI_AD[31:0] bus. PCI_C/BE[3:0] then determines which of the four bytes from system memory the PCI device actually reads.

5.4.9.2 PCI Write to System Memory

For a PCI write to system memory, PCI_AD[2] serves to identify whether the bytes fall within the high or low 32-bit word of the 64-bit memory data bus. The 653 Buffer actually asserts the 32-bit PCI_AD lines on both the high and low halves of the memory data bus. The CAS[7:0]# values then determine the bytes that are actually written to system memory. Table 5–13 shows the CAS[7:0]# settings for PCI writes to system memory.

Table 5–13. CAS[7:0]# Assertion for PCI Writes to System Memory

PCI_C/BE[3:0]#	PCI_AD[2] = 0 CAS[7:0]#	PCI_AD[2] = 1 CAS[7:0]#
1111 *	11111111	11111111
1110	11111110	11101111
1101	11111101	11011111
1100	11111100	11001111
1011	11111011	10111111
1010 *	11111010	10101111
1001	11111001	10011111
1000	11111000	10001111
0111	11110111	01111111
0110 *	11110110	01101111
0101 *	11110101	01011111
0100 *	11110100	01001111

Table 5–13. CAS[7:0]# Assertion for PCI Writes to System Memory (Continued)

PCI_C/BE[3:0]#	PCI_AD[2] = 0 CAS[7:0]#	PCI_AD[2] = 1 CAS[7:0]#
0011	11110011	00111111
0010 *	11110010	00101111
0001	11110001	00011111
0000	11110000	00001111

* These byte enables are not normally produced by PCI devices.

5.4.10 System Memory Control Signals—BE_PAR_EN# and LE_PAR_EN#

The parity control signals—BE_PAR_EN# and LE_PAR_EN#—are asserted during a valid memory read cycle based on the state of LE_MODE_SEL#. These signals can be used to gate the parity bits from memory to the proper CPU data parity lines. BE_PAR_EN# is asserted during big-endian mode and LE_PAR_EN# is asserted during little-endian mode. Appendix B shows an example of how to arrange these connections.

5.5 The 60X CPU Bus Cycles

The 654 Controller and 653 Buffer provide the control bridge for the 60X CPU to access system memory (DRAM), system ROM, the error address register, and PCI devices on the PCI bus. 60X CPU addresses from 0 to 256M access system memory in 1-byte to 8-byte single-beat transfers or 32-byte burst transfers. (System memory is actually mapped from 0 to 2G, but the 650 Controller can only map up to eight 32M SIMMs so the maximum memory address is 256M.)

60X CPU addresses from 2G to 3G can be translated to PCI I/O, PCI configuration, or PCI interrupt acknowledge transactions on the PCI bus in the range of 0 to 1G. 60X CPU addresses from 3G to 4G can be translated to PCI memory transactions on the PCI address bus in the range of 0 to 1G. All accesses to PCI space must be single-cycle accesses with sizes of 1, 2, 3, or 4 bytes that do not cross a 32-bit word boundary.

Table 4–1 shows the mapping the 650 Bridge performs for addresses from the 60X CPU. The table lists all the possible transactions that can occur as a result of the 60X CPU asserting an address and transfer type TT[0:3] on the CPU bus.

5.5.1 Data Transfers on the 60X CPU Bus

This section describes 60X CPU operations that are common whether the target is system memory, PCI, or ROM.

5.5.1.1 Transfer Start (TS#) and Transfer Acknowledge (TA# and TEA#)

A 60X bus device cannot assert transfer start (TS#) until the 654 Controller grants the address bus (either CPU_GNT# or L2_CACHE_GNT# is asserted). The 654 Controller does not support pipelining bus transactions—AACK# is not asserted until the last TA#. However, pipelining by an L2 cache is supported by allowing assertion of AACK# one processor clock prior to the last TA#.

Successful completion of the 60X transaction results in a transfer acknowledge (TA#) asserted to the CPU. Unsuccessful completion (parity error, illegal transfer size, or illegal alignment) results in a transfer error acknowledge (TEA#). See Section 5.8 for error conditions.

5.5.1.2 60X CPU Transfer Types—TT[0:3]

Table 5–14 shows all the possible transactions, based on TT[0:3], that the 60X CPU can assert on the host bus. As the table shows, only two of these transactions can be initiated by the 650 Bridge when it masters the CPU bus for snoop cycles.

The 650 Bridge ignores TT[4], the XFERDAT signal.

The individual TT[0:3] transfer type signals are decoded as follows:

- TT[0], special operations. The 654 Controller decodes this signal and TT[2] for two special instructions—*eciwx* and *ecowx*. The 650 asserts 64 one-bits on the 60X CPU data bus for *eciwx*.
- TT[1], read operations. The 654 Controller initiates a read operation when this signal is asserted (set to 1), a write operation when it is deasserted (set to 0).
- TT[2], invalidate caches. This signal is interpreted with TT[0] to initiate special cycles for the *eciwx* and *ecowx* 60X CPU instructions.
- TT[3], address-only operations. When TT[3] is deasserted (set to 0) the cycle is address-only and the 654 Controller responds with AACK#.

Table 5–14. TT[0:3]—Transfer Type Codes on the 60X CPU Host Bus

TT[0:3]	60X Bus Mnemonic	650 Bridge Operation	650 Can Initiate?
0000b	Clean sector	Address only, the 650 asserts AACK#	N
0001b	Write with flush	Write cycle	Y
0010b	Flush sector	Address only, the 650 asserts AACK#	N
0011b	Write with kill	Write cycle	N
0100b	sync	Address only, the 650 asserts AACK#	N
0101b	Read	Read cycle	Y
0110b	Kill sector	Address only, the 650 asserts AACK#	N
0111b	Read with intent to modify	Read cycle	N
1000b	— (Reserved)	Address only, the 650 asserts AACK#	N
1001b	Write with flush atomic	Write cycle	N
1010b	External control out (ecowx)	The 650 asserts AACK# and TA# (This instruction is not supported by the 650 Bridge)	N
1011b	— (Reserved)	Write cycle	N
1100b	TLB invalidate	Address only, the 650 asserts AACK#	N
1101b	Read atomic	Read cycle	N
1110b	External control in (eciwx)	650 asserts 64 one-bits on data bus, AACK#, and TA# (This instruction is not supported by the 650 Bridge)	N
1111b	Read with intent to modify atomic	Read cycle	N

5.5.1.3 CPU Address-Only Access

A 60X address-only access is normally terminated by the assertion of AACK#. No error is generated for address-only transactions. No memory or PCI cycles are generated.

5.5.1.4 *ECIWX* and *ECOWX*

The 650 Bridge does not support these transaction types. It does not produce an exception, and terminates the cycles by asserting AACK# and TA#. On an *eciwx* transaction, the 650 Bridge asserts 64 one-bits on the CPU data bus.

5.5.1.5 CPU Address Alignments

The 60X family of processors supports both big-endian and little-endian addressing modes. The 654 Controller also supports these two addressing modes. When the processor accesses system memory, the 654 Controller decodes TBST#, TSIZ[0:2], CPU_ADDR[29:31], and LE_MODE_SEL# to determine which CAS# lines to assert. During processor-mastered cycles to a PCI target, decode is based on TSIZ[0:2], CPU_ADDR[29:31], and LE_MODE_SEL# to determine the PCI_C/BE[3:0] byte enables that must be asserted.

The 654 Controller supports 1, 2, 3, 4, 8-byte, and burst accesses as shown in Table 5–15.

Table 5–15. 654 Controller Transfer Sizes From the 60X CPU

Transfer Size	To Memory	To PCI
1-byte	supported	supported
2-byte	supported	cannot cross a 4-byte boundary
3-byte	supported	cannot cross a 4-byte boundary (the 60X CPU does not produce 3-byte transfers in little-endian mode)
4-byte	supported	cannot cross a 4-byte boundary
8-byte	supported	no 8-byte transfers
32-byte burst	supported	no burst

5.5.2 60X CPU to System Memory (DRAM) Cycles

System memory (DRAM) reads and writes can be initiated by the 60X CPU and by PCI devices. All system memory reads and writes by the 60X CPU are snooped by the L2 cache. System memory reads and writes by PCI devices are snooped by both the L1 and L2 caches.

The 60X CPU initiates system memory reads and writes by mastering the host bus and asserting TT[0:3], the transfer type, TTSIZ[0:2], the transfer size, TBST#, the transfer burst signal, and CPU_ADDR[00:31] for the required address.

5.5.2.1 60X CPU to System Memory TSIZ[0:2] and TBST# Encoding

The 654 Controller supports single-beat transfers to system memory of 1, 2, 3, 4, and 8 bytes as well as 32-byte burst transfers. Transfers to PCI targets must be four bytes or less (no burst transfers from the 60X CPU to the PCI bus are allowed). Table 5–16 shows the valid TBST# and TSIZ[0:2] encodings for 60X CPU to system memory cycles.

Illegal combinations of TSIZ[0:2] and CPU_ADDR[29:31] are detected and an error cycle is generated as defined in Section 5.8.

Table 5–16. 60X CPU to System Memory Size Alignment

TBST#	TSIZ [0:2]	Size	Big-Endian Support	Little-Endian Support
1	001	1 byte	All accesses supported	All accesses supported
1	010	2 byte	All accesses supported	All accesses supported
1	011	3 byte	All accesses supported	Not generated by CPU
1	100	4 byte	All accesses supported	All accesses supported
1	101	5 byte	Not supported. TT_ERR# asserted	Not generated by CPU.
1	110	6 byte	Not supported. TT_ERR# asserted	Not generated by CPU.
1	111	7 byte	Not supported. TT_ERR# asserted	Not generated by CPU.
1	000	8 byte	Double-word aligned	Double-word aligned
0	XXX	32 byt	Supported	Supported

5.5.2.2 Summary of CPU Read and Write System Memory Characteristics

The following characteristics apply to 60X CPU reads and writes of system memory.

- Valid addresses range from 0 to 256M (top of real memory is programmable).
- Memory accesses above top-of-memory terminate with TA#—no error is generated.
- On a read above top-of-memory, 64 one-bits are returned.
- The CPU can read or write system memory in single-beat mode.
  - Transfer sizes of 1, 2, 3, 4, or 8 bytes are supported.
- The CPU can read or write system memory in burst mode.
  - Transfer size of 32 bytes is supported (four beats of eight bytes).
  - Each eight-byte beat must be double-word aligned.
- Successful completion of a memory cycle results in a TA# asserted to the CPU.
- Unsuccessful completion (parity error or illegal transfer size) results in a TEA# and an error bit is asserted as follows:
  - MEM_PAR_ERR# is asserted for a parity error.
  - TT_ERR# is asserted for size and alignment errors.
  - The error address is latched in the 653 Buffer.

5.5.3 60X CPU to PCI Cycles

60X addresses from 2G to 3G are translated to addresses on the PCI address bus in the range of 0 to 1G. 60X addresses from 3G to 4G are also translated to addresses on the PCI address bus in the range of 0 to 1G. When transfer start (TS#) is asserted with an address in the PCI address range, the 654 Controller initiates a PCI transaction on the PCI bus in conformance with the PCI standard described in the *PCI Local Bus Specification, Revision 2.0.*

During the address phase, the 654 Controller asserts the remapped PCI address onto PCI_AD[31:0] with PCI_FRAME# and a PCI bus command based on the 60X transfer type. During the data phase (after the target asserts PCI_DEVSEL#), the 654 Controller deasserts PCI_FRAME#, asserts PCI_IRDY#, and drives the byte enables based on the 60X address and the current endian mode.

5.5.3.1 Valid 60X CPU to PCI Transactions

A 60X transfer to PCI address space must be a single-beat transfer of one to four bytes that does not cross a word boundary. The following 60X CPU to PCI transactions are possible.

- CPU read or write PCI configuration space (type 0 only)
- CPU read or write PCI I/O space
- CPU read or write PCI memory
- CPU read of PCI interrupt acknowledge vector

5.5.3.2 Termination Responses for 60X CPU to PCI Transactions

Successful completion terminates with a TA#. Unsuccessful completion results in a TEA# for the following cases:

- PCI Master abort due to system timeout (no PCI_TRDY# response within 60us after PCI_DEVSEL#)
- PCI master abort—no target responds with a PCI_DEVSEL# to the current PCI bus transaction initiated by the 654 Controller. Except on PCI configuration cycles where a read returns 64 one-bits and a write terminates with no error.
- PCI target abort—target responds to the current PCI bus transaction, initiated by the 654 Controller, by deasserting PCI_DEVSEL# and asserting PCI_STOP#.

5.5.3.3 PCI Target Retry

If a PCI target responds with a target retry (PCI_DEVSEL# and PCI_STOP# asserted) to the current PCI bus transaction, the 654 Controller asserts address retry (ARTRY#) on the 60X CPU bus.

5.5.3.4 PCI_C/BE[3:0]#—PCI Bus Command/Byte Enable Generation

The bus commands and byte enables for the PCI bus are multiplexed on four lines of the PCI bus (PCI_C/BE[3:0]#). During the address phase of the PCI bus (ADDRHI/DATALO asserted high), the bus command for the current transaction is asserted. During the data phase of the PCI bus (ADDRHI/DATALO negated low), the byte enables for the current data transfer are asserted.

5.5.3.5 60X CPU to PCI Bus Commands

The 654 Controller generates bus commands based on a decode of CPU_ADDR[0:8] and, for PCI interrupt acknowledge or read error address, CPU_ADDR[19]. The 654 Controller maps addresses on the 60X CPU host bus from 0 to 2G as system memory (DRAM) reads and writes. Addresses from 2G to 4G are mapped as PCI cycles, system ROM reads and writes, or memory parity error address reads. The four PCI cycles are as follows:

- PCI Interrupt Acknowledge
- PCI I/O
- PCI Memory
- PCI Configuration (type 0 only)

Table 5–17 shows the PCI bus commands that the 654 Controller asserts on the PCI_C/BE[3:0]# lines during the address phase of a CPU to PCI bus transaction. The value of TT[1], a CPU transfer type bit, determines whether the PCI cycle is a read or a write.

Table 5–17. 60X CPU to PCI Bus Commands

60X Address	PCI Cycle	TT[1] (Read=1)	PCI_C/BE[3:0]#
2G to 2G + 8M	I/O Cycle	1	0010
		0	0011
2G + 8M to 2G + 16M	Configuration Cycle	1	1010
		0	1011
2G + 16M to 3G – 8M	I/O Cycle	1	0010
		0	0011
3G – 8M to 3G	Interrupt Acknowledge	1	0000
(CPU_ADDR[19] = 1)		0	not allowed
3G to 4G – 8M	Memory Cycle	1	0110
		0	0111

5.5.3.6 PCI Byte Enables

During the data phase of the PCI bus transaction, the 654 Controller individually asserts the PCI_C/BE[3:0]# lines to enable each of the four corresponding bytes of the PCI_AD bus. To determine the byte enables, the 654 Controller decodes the CPU transfer size, TSIZ[0:2], the lower two bits of the CPU address, CPU_ADDR[30:31], and the endian mode select, LE_MODE_SEL#.

One- to four-byte transfers are supported and decoded as shown in Table 5–18. A byte lane is enabled when its PCI_C/BE[3:0]# line is zero. In Table 5–18, xxxx indicates an illegal transfer attempt. The 654 Controller supports one-byte to four-byte transfers to the PCI bus that do not cross a 32-bit word boundary.

Table 5–18. PCI Byte Enables for PCI_C/BE[3:0]#

	CPU_ADDR[30:31] Big-Endian Mode (LE_MODE_SEL#=1)				CPU_ADDR[30:31] Little-Endian Mode (LE_MODE_SEL#=0)			
TSIZ[0:2]	00	01	10	11	00	01	10	11
001 (1)	1110[1]	1101	1011	0111	0111	1011	1101	1110
010 (2)	1100	1001	0011	xxxx	0011	1001[2]	1100	xxxx[2]
011 (3)	1000	0001	xxxx	xxxx	0001[2]	1000[2]	xxxx[2]	xxxx[2]
100 (4)	0000	xxxx	xxxx	xxxx	0000	xxxx[2]	xxxx[2]	xxxx[2]

[1] All entries are PCI_C/BE[3:0]#. Bytes are enabled by 0 in PCI_C/BE[3:0]#.

[2] Does not occur from 60X CPU.

5.5.3.7 Transfer Size Parameters for the PCI Bus

The 654 Controller supports 1-, 2-, 3-, and 4-byte accesses to the PCI bus as follows:

- All single-byte accesses are supported.
- Two-byte and three-byte accesses that do not cross word (32-bit) boundaries.
- Four-byte accesses that are word-aligned (32-bits) to the PCI bus.
- Eight-byte accesses are not allowed to the PCI space.

5.5.4 60X CPU to PCI Interrupt Acknowledge Cycles

When the 60X CPU executes a memory read from 3G – 8M to 3G with CPU_ADDR[19] = 1, the 654 Controller generates a PCI interrupt acknowledge transaction, and then returns the interrupt vector that is asserted on the PCI bus by the interrupt controller.

In the event of an asynchronous system error (NMI_REQ# and some L2 cache data parity errors) the 654 Controller generates an interrupt to the CPU. When the CPU reads the interrupt acknowledge address (BFFF FFF0h), the 654 Controller terminates the cycle with a TEA# (if MASK_TEA# is not asserted) and asserts 64 one-bits onto the CPU data bus. In this case, no PCI interrupt acknowledge cycle is generated. See Section 5.8.

5.5.5 60X CPU to Read Error Address Cycles

When the 60X CPU executes a memory read from 3G – 8M to 3G with CPU_ADDR[19] = 0, the 654 Controller asserts ERR_ADDR_SEL# to the 653 Buffer to return the address that was saved when the system error occurred. This access also resets TT_ERR# and MEM_PAR_ERR# signals.

The read error address cycle does not produce a PCI transaction.

5.5.6 60X CPU to System ROM Cycles

The 650 Bridge implements a boot ROM access system which minimizes pin and package count while still allowing the use of byte-wide devices on an 8-byte data bus. The 650 Bridge design is optimized for a 120ns flash memory device, but any EEPROM, non–volatile RAM, EPROM, PROM, ROM, PCMCIA, or combination of devices meeting the timing requirements can be designed in at the system level. A method is provided for writing flash ROM or other read/write devices. System ROM can contain the POST and BOOT code and vital product data for the system.

Figure 5–13 shows how system ROM addresses and data are transferred over the PCI_AD[31:0] lines. Although connected to the PCI_AD lines, the system ROM is not a PCI agent. The 654 Controller keeps the ROM from interfering with PCI bus transactions by deasserting the ROM control signals during PCI transactions. Also, the 654 arbiter will not grant the bus to any PCI agent while ROM cycles are in progress. The 654 Controller does not assert any PCI control signals (FRAME#, etc.) during system ROM transfers and therefore no PCI devices are affected by the system ROM activity. PCI bus masters are unable to access the system ROM.

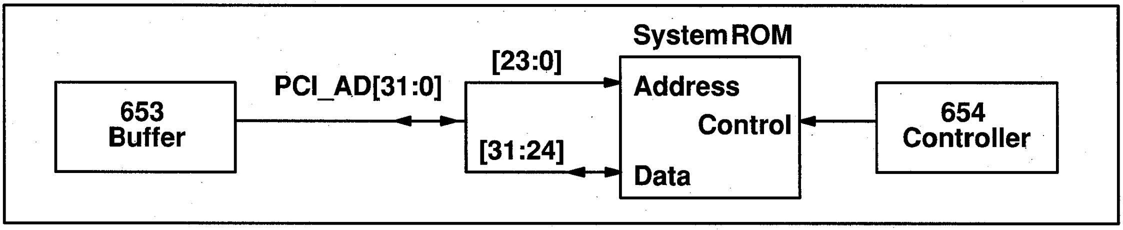


Figure 5–13. ROM Connections

5.5.6.1 ROM Addressing

During ROM reads, system ROM is linearly mapped to CPU memory space from 4G–8M to 4G (FF80 0000h to FFFF FFFFh). Since the 60X CPU begins fetching instructions at FFF0 0100h after a reset, the most convenient way to use a 512k device as system ROM with the 60X CPU is to use it from 4G–1M (FFF0 0000h) to 4G. This is implemented by connecting PCI_AD[18:0] to ROM_A[18:0] with no translation, which places the ROM 0 address at CPU memory addresses FF80 0000h, FF88 0000h, FF90 0000h,..., FFF0 0000, FFF8 0000h. Connected like this, the system ROM is aligned with 4G – 8M, but with alias addresses every 512K up to 4G.

Writing to flash ROM is a specialized cycle. A CPU memory write to any even address in the range 4G – 8M to 4G initiates a ROM write cycle.

5.5.6.2 ROM Access Data Sizes and Alignments

ROM read cycles ignore transfer size (TSIZ[0:2]) and alignment (CPU_ADDR[29:31]). The 653 Buffer begins by forcing the low-order three bits of the address to 000b, then reads the ROM eight times, incrementing the ROM address by one for each read. The eight bytes read from ROM are accumulated into a single 64-bit double word which is then driven onto the CPU data bus.

Only 4-byte memory write cycles (store word) are supported to the ROM. One of these bytes is used as data, and the other three are used as address information. Burst writes are not supported.

5.5.6.3 Single-Beat ROM Reads

If TBST# is not asserted during the ROM read cycle, the 650 Bridge executes a single-beat ROM read operation. This operation delivers eight bytes of ROM data to the CPU. The 650 begins by reading ROM data starting at the address to which the CPU memory access has been mapped. The 650 places that byte into a shift register that reads out onto the 60X data bus. The 650 then increments the value of the ROM address lines, and shifts that byte into the shift register, which pushes the first byte over one byte. The 650 continues this pattern until 8 bytes have been read out of the ROM and driven onto the 60X data bus. The 654 Controller then asserts AACK# and TA# for one CPU_CLK cycle and the 60X CPU completes the transfer.

5.5.6.4 Burst ROM Reads

The PowerPC 601 microprocessor begins instruction fetching in burst mode after a reset (the 603 CPU and 604 CPU do not come up with burst mode enabled). To support burst mode, the 650 Bridge operates in a pseudo burst mode, which supplies the same eight bytes of data (from the ROM) to the CPU on each beat of a 4-beat burst.

A burst ROM read begins with the 654 Bridge executing a single-beat ROM read operation, which assembles eight bytes of ROM data into a double word on the CPU data bus. For a single-beat read, the 650 Bridge then asserts TA# and AACK# for one CPU_CLK cycle, and the 60X CPU completes the transfer. For a burst ROM read however, the 654 Bridge asserts TA# for four CPU_CLK cycles, with AACK# asserted on the fourth cycle. The same data remains asserted on the CPU data bus for all four of the data cycles.

5.5.6.5 Programming the ROM Boot For 601 Burst Reads

To construct the bootstrap portion of the code that is required for use with the 601 CPU pseudo burst mode ROM reads described in Section 5.5.6.4, the first part of the system ROM can be coded as follows:

```
Instruction 1
Branch to instruction 2
No–op
No–op
No–op
No–op
No–op
No–op
Instruction 2
Branch to instruction 3
6 No–ops
Instruction 3
Branch to instruction 4
etc.
```

The six no-op instructions serve as filler for the unexecuted phases of the burst reads of the system ROM. The no-op codes are not transferred during the burst read, only the first two instructions (64 bits) are read and then passed four times to the 601 CPU during a startup burst read of system ROM.

When enough instructions have been executed, the bootstrap code can turn off the 601 cache, and the remaining ROM data can be read contiguously as single-beat reads.

5.5.6.6 60X CPU to Flash ROM Write Cycles

The 650 Bridge decodes a CPU store word to any even address from 4G – 8M to 4G (ex. FFFF FFF0) as a flash ROM write cycle. The three low-order bytes of the CPU data word are driven onto the ROM address lines, and the upper byte is driven onto the ROM data lines. Only single beat, four-byte write transfers (store word) are supported—bursts are not supported. For example, a store word instruction with data=00ABCDEF would write EF to ROM location 00ABCD.

5.5.6.7 Effect of Endian Mode on ROM Writes

Writes to flash ROM can be performed while the system is in either big-endian or little-endian mode. During ROM writes, the data byte swapper and the address unmunger are controlled according to endian mode, but the address unmunging (in little-endian mode) has no effect on the placement of the data because the CPU_ADDR[29:31] bits are ignored. Therefore software must reverse the byte significance of the data and addresses encoded into the store instructions for little-endian mode. In little-endian mode, the data must be aligned at CPU_DATA[24:31] and the address (byte swapped) at CPU_DATA[00:23] before the store word instruction is executed.

5.5.6.8 Flash ROM Protection

The 650 Bridge decodes a CPU write to any odd address from 4G – 8M to 4G as a flash ROM write lockout cycle. For example, a write to FFFF FFF1h locks out subsequent flash ROM writes. Writing any data to this port address locks out all flash ROM writes until the power is turned off and back on. In addition, flash ROM devices can have the means to permanently lock out sectors by writing control sequences. Flash ROM specifications contain details.

5.5.7 60X CPU to System ROM Detailed Operation

As shown in Figure 5–14 and Figure 5–17, the address and address attribute signals in the example system (see Appendix B) flow from the 60X CPU into the 654 Controller, which decodes these signals and issues the appropriate control signals. The data flow during writes is from the CPU, through the 653 Buffer, and onto the PCI_AD lines to the ROM. During reads, the data flow is from the ROM onto the PCI_AD lines, through the 653 Buffer into the CPU. The address flow during ROM reads is from the CPU, through the 653 Buffer, and onto the PCI_AD lines to the ROM. During writes, the address from the CPU does not flow to the ROM—the ROM address is encoded in the data that the CPU writes to ROM space, as explained below.

The following discussion assumes that the system is operating in big-endian mode, which is typically the case during ROM transfers. Differences in the operation of the system in little-endian mode are noted but not usually detailed in the example operations. A full understanding of the 653 Buffer is also helpful—see Appendix C.

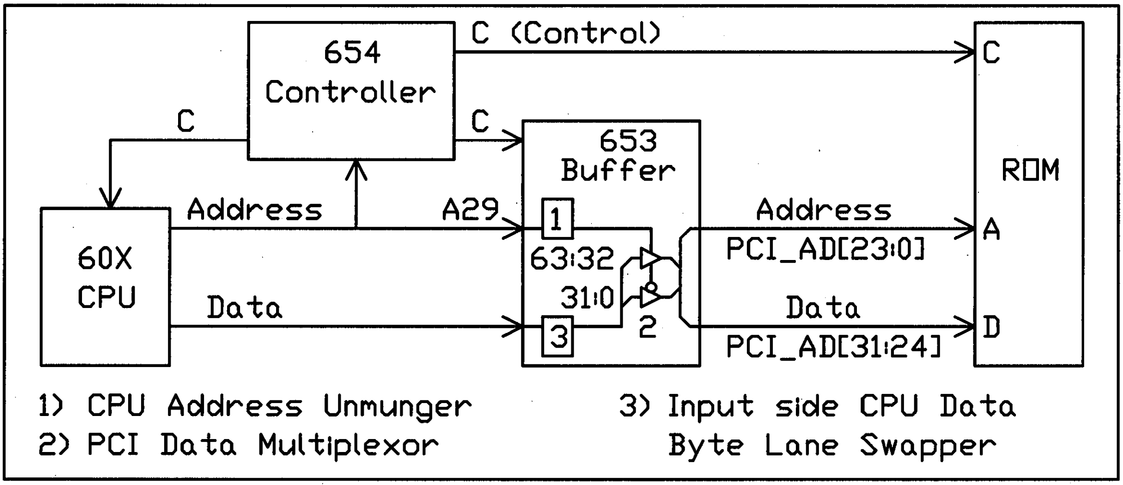


Figure 5–14. CPU to ROM Write Address and Data Flows

5.5.7.1 ROM Write Detailed Operation

A flash memory (ROM) write operation occurs as the result of a 60X CPU store word instruction to any CPU bus location in the address range FF80 0000h through FFFF FFFEh while A31 = 0. For example, FF80 0010h writes to ROM, but FF80 0011h does not. As shown in Figure 5–14, during a ROM write in the example system the 653 Buffer does not forward the information from the CPU bus address lines to the ROM address lines. Instead, the information from the CPU data bus is split into two fields. The upper byte is written into the ROM as data, and the low-order three bytes are used to address the ROM.

As shown in Figure 5–14, the data from the CPU flows into the 653 Buffer, into the input side CPU data byte lane swapper, which reverses the byte order in little-endian mode, but does not affect the byte order in big-endian mode. This CPU data is sent to the PCI output multiplexers. Since this is not a PCI cycle, the multiplexers are set to drive the PCI bus only with the data taken from the CPU bus (there is no address phase). Four bytes of the eight-byte CPU data bus are selected to be driven onto the PCI_AD bus, and the ordering of the selected bytes depends on CPU address bit A[29] (big-endian) and the endian mode of the system (see Table 5–19 and Table 5–20).

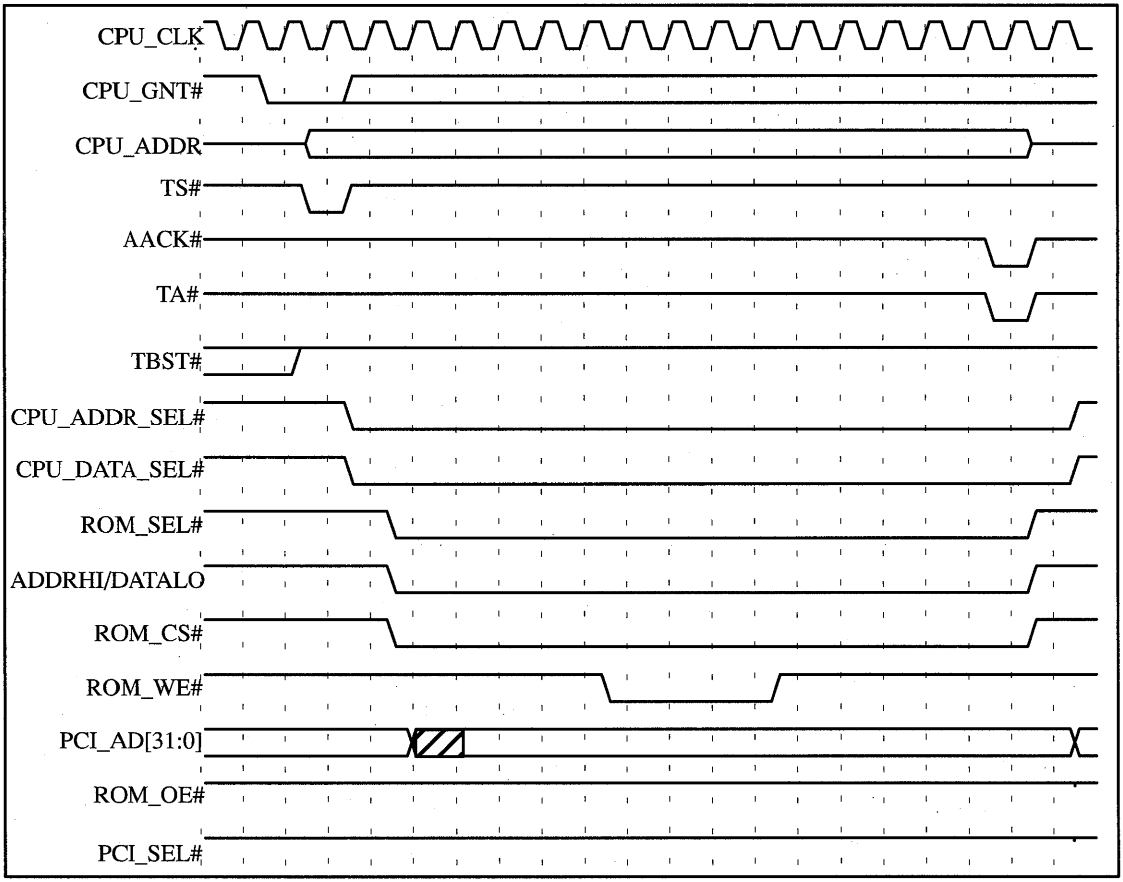


Figure 5–15. CPU to ROM Write Timing Diagram

Figure 5–15 shows the signals involved in a ROM write operation. The CPU initiates the transfer by asserting TS#. The 654 Controller decodes the required operation and asserts CPU_ADDR_SEL# and CPU_DATA_SEL# on the CPU_CLK that TS# was sampled valid. On the next CPU_CLK, the 654 Controller asserts ROM_SEL# and ADDRHI/DATALO to the 653 Buffer and asserts ROM_CS#. These signals open the appropriate data and address pathways in the 653 Buffer and select the ROM. The 654 Controller asserts ROM_WE# from the sixth to the tenth CPU_CLK cycle after TS#, and then asserts AACK# and TA# to the CPU on cycle 15 to end the transfer.

Table 5–19. ROM Write Data Flow in Big-Endian Mode

CPU store word to x—x x0x0b in big-endian Mode. Not munged CPU A[29] = 0. Not unmunged pci_addr_out[2] = 0. 653 after swapper data bus [31:0] selected for output to PCI_AD[31:0].				CPU store word to x—x x1x0b in big-endian Mode. Not munged CPU A[29] = 1. Not unmunged pci_addr_out[2] = 1. 653 after swapper data bus [63:32] selected for output to PCI_AD[31:0].			
CPU DATA	After Swapper	PCI_AD	ROM Signal	CPU DATA	After Swapper	PCI_AD	ROM Signal
0:7	7:0	7:0	A[7:0]	0:7	7:0	not used	not used
8:15	15:8	15:8	A[15:8]	8:15	15:8	not used	not used
16:23	23:16	23:16	A[23:16]	16:23	23:16	not used	not used
24:31	31:24	31:24	D[7:0]	24:31	31:24	not used	not used
32:39	39:32	not used	not used	32:39	39:32	7:0	A[7:0]
40:47	47:40	not used	not used	40:47	47:40	15:8	A[15:8]
48:55	55:48	not used	not used	48:55	55:48	23:16	A[23:16]
56:63	63:56	not used	not used	56:63	63:56	31:24	D[7:0]

Table 5–20. ROM Write Data Flow in Little-Endian Mode

CPU store word to x—x x0x0b in little-endian Mode. Munged CPU A[29] = 1. Unmunged pci_addr_out[2] = 0. 653 after swapper data bus [31:0] selected for output to PCI_AD[31:0].				CPU store word to x—x x1x0b in little-endian Mode. Munged CPU A[29] = 0. Unmunged pci_addr_out[2] = 1. 653 after swapper data bus [63:32] selected for output to PCI_AD[31:0].			
CPU DATA	After Swapper	PCI_AD	ROM Signal	CPU DATA	After Swapper	PCI_AD	ROM Signal
0:7	63:56	not used	not used	0:7	63:56	31:24	D[7:0]
8:15	55:48	not used	not used	8:15	55:48	23:16	A[23:16]
16:23	47:40	not used	not used	16:23	47:40	15:8	A[15:8]
24:31	39:32	not used	not used	24:31	39:32	7:0	A[7:0]
32:39	31:24	31:24	D[7:0]	32:39	31:24	not used	not used
40:47	23:16	23:16	A[23:16]	40:47	23:16	not used	not used
48:55	15:8	15:8	A[15:8]	48:55	15:8	not used	not used
56:63	7:0	7:0	A[7:0]	56:63	7:0	not used	not used

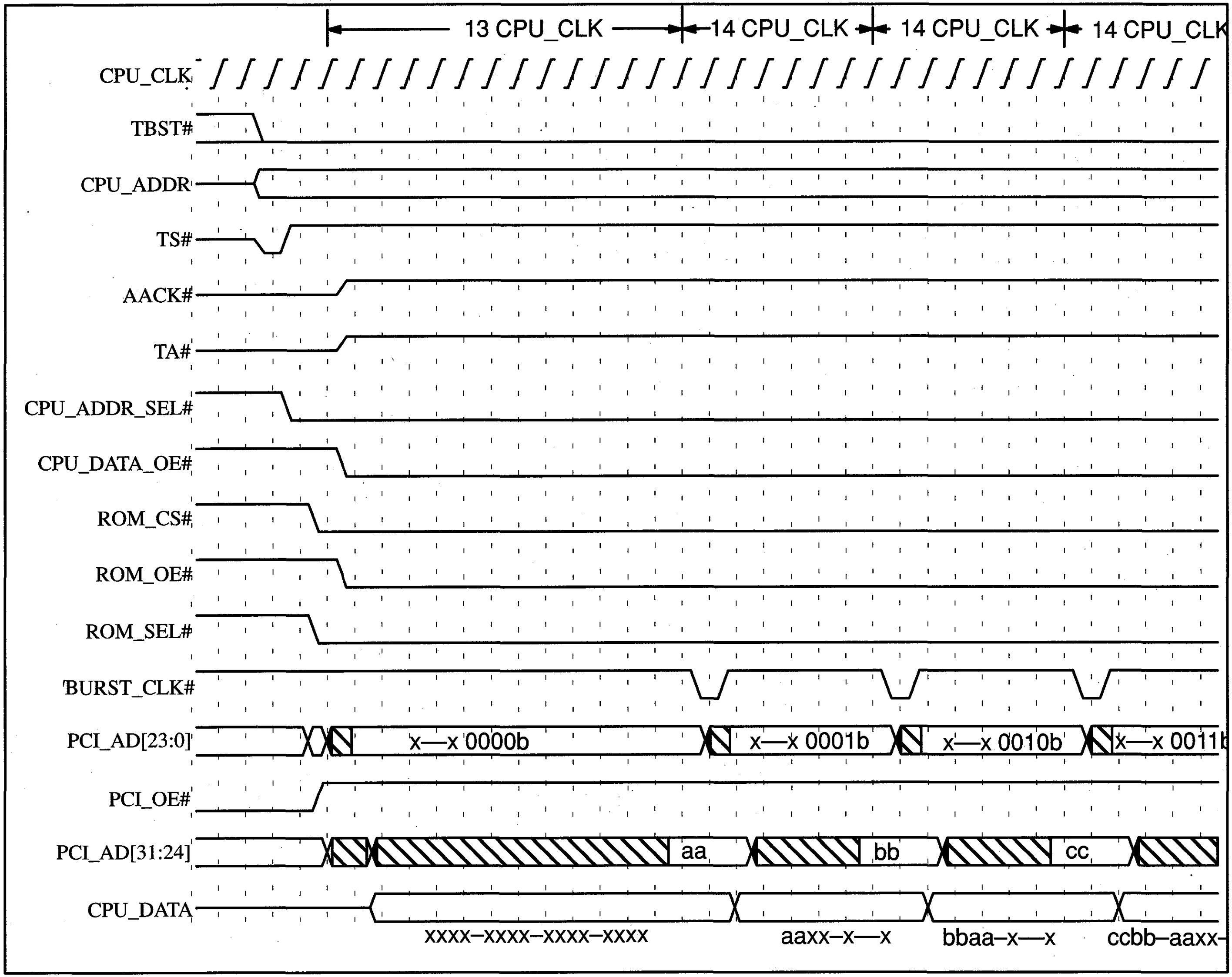


Figure 5–16. Timing Diagram, CPU to ROM Read

5.5.7.2 ROM Read Detailed Operation

Figure 5–16 and Figure 5–18 show a complete ROM read timing diagram. A ROM read operation occurs as the result of a CPU memory read to CPU bus address range 4G – 8M to 4G. Once the 650 Bridge has detected the correct combination of CPU address and address attribute signals, it starts the ROM read engine, doing eight one-byte reads from the ROM, stacking up the eight bytes in a shift register, and then transfering the eight–byte double word to the CPU, all of which takes 117 CPU_CLK cycles for a single-beat CPU transfer. If the operation is a four-beat burst read transfer, such as a 601 CPU does at power up, the same eight–byte double word is transfered four times to the CPU, which takes three more CPU_CLK cycles for a total of 120 clock cycles.

Figure 5–16 shows the signals involved in a ROM read operation starting at ROM address x—x 0000b, which is initiated as the CPU begins a memory read from a CPU bus address mapped to ROM space (4G – 8M to 4G) by asserting TS#. The 654 Controller asserts ROM_CS# and ROM_OE# to the ROM, and asserts CPU_ADDR_SEL#, CPU_DATA_OE#, and ROM_SEL# to the 653.

As shown in Figure 5–17, during a ROM read in the example system (see Appendix B), the address information flows into the 653 Buffer, which flows the address through the CPU address unmunger, the ROM read burst counter, and the CPU burst counter. The address then flows out of the 653 Buffer onto PCI_AD[23:0], and then to the ROM. The unmunger operates normally, but does not actually affect the address presented to the ROM, due to the operation of the ROM read burst counter, as discussed below. The CPU burst counter is also not used, since only one eight-byte double word is actually accessed, even during a four-beat CPU burst.

As shown in Figure 5–17, during a ROM read in the example system, the data flows from the ROM onto the PCI_AD[31:24] lines and into the 653 Buffer, where it is stacked up in the ROM data shift register, sent through the output side CPU data byte lane swapper (where it is byte–reversed in little-endian mode), and then sent out of the 653 Buffer to the CPU on the CPU bus data lines.

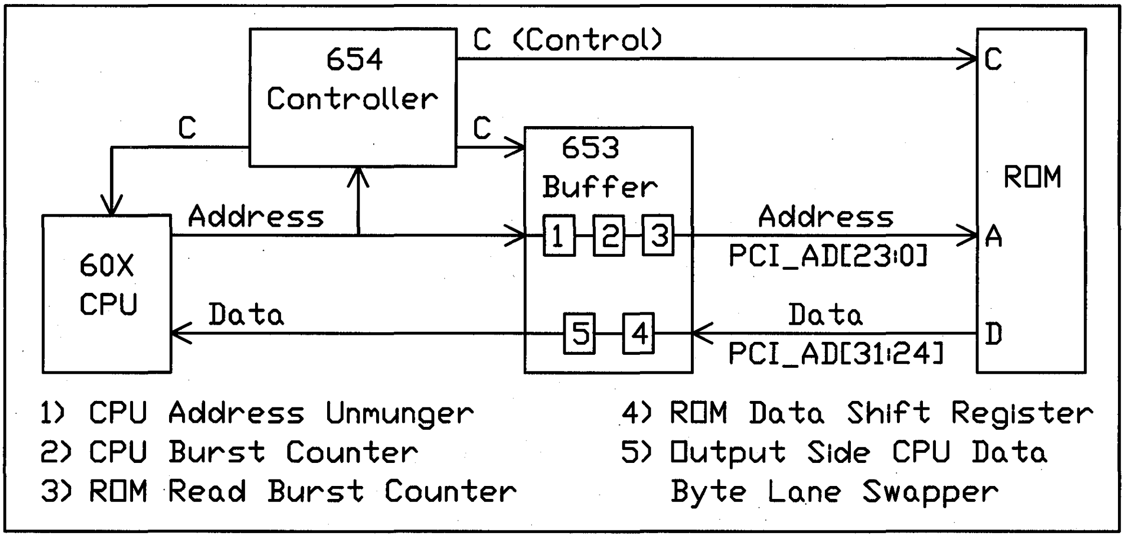


Figure 5–17. CPU to ROM Read Address and Data Flows

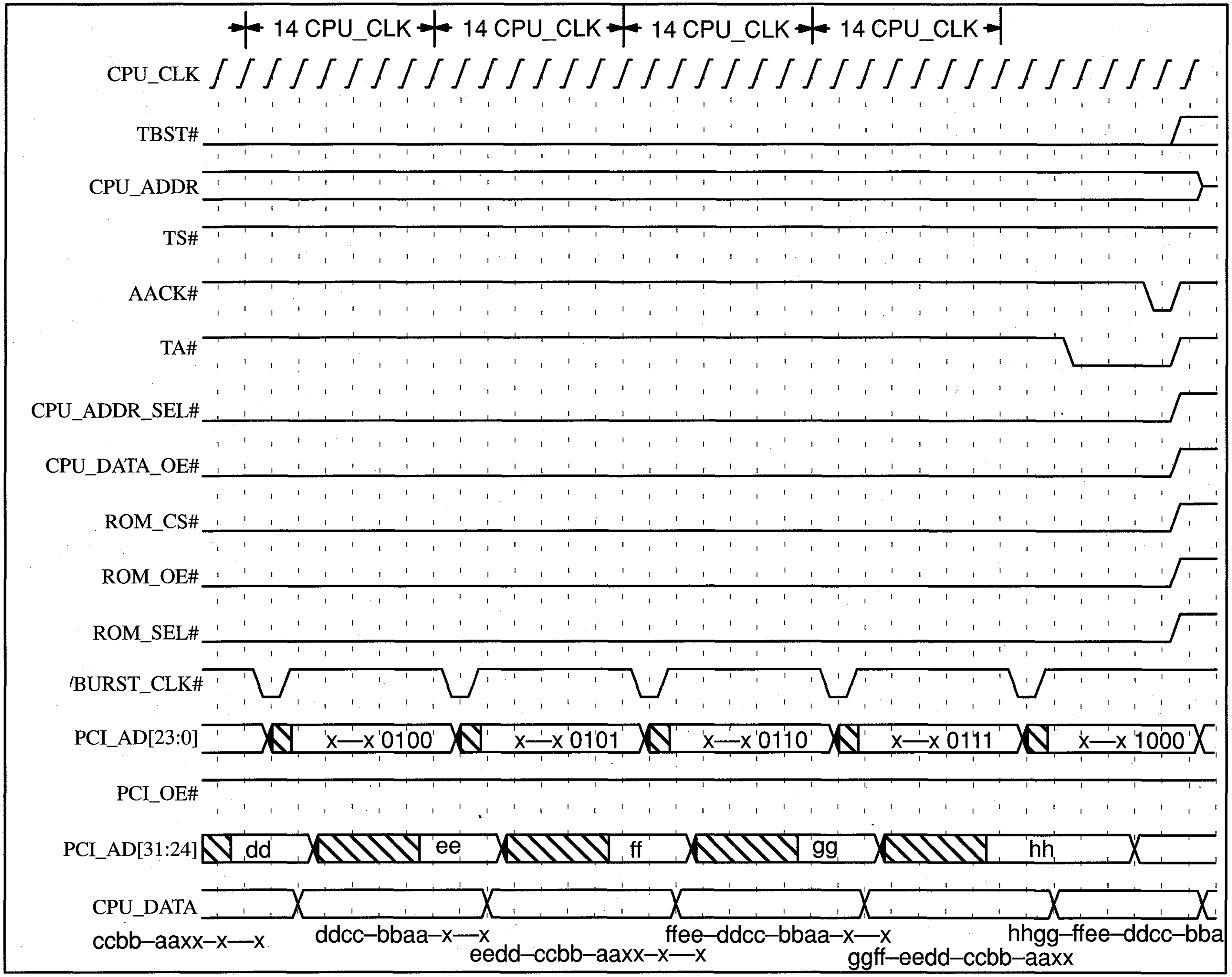


Figure 5–18. Timing Diagram, CPU to ROM Read, Continued

In the 653, the CPU address lines are chosen as the source of the address which flows into the ROM burst counter. Here the three least-significant address bits [2:0] are forced to 000b (making the operation of the unmunger ineffective), and are driven onto PCI_AD[23:0] (which are connected to the address lines of the ROM in the example system). This address and the ROM control lines access a single byte of ROM data, which flows onto PCI_AD[31:24] and into the 653 Buffer, at the input of the ROM read shift register. At this point the shift register and the CPU data lines contain no useful information, as shown in Figure 5–16 on CPU_DATA by xxxx–xxxx–xxxx–xxxx(h).

After waiting for the ROM data to stablize, the 654 Controller asserts ROM/BURST_CLK# for one CPU_CLK cycle. This causes the 653 Buffer to latch the ROM data byte from PCI_AD[31:24] into the ROM data shift register, shuffle all the other data bytes in the shift register down one byte position, and place the new byte on the internal data bus in byte lane 0—the most-significant byte lane. This data flows through the output side CPU data byte lane swapper (which will swap the byte lanes around if the system is in little-endian mode, but no swapping is done in big-endian mode). So, in big-endian mode, the data byte from ROM location x—x 0000b now appears on CPU_DATA[0:7], the most significant byte. The other CPU data bytes contain no useful information. This is shown in Figure 5–16 on CPU_DATA as aaxx–xxxx–xxxx–xxxx, where aa is the byte of data from ROM location x—x 0000b.

This initial assertion of ROM/BURST_CLK# also latches the CPU address into the ROM read burst counter and increments bits [2:0] to 001b. This address flows out to the ROM. After waiting for the ROM data to settle out, the 654 Controller again asserts ROM/BURST_CLK#, latching the data from ROM location x—x 0001b (shown here as bb) onto CPU byte lane 0, shuffling all the other bytes down, and incrementing the ROM address. CPU_DATA[0:63] now contains bbaa–xxxx–xxxx–xxxx. This process continues for a total of eight ROM/BURST_CLK# pulses, after which CPU_DATA[0:63] contains hhgg–ffee–ddcc–bbaa, the 8 bytes of data from the ROM. (Timing diagram Figure 5–16 is continued as Figure 5–18).

For a single-beat CPU read transfer, the 654 Controller then completes the transfer by asserting AACK# and TA# to the 60X CPU for one CPU_CLK cycle, and deasserting the 653 Buffer and ROM control signals which returns the system to the bus idle state.

For a burst-mode read transfer, after the eighth ROM/BURST_CLK#, the 654 Controller asserts TA# for four CPU_CLK cycles (rather than for just one CPU_CLK cycle) while holding the same data on the CPU data bus, and asserts AACK# for one cycle on the fourth TA#. This transfers the same eight bytes to the 60X CPU on each beat of the burst transfer. No additional data is read from the ROM. The CPU transfer is completed on the fourth cycle as the 654 Controller asserts TA# for the fourth CPU_CLK cycle and asserts AACK#. The 654 Controller then returns the system to the bus idle state by deasserting the 653 Buffer and ROM control signals.

5.6 The PCI to 650 Bridge Transactions

A read operation always returns a 64-bit double-word since system memory is a double-word bus (eight-byte data bus). PCI_AD[2] is used to select between the high-order and low-order words within this double-word. Memory parity is checked based on a full double-word. (A parity error can occur if all of memory is not initialized prior to access.)

A PCI device can assert PCI_FRAME# to initiate an address phase after the 654 Controller grants the address bus (either IO_BRDG_GNT# or one of the five PCI_GNT lines) to the PCI device. Successful completion of the PCI transaction results in a target ready (PCI_TRDY#) asserted to the PCI device. Unsuccessful completion (memory out-of-range, parity error) results in a target abort (PCI_DEVSEL# deasserted and PCI_STOP# asserted), and an error bit is asserted—MEM_PAR_ERR# for a parity error.

If a parity error occurs during a PCI read of system memory, the 654 Controller asserts TRDY#, then drives incorrect (inverted) parity onto the PCI_PAR line on the PCI clock after TRDY#, and then target aborts (PCI_DEVSEL# deasserted and PCI_STOP# asserted). The 654 Controller asserts the MEM_PAR_ERR# signal and generates an interrupt to the 60X CPU. All subsequent PCI transactions to system memory from any agent are terminated with a target abort until after the 60X reads the error address register.

5.6.1 PCI to System Memory Cycles

- A PCI address from 2G to 2G + 256M translates to system memory from 0 to 256M.
- PCI memory read cycles from 0G to 2G translate to 3G to 4G. These cycles cause snoop cycles but no hits because the 3G to 4G address range is reserved as non-cacheable.
- Single or burst transfers are supported (PCI 2.0 specification compliant).
- From one to four bytes per beat are allowed (controlled by PCI byte enables).

5.6.1.1 I/O Bridge to System Memory

- Supported by the 650 Bridge chip set if I/O bridge support exists.
- ISA_MASTER# pin allows special translation for ISA master addresses from 0 to 16M on the PCI bus.

5.6.1.2 ISA Master Memory Addressing

The 650 Bridge forwards PCI memory cycles which are the the result of an ISA bus master operation to system memory. The ISA bridge asserts ISA_MASTER# and IO_BRDG_HOLD# to the 650 Bridge to indicate ISA bus master operations.

Note: If the DMA produces an address in the 0 to 2G range without asserting ISA_MASTER#, a PCI cycle runs, but the 650 Bridge does not forward it to system memory because the address range is not 2G to 4G.

5.6.1.3 ISA Master Signal Timing

Figure 5–19 shows the timing relationships for ISA master operations.

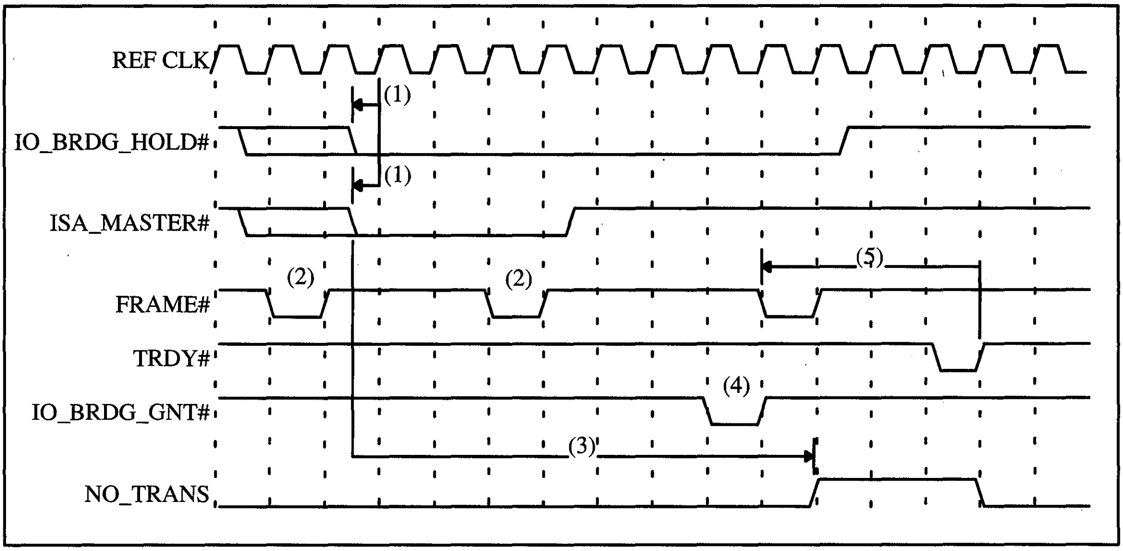


Figure 5–19. ISA Master Signal Timing

Notes for Figure 5–19:

1. IO_BRDG_HOLD# and ISA_MASTER# must be sampled asserted on the same clock for the 650 Bridge to recognize an ISA master transaction pending condition.
2. These transactions are mastered by the 650 Bridge or by a PCI bus master other than the I/O bus bridge.
3. When the ISA master transaction pending condition is recognized, the 650 responds to the next PCI transaction mastered by the I/O bus bridge as a PCI transaction on behalf of an ISA bus master. The 650 asserts NO_TRANS to disable the address translation that normally inverts the most significant address bit when a PCI bus master accesses system memory.
4. The arbiter grants the system to the I/O bus bridge.
5. This PCI transaction is mastered by the I/O bus bridge for the ISA bus master that has ISA bus mastership.

5.6.1.4 PCI to System Memory (DRAM) PCI_C/BE[3:0]# Bus Commands

Table 5–21 shows the PCI bus command decoding. When a PCI master has the PCI bus grant and asserts PCI_FRAME#, the 654 Controller decodes PCI_C/BE[3:0]# to determine if the PCI device is trying to access system memory. The 654 Controller maps PCI memory cycles with addresses in the range of 2G to 4G as system memory (DRAM) reads and writes. The 650 Bridge only responds to PCI memory read and write cycles. All other cycles initiated by PCI devices on the PCI bus are ignored by the 650 Bridge.

If a memory cycle is decoded, the 654 Controller must determine if the translated memory address is in the range, from 0 to 2G, of system memory (DRAM). PCI devices address system memory with addresses from 2G to 4G. The 653 Buffer inverts PCI_AD[31] to remap PCI addresses in the 2G to 4G range to 0 to 2G.

The 654 Controller decodes the PCI bus address from the 60X CPU bus after the address is translated by the 653 Buffer (PCI_AD[31] is inverted). The 654 Controller initiates system memory (DRAM) cycles with snooping for addresses on the 60X CPU bus from 0 to 2G. For a 60X CPU bus address from 2G to 4G, the 654 Controller aborts the memory cycle. In this case, the snoop cycle is always a miss because addresses from 2G to 4G are reserved as not cacheable. (They must be marked non-cacheable in the 60X CPU).

Table 5–21. PCI Bus Commands from PCI Masters

PCI_C/BE[3:0]#	PCI Transaction	Decoded as:
0000	Interrupt Acknowledge	none
0001	Special Cycle	none
0010	I/O Read	none
0011	I/O Write	none
0100	Reserved	none
0101	Reserved	none
0110	Memory Read	memory read
0111	Memory Write	memory write
1000	Reserved	none
1001	Reserved	none
1010	Configuration Read	none
1011	Configuration Write	none
1100	Memory Read Multiple	memory read
1101	Dual Address Cycle	none
1110	Memory Read Line	memory read
1111	Memory Write and Invalidate	memory write

5.6.1.5 Snoop Cycle Control Signals on the 60X CPU Host Bus

The 654 Controller maintains cache coherency with the L1 and L2 caches by running snoop cycles on the 60X CPU bus for every PCI read or write to system memory, including burst transactions. Section 5.7 describes the processing of snoop cycles in detail. To execute a snoop cycle, the 654 Controller asserts the following 60X CPU bus control signals:

- TBST# is negated
- TSIZ[0:2] (transfer size) is set to binary 100 (four bytes or one word).
- TT[0:3] (transfer type) is set to 0101b for snooping a read to system memory and to 0001b for snooping a write to system memory.

The 60X CPU and a write-back L2 cache respond to a cache snoop hit by asserting ARTRY#. See Section 5.7 for a complete description of the processing of snoop cycles. The 650 Bridge asserts a target retry on the PCI bus when a cache device asserts ARTRY# for a cache hit. After the cache completes its writeback, the 650 Bridge grants the bus to the original PCI device to retry the target retried transfer (if it is requesting the bus).

5.7 L2 Secondary Cache Protocol

The L2 cache provides two different services—caching for 60X CPU accesses to system memory and snooping for PCI to system memory accesses. Table 5–22 shows the actions the L1 and L2 caches and the 650 Bridge can take when the 60X CPU reads or writes system memory and when a PCI device reads or writes system memory.

Table 5–22. Cache and 650 Bridge Action Table

Transfer	L1 Action	L2 Action	650 Bridge Action
CPU to Memory Transfer	Any action taken by the L1 does not have an effect on the 60X bus, because the L1 activity takes place completely within the 60X CPU.	No bus action. (The L2 can invalidate, snarf, update, etc.)	Paces transfer. Accesses DRAM.
		Claims transfer with L2_CLAIM#. Paces the transfer. Supplies (or receives) the data.	Does not pace the transfer. Does not access DRAM.
		Backs off transfer with ARTRY#. Asserts bus request.	Does not pace the transfer. Does not access DRAM. Arbitrates.
PCI to Memory Transfer	No bus action	No bus action.	Paces transfer. Accesses DRAM.
	Backs off transfer with ARTRY#. Asserts bus request.	No bus action.	Does not pace the transfer. Does not access DRAM. Arbitrates.
	No bus action.	Backs off transfer with ARTRY#. Asserts bus request.	Does not pace the transfer. Does not access DRAM. Arbitrates.
	Backs off transfer with ARTRY#. Asserts bus request.	Backs off transfer with ARTRY#. Asserts bus request.	Does not pace the transfer. Does not access DRAM. Arbitrates.

5.7.1 L2 Caching for 60X CPU Accesses to System Memory

60X CPU reads or writes of system memory are serviced by the 650 Bridge unless L2_CLAIM# is asserted (with L2_PRESENT# asserted) on the second CPU clock after the assertion of TS# (transaction start) by the 60X CPU. When the 654 Controller senses L2_CLAIM# asserted, it drops the 650 Bridge completely out of the servicing of the transaction, and the L2 cache takes over driving the data, AACK#, and TA# lines to complete the cycle to the 60X CPU.

The L2_CLAIM# signal must be held asserted by the L2 cache throughout the remainder of the memory transaction, until the L2 asserts AACK# and TA# at the end of the transaction. The L2 Cache can assert AACK# one clock prior to the final TA# or during the same clock of the final TA#.

The L2_CLAIM# signal can be asserted before the second CPU clock after TS# is asserted by the 60X CPU, but the 650 Bridge only samples the signal on the second CPU clock.

5.7.2 Cache Snooping for PCI to System Memory Accesses

On PCI to memory cycles, the 654 Controller masters the 60X bus to drive the required snoop cycles to the 60X and L2. During PCI to system memory cycles, the 60X CPU and the L2 cache assert ARTRY# (rather than L2_CLAIM#) for a cache hit. The 654 Controller drives AACK# active one CPU bus clock after it asserts TS# and then samples ARTRY# the next CPU bus clock.

To execute a snoop cycle, the 654 Controller drives the following 60X CPU bus control signals:

- TSIZ[0:2] (transfer size) is set to binary 100 (four bytes or one word).
- TT[0:3] (transfer type) is set to 0101b for snooping a read to system memory and to 0001b for snooping a write to system memory.
- TBST# is deasserted

5.7.2.1 Restoring ARTRY#

If the setup bit ARSTR is set to one when the 654 Controller samples ARTRY# active on the second clock after TS#, the 654 Controller drives ARTRY# inactive the second clock after AACK# is inactive and then tri-states its buffer. This is required because neither the L2 nor the 60X can restore ARTRY# since they both can be driving it and one is a 3.3V or 3.6V part and the other can be a 5.0V part. If ARSTR is not set, the device that asserts ARTRY# must deassert ARTRY# one clock after AACK# is asserted.

5.7.2.2 Arbitration on Cache Hits

The 654 Controller samples L2_CACHE_REQ# and CPU_REQ# the clock after ARTRY# is asserted by either the 60X or the L2 or both. If only one request is active, the 654 Controller grants the bus to that requester (or refresh) before granting the bus to another master.

If both the 60X and L2 bus request lines are active, the 654 Controller grants the bus to the 60X first. If after the end of the cycle, the L2_CACHE_REQ# is still active, then bus mastership is granted to the L2. (A well-behaved write-back L2 updates during the 60X write and drops its bus request.)

5.7.3 Error Checking for the L2 Cache

L2 cache designs store 64 data bits plus the 8 parity bits from the 60X CPU bus. When the L2 cache supplies data, it asserts both the 64 data bits and the 8 parity bits on the CPU bus. The 650 Bridge uses the DPE# signal from the 60X CPU to determine if there has been a parity error in the data supplied by the L2 cache.

The 654 Controller samples DPE# from the 60X CPU two clocks after each TA# is asserted by the L2 (L2_CLAIM# was asserted) to determine if the L2 data has good parity. If the 654 Controller samples DPE# asserted, TEA# is asserted if the cycle is still in progress on the bus. (If DPE# occurs after the cycle completes, the 654 asserts INT_CPU#.)

5.7.4 Additional L2 Cache Information

There is an exception to snooping the 60X CPU bus on every PCI to memory cycle. It occurs during a burst transaction when the PCI access is the most-significant word within a double-word boundary of system memory. A snoop is not necessary in this case because the cache sector has already been snooped by the low-order word within the double-word boundary.

The 654 Controller does not assert TA# during snoop cycles because these cycles are being run only to snoop the 60X CPU and L2 cache for addresses that a PCI master is running to system memory.

A write-back L2 cache can execute 32-byte burst read or write cycles on the 60X CPU bus in the same manner as the 60X. The 654 Controller grants permission to the L2 cache to master the bus (L2_CACHE_GNT# is asserted) before the L2 cache can initiate a 60X cycle with TS#.

5.7.5 Example of a PCI to Memory Read Transaction With Cache Hit

The PCI to memory read transaction shown in Figure 5–20 is identical to the one shown in Figure 4–15 up to PCI_CLK 2. To signal a cache hit, either the L1 or L2 cache asserts ARTRY# so that the 654 Controller samples it asserted on PCI_CLK 3. At this point, the 654 Controller generates a PCI target retry, shuts down the memory controller, and begins an arbiter switch.

The 654 Controller generates a PCI target retry by asserting STOP# on PCI_CLK 3 (DEVSEL# remains asserted from PCI_CLK 2). On PCI_CLK 4, the 654 Controller negates DEVSEL# and STOP# (TRDY# was not yet asserted). A well behaved PCI bus master will also remove FRAME# and IRDY# by PCI_CLK 4. The 654 Controller also negates PCI_OE# on PCI_CLK 4 to disable the PCI_AD bus drivers in the 653 Buffer.

To shut down the memory controller, the 654 negates MEM_DATA_SEL# on PCI_CLK 4.

During PCI to memory write transactions with page hit and cache hit, WE# and MEM_DATA_OE# are negated on PCI_CLK 3, before the CAS# access sequence begins. During PCI to memory write transactions with page miss and cache hit, RAS# and RASHI/CASLO are driven high on the CPU_CLK following PCI_CLK 2—a cache hit on PCI_CLK 3 causes them to be left high. This also forces a page miss on the next memory access.

To begin the arbiter switch, the 654 Controller removes the grant from the PCI bus master by PCI_CLK 4. Meanwhile, the cache that signaled a cache hit (L1 or L2 or both) asserts its bus request signal. On the CPU_CLK following PCI_CLK 5, the arbiter grants the bus to the requesting cache.

Since the current bus master is losing the bus grant, PCI_SEL# and CPU_ADDR_OE# are driven high and AACK# is tri-stated on PCI_CLK 5.

The following notes refer to Figure 5–20:

1. These signals are sourced by the responding cache, L1 or L2.
2. During PCI to memory transactions during which there is a cache hit by the L1 or L2 cache, the responding cache must assert ARTRY# on the CPU_CLK after it samples TS# active, or the cache hit condition will not be recognized.

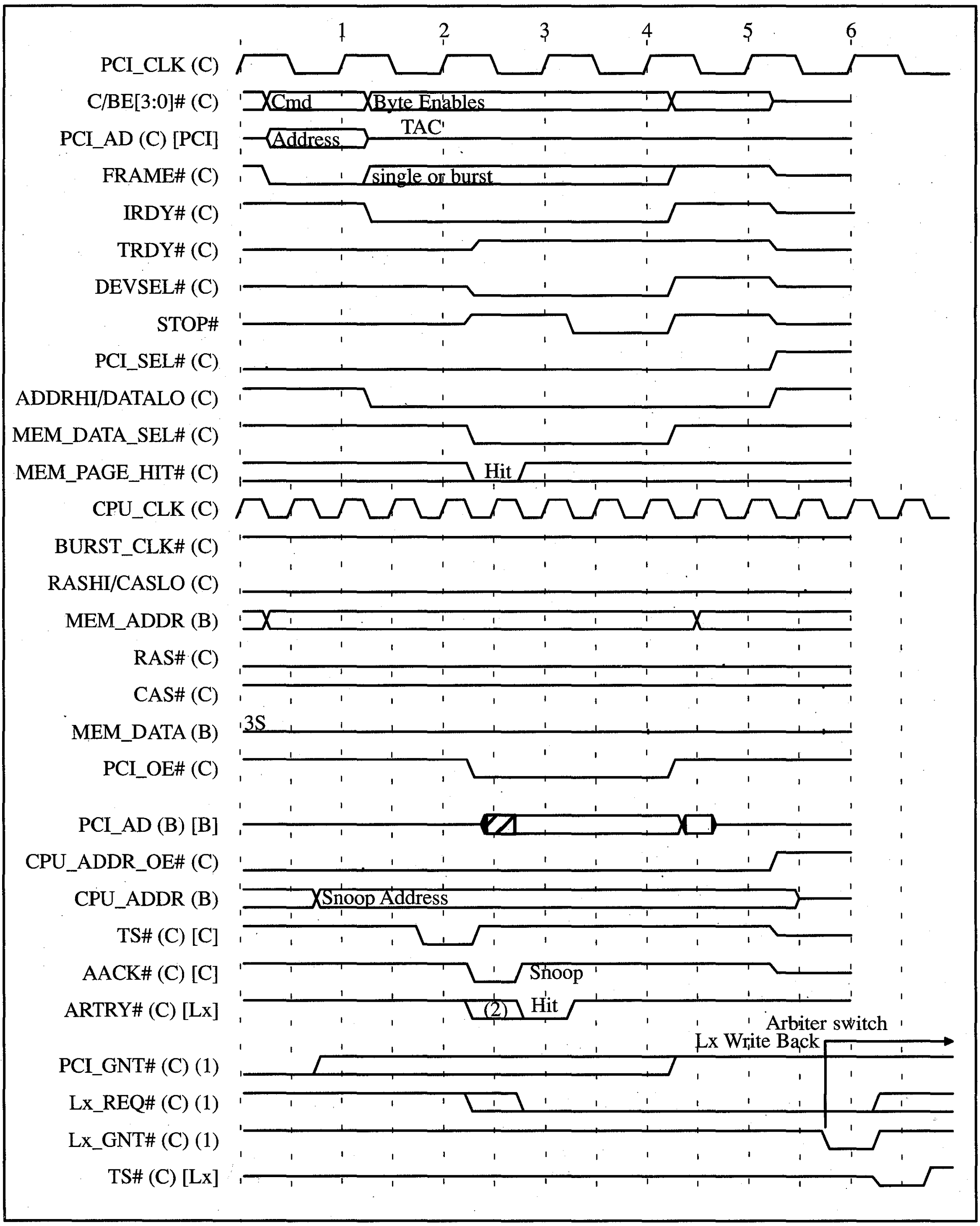


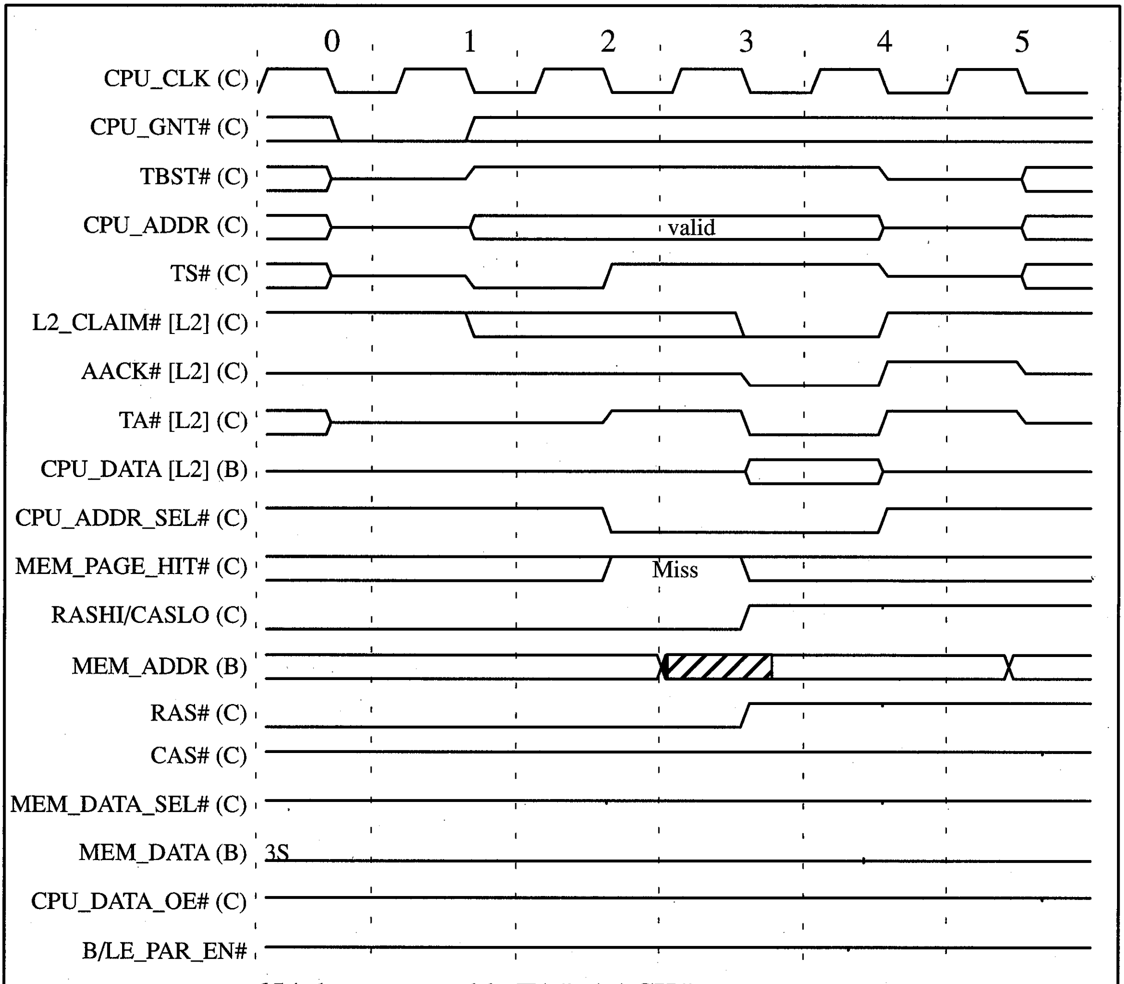
Figure 5–20. PCI to Memory Read – Cache Hit Timing Diagram

]

5.7.6 Example of a CPU to Memory Read Transfer With Page Miss and L2 Cache Hit

The single-beat CPU to memory read transfer shown in Figure 5–21 is typical of a CPU to memory read with page miss during cycles 0 through 2. By cycle 3 however, the L2 has driven L2_CLAIM# active. During cycle 3, the example L2 drives the requested data onto the 60X data bus, and asserts TA# and AACK# for one cycle, completing the transfer. If this was a burst transfer, the example L2 would have kept TA# active for three more cycles while delivering three more beats of data.

Since a page miss occured on this transfer, RAS# and RASHI/CASLO were sent high in cycle 3. These two signals are left high. The 654 Controller leaves AACK# and TA# tri-stated, and does not assert CPU_DATA_OE#.



654 does not enable TA#, AACK#
654 does not assert CPU_DATA_OE#

Figure 5–21. CPU to Memory Read – Page Miss, Cache Hit Timing Diagram

5.8 The System Error Handler

Table 5–23 summarizes the 654 Controller responses to system errors. The 654 Controller asserts TT_ERR#, DPE_ERR#, MEM_PAR_ERR#, or ALL_ONES_SEL# in response to an illegal operation or error condition.

The 654 Controller asserts TT_ERR# to report transfer type errors. In response to an error address read transaction from the 60X CPU (a load word instruction in the range 3G – 8M to 3G with CPU_ADDR[19] equal to 0), the 654 asserts ERR_ADDR_SEL# to read out the transfer type error address. TT_ERR# is asserted by the 654 until the processor performs an error address read transaction.

The 654 Controller asserts MEM_PAR_ERR# to report memory parity errors. In response to an error address read transaction from the 60X CPU, the 654 asserts ERR_ADDR_SEL# to read out the parity error address. MEM_PAR_ERR# is asserted by the 654 until the processor performs an error address read transaction.

The 654 Controller does not report an out-of-bounds memory access from the processor as an error. (Memory reads return 64 one-bits.) The 654 stops out-of-bounds PCI to memory cycles with a target abort protocol.

5.8.1 TEA# Error Reporting

The 654 Controller asserts TEA# (transfer error acknowledge) instead of TA# (transfer acknowledge) when various conditions are detected. TEA# can be masked by the MASK_TEA# signal. When MASK_TEA# is asserted, TA# overrides TEA# in all circumstances except the PIO cycle error (when XATS# is asserted). MASK_TEA# can be useful for debugging system errors.

TEA# is asserted to terminate a 60X CPU cycle instead of TA# under the following circumstances:

- On illegal transfer type errors including PIO cycles (XATS# is asserted)
- DPE# data parity errors from the L2 cache—if the error is on the first beat of a burst (INT_CPU# is used for the second, third, and fourth beats. See Section 5.8.2)
- Memory parity errors reported by the 653 Buffer on CPU accesses to system memory
- PCI target aborts
- PCI target timeouts—no response within 60us of PCI_DEVSEL# with PCI at 33MHz
- No PCI_DEVSEL# from a PCI target within seven PCI clocks from the assertion of PCI_FRAME# (except on PCI configuration cycles)
- Illegal transfer sizes and alignments of 60X CPU cycles
- As the response to a PCI Interrupt acknowledge cycle from the 60X CPU following any of the conditions in Section 5.8.2

5.8.2 Interrupt Reporting

The 654 Controller asserts the interrupt signal, INT_CPU#, to the 60X CPU to initiate a cycle so that TEA# can be reported based on the following circumstances:

- When DPE# is asserted by the CPU (two clocks after TA#) on the second, third, or fourth beats of a CPU burst read of an L2 cache.
- Memory parity error on PCI bus mastered cycle. (All subsequent PCI accesses to memory are target aborted.)
- Non-maskable interrupt from the I/O bridge.

INT_CPU# is asserted until the processor initiates a read transaction with the PCI interrupt acknowledge address. ALL_ONES_SEL# is driven during the interrupt acknowledge cycle, which causes all one-bits to be read as the address, along with TEA# (if MASK_TEA# is deasserted).

Table 5–23. System Error Reporting

Activity	Description	654 Controller Response	Error Status Signals Asserted
601-Initiated Transactions	Memory Out-of-Range Read	TA#, ALL_ONES_SEL#	none
	Memory Out-of-Range Write	TA# (no write occurs)	none
	2-Byte transfer with A29-31 = 111	TEA#	TT_ERR#
	3-Byte transfer with A29-31 = 110, 111		
	4-Byte transfer with A29-31 = 101, 110, 111		
	5, 6, or 7-Byte transfer		
	8-Byte transfer with A29–31 not = 000		
	Parity Error from 653 Buffer	TEA#	MEM_PAR_ERR#
	Processor/L2 DPE#	TEA# (or INT_CPU#)	DPE_ERR# (pulsed)
	PIO Cycle: XATS# asserted	TEA#	TT_ERR#
	ecowx	TA#	none
	eciwx	TA#, ALL_ONES_SEL#	none
601 to PCI	Master Abort (except Config.)	TEA#, ALL_ONES_SEL# (read)	TT_ERR#
	Configuration R/W Master Abort	TA#, ALL_ONES_SEL# (read)	none
	Target Abort	TEA#	TT_ERR#
	Target Retry	ARTRY#	none
PCI to Memory	Master Abort	none	none
	Memory out-of-bounds	Target abort, INT_CPU#	TT_ERR#
	Parity Error	Drive PCI_PAR invalid on current cycle. Target abort all following cycles until a read of the error address register, INT_CPU#.	MEM_PAR_ERR#
NMI	Non-maskable interrupt	INT_CPU#	none
Interrupt Acknowledge Cycle	Due to above error conditions and NMI	TEA#	no change
Read Error Address Latch	Due to above error conditions	TA#, ERR_ADDR_SEL#	Deassert TT_ERR# and MEM_PAR_ERR#

5.8.3 Saving Memory Parity Error Addresses

Memory parity generation and checking is supported within the 653 Buffer and controlled by the 654 Controller in conjunction with transfer type errors as shown in Figure 5–22. The 653 Buffer has an internal register that stores the address on memory parity errors. Each processor or PCI to memory access latches the address onto the error address register within the 653 Buffer as long as the signal L_ERR_ADDR# is asserted. On a memory access with a parity error from the 653 Bridge (MEM_PAR_GOOD deasserted), MEM_PAR_ERR# is asserted low to inhibit latching new addresses into the 653 Buffer error address register by means of the support gate shown in Figure 5–22.

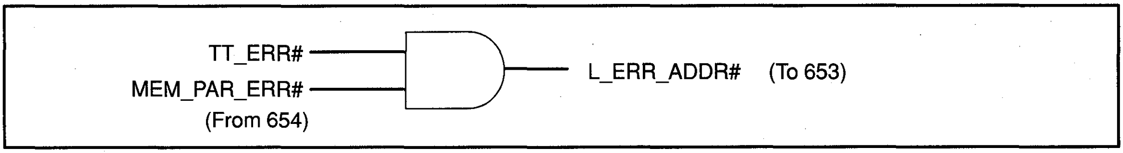


Figure 5–22. Error Address External Support Gate

MEM_PAR_ERR# remains asserted until the end of the cycle that the 60X CPU accesses the contents of the 653 Buffer error address register. The 654 Controller asserts ERR_ADDR_SEL# to the 653 Buffer when the 60X CPU reads the error address register. The stored error address is then read back to the CPU.

The 653 Buffer asserts the error address register on both words of the double-word 60X CPU data bus so that reads of either word get the error address. At the end of the error address read cycle, MEM_PAR_ERR#, TT_ERR# and ERR_ADDR_SEL# are all deasserted.

The 654 Controller does not report an out-of-bounds memory access from the processor, but it does stop an out-of-bounds PCI to system memory cycle with the target abort protocol.

5.8.4 Data Parity Error (DPE_ERR#)

The 654 Controller asserts DPE_ERR# in response to a data parity error (DPE#) from an L2 cache access. DPE_ERR# is asserted for two 60X CPU bus clocks for each data parity error detected.

5.8.5 Transfer Type Error

The 654 Controller asserts TT_ERR# to report transfer type errors. TT_ERR# is asserted until the 60X CPU accesses the error address register within the 653 Buffer.

Transfer type error (TT_ERR#) cycles include:

- PIO cycles—XATS# asserted (not masked by MASK_TEA#)
- A 60X CPU transfer size of five, six, or seven bytes to system memory or PCI
- A 60X CPU transfer size of two, three, or four bytes crossing a word boundary when target is PCI or system memory
- A 60X CPU transfer size of five to eight bytes when target is PCI
- A 60X CPU burst when the target is PCI
- A PCI target abort
- No PCI_DEVSEL# from target—except on PCI configuration cycles
- A PCI target system time-out—if no response 60us (33MHz PCI) after PCI_DEVSEL#

The 654 Controller does not report an out-of-bounds memory access from a PCI master to system memory, but it does stop an out-of-bounds PCI cycle to system memory with target abort protocol.

On detection of a transfer-type error, the 654 Controller asserts TT_ERR# to inhibit latching new addresses into the 653 Buffer error address register (see Figure 5–22). TT_ERR# remains asserted until the end of the cycle when the 60X CPU reads the error address register.

The 654 Controller asserts ERR_ADDR_SEL# to the 653 Buffer when the 60X CPU reads the error address register. The 653 Buffer asserts the error address register on both words of the 60X CPU data bus. At the end of the error address read cycle, TT_ERR# and ERR_ADDR_SEL# are deasserted.

5.8.6 Illegal PCI Operations

If the PCI bus runs a cycle to an out-of-bounds system memory address, the 654 Controller uses the target abort protocol to stop the PCI cycle.

5.8.7 Non-Maskable Interrupt (NMI_REQ)

The 654 Controller asserts INT_CPU# in response to NMI_REQ. When the 60X CPU drives a PCI interrupt acknowledge transaction back in response to INT_CPU#, the 654 Controller immediately asserts TEA# and 64 one-bits on the CPU data bus without generating a PCI interrupt acknowledge transaction. An interrupt acknowledge cycle that is immediately terminated by TEA# is the result of one of the conditions listed in Section 5.8.2.

Section 6
Electrical Characteristics

Unless otherwise noted, all specifications in this section apply to both the 653 Buffer and to the 654 Controller.

6.1 Absolute Maximum Ratings

Stresses in excess of those listed in Table 6–1 may damage and/or decrease the reliability of the 650 Bridge. Additionally, stressing the 650 Bridge in excess of the conditions listed as *Recommended Operating Conditions* is neither intended nor supported. All voltages are referenced to ground (V_{SS}).

Table 6–1. Absolute Maximum Ratings, 650 Bridge

Symbol	Parameter	Min	Max	Units
Tjst	Junction Temperature, Storage	-40	125	deg C
Tjp	Junction Temperature, Power Applied	-25	100	deg C
V_{DD}	Supply Voltage	2.7	3.9	V
Vi	DC Voltage Applied to Any Input	–.5	5.5	V
Vo	DC Voltage Applied to Any Output (Output Tri-stated)	–.5	5.5	V
	ESD Withstand	2.2	—	kV
	Latchup current	100	—	mA

6.2 Recommended Operating Conditions

This section lists the conditions under which the 650 Bridge is intended to operate.

6.2.1 Signal And Temperature Ranges

Table 6–2. Recommended Operating Conditions, 650 Bridge

Symbol	Parameter	Min	Max	Units	Notes
V_{DD}	Supply Voltage	3.0	3.8	v	
V_I	DC Voltage Applied to Any Input Pin	–.5	5.5	v	(1)
V_O	DC Voltage Applied to Any Output Pin	–.5	5.5	v	(2)
Top	Junction Temperature, Operating	10	85	deg C	

Notes For Table 6–2:

1) Allowed range of DC voltage applied to any I/O pin in input mode or to any input pin. The pins shown as Type = PCI may conduct excess current if forced above V_{DD} + 1.5v.
2) Allowed range of DC voltage applied to any output pin while the output is tri-stated. The pins shown as Type = PCI may conduct excess current if forced above V_{DD} + 1.5v.

6.2.2 Power Dissipation

Neither the 653 Buffer nor the 654 Controller are expected to require a heat sink, when used in the manner described in this manual. However, thermal management is a complex discipline, and the application is the responsibility of the designer.

Table 6–3 lists the power supply current and power dissipation under various conditions for the 653 Buffer and the 654 Controller.

Table 6–3. Power Dissipation (See Note 1)

Parameter		Current (mA)		Power (mW)		Notes
		Typ	Max	Typ	Max	Notes
Power Supply Current – System quiescent.	653	43.7	—	157	—	(2)
	654	8.0	—	29	—	(2)
Power Supply Current – Memory accesses with PCI bus idle.	653	48.0	—	173	—	(3)
	654	19	—	68	—	(3)
Power Supply Current – Worst case activity.	653	59	118	212	425	(4)
	654	63	126	227	454	(4)

Notes for Table 6–3:

1) 60X CPU bus running at 66MHz, PCI bus running at 33MHz and 3.3v, L2 Cache installed, 5 PCI loads, 8 DRAM loads, 653 Buffer and 654 Controller at V_{DD} = 3.6v. This data assumes that the 650 Bridge is connected in a manner similar to that shown in the example system.
2) 60X CPU executing from L1 cache, DRAM refresh enabled, PCI bus idle.
3) 60X CPU executing repeated burst memory transfers.

4) A mix of instruction fetches and PCI accesses with addresses broadcast to the CPU bus for snooping
5) The 653 Buffer is supplied in a 304 pin Ceramic (C4) Quad Flat Pack
6) The 654 Controller is supplied in a 160 pin Plastic Quad Flat Pack

6.2.3 Thermal Characteristics

Table 6–4 shows typical thermal resistances associated with the 653 Buffer and the 654 Controller. Each row shows data for a given air flow condition at the chip package. The row titled *Convection* shows data for the chip package in free air with no forced air cooling, with the package mounted horizontally on the upper surface of a PCB. The other rows show data for a variety of forced air flow conditions. The values shown do not include a significant amount of heat flow through the pins of the chip, either to or from the PCB.

From Table 6–3, the worst case power dissipation of either the 653 Buffer or the 654 Controller is less than .5 W. Using T_J(max) = 85 deg C, and T_A = 50 deg C;

$$\Theta\text{j–a (max)} = \frac{T_J - T_A}{\text{Pd}} = \frac{85\text{ C} - 50\text{ C}}{.5\text{ W}} = 70\text{ deg C/W}$$

Examination of Table 6–4 shows that neither the 653 Buffer or the 654 Controller require a heat sink, even with worst case power dissipation, to maintain a junction temperature of less than 85 deg C in free air at 50 deg C ambient.

Table 6–4. Typical Thermal Resistance, Junction to Ambient, No Heat Sink

Airflow	Θj–a, 653 Buffer	Θj–a, 654 Controller	Units
Convection	33.4	54	deg C/W
.25 M/s (50fpm)	29.7	47	deg C/W
.5 M/s (100fpm)	26.8	44	deg C/W
1 M/s (200fpm)	24.0	40	deg C/W

6.3 Common Characteristics

The specifications shown in Table 6–5 are common to both the 653 Buffer and the 654 Controller.

Table 6–5. 650 Bridge Common Characteristics (See Note 1)

Symbol	Parameter	Type	Min	Max	Units	Notes
V_{IL}	Input Low Voltage	All	—	.8	v	(2)
V_{IH}	Input High Voltage	All	2.0	—	v	(2)
I_{IL}	Input Leakage Current	All	—	1	uA	(2)
V_{OL}	Output Low Voltage	TTL	—	.40	v	(3)
		PCI	—	.55	v	(3)
V_{OH}	Output High Voltage	TTL	2.4	—	v	(3)
		PCI	2.4	—	v	(3)
I_{O3S}	Output Tri-state Leakage Current	TTL	—	10	uA	(3)
		PCI	—	70	uA	(3)

Notes for Table 6–5:
1) Over Recommended Operating Conditions.
2) Values apply to each I/O pin in input mode and to each input pin.
3) Values apply to each output pin and to each I/O pin in output mode.

6.4 Package/Pin Electrical Characteristics

6.4.1 653 Buffer Model

The electrical model of the effects of package and pin parasitic effects on the 653 Buffer is shown in Figure 6–1. The ranges of values shown are nominal only, are not guaranteed, and vary from pin to pin. The lower values are typical of pins that are located on the side of the package and which are closest to the chip. The higher values are typical of corner pins. Note that the C1 capacitance is the value shown for DPC (Die Pad Capacitance) in Table 6–6 (which is due to the I/O book). C2 represents a distributed capacitance, and L1 represents a lumped loop inductance which includes the effects of inductance from the driver book to the power supply pins.

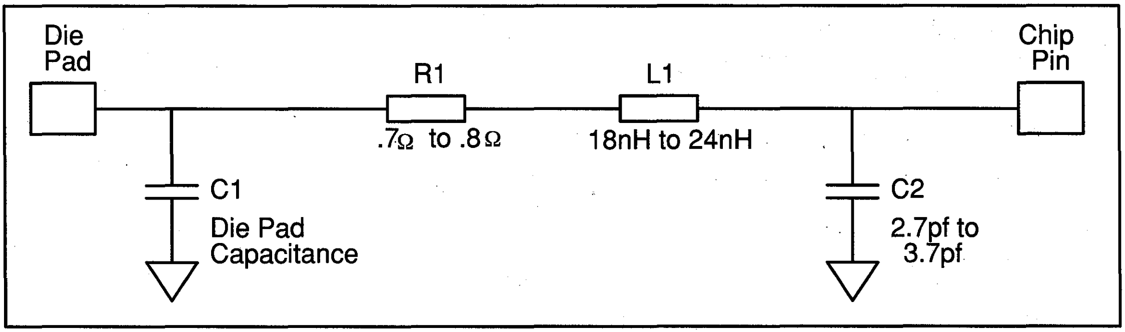


Figure 6–1. 653 Package/Pin Electrical Model

6.4.2 654 Controller Model

The electrical model of the effects of package and pin parasitic effects on the 654 Controller is shown in Figure 6–2. The values shown are nominal values only, are not guaranteed, and vary from pin to pin. Values lower than nominal are typical of pins located on the side of the package and which are closest to the chip. Values higher than nominal are typical of corner pins. Typical ranges are proportionally similar to those shown for the 653. Note that the total output capacitance of the die pad is derived by adding the value shown in Figure 6–2 (which is due to the package) to the value shown for DPC (Die Pad Capacitance) in Table 6–7 (which is due to the I/O book).

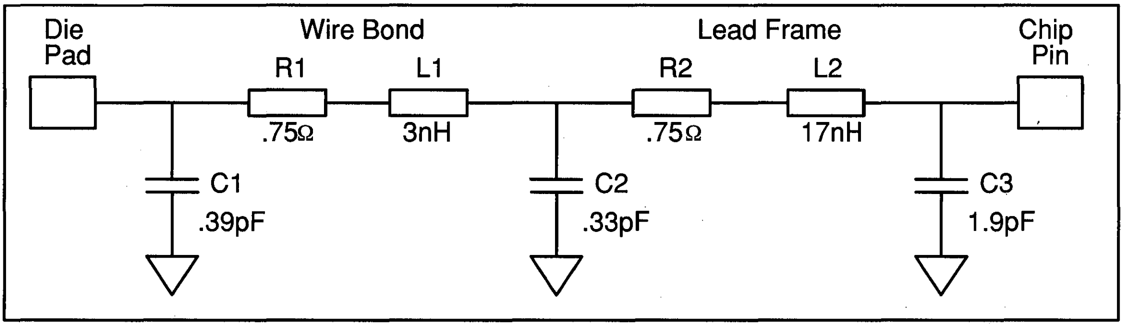


Figure 6–2. 654 Package/Pin Electrical Model

6.5 653 Buffer DC Characteristics By Signal

Table 6–6. 653 Buffer DC Characteristics By Signal (See Note 1)

Signal	Pin	I/O	Book,	Type	P/L	I_{OL}	I_{OH}	DPC (4)	
			Pad (2)	(3)	(2)	(mA)	(mA)	Min	Max
ADDRHI/DATALO	75	I	CBJD,B0	TTL	A	—	—	.80	.95
ALL_ONES_SEL#	10	I	CBSX,B0	TTL	A	—	—	.80	.95
BURST_CLK#	73	I	CBJD,B0	TTL	A	—	—	.80	.95
CONTIG_IO	34	I	CBJE,B0	TTL	F	—	—	.80	.95
CPU_ADDR [0:2, 21:22]		I/O	CBNS,11	TTL	C	6	4	3.4	4.0
CPU_ADDR [12, 23:31]		I/O	CBNS,10	TTL	C	12	8	3.7	4.3
CPU_ADDR [3:11, 13:20]		I/O	CBNS,11	TTL	B	6	4	2.9	3.5
CPU_ADDR_OE#	159	I	CBJE,B0	TTL	F	—	—	.80	.95
CPU_ADDR_SEL#	158	I	CBJE,B0	TTL	F	—	—	.80	.95
CPU_DATA [0:63]		I/O	CBNS,10	TTL	B	12	8	3.2	3.8
CPU_DATA_OE#	192	I	CBJE,B0	TTL	F	—	—	.80	.95
CPU_DATA_SEL#	168	I	CBJD,B0	TTL	A	—	—	.80	.95
DRAMX9HI/X10LO	65	I	CBJD,B0	TTL	A	—	—	.80	.95
ERR_ADDR_SEL#	223	I	CBSZ,B0	TTL	A	—	—	.80	.95
GND (35 pins)		—	—	—	—	—	—	—	—
L_ERR_ADDR#	6	I	CBSY,B0	TTL	A	—	—	.80	.95
L_PCI_DATA#	67	I	CBJD,B0	TTL	A	—	—	.80	.95
LE_MODE_SEL#	262	I	CBJE,B0	TTL	F	—	—	.80	.95
MEM_ADDR [11:0]		O	CBNO,10	TTL	C	12	8	3.5	4.1
MEM_ADDR0_B	247	O	CBNO,10	TTL	C	12	8	3.5	4.1
MEM_DATA [63:0]		I/O	CBNS,10	TTL	B	12	8	3.2	3.8
MEM_DATA_OE#	7	I	CBJE,B0	TTL	F	—	—	.80	.95
MEM_DATA_SEL#	72	I	CBJD,B0	TTL	A	—	—	.80	.95
MEM_PAGE_HIT#	64	O	CBNO,11	TTL	C	6	4	3.2	3.8
MEM_PAR [7:0]		I/O	CBNS,10	TTL	C	12	8	3.7	4.3
MEM_PAR_GOOD	71	O	CBNQ,10	TTL	C	12	8	3.5	4.1
NO_TRANS	43	I	CBJD,B0	TTL	A	—	—	.80	.95
PCI_AD [31:0]		I/O	CBUK,10	PCI	B	6	6	3.1	3.7

Table 6–6. 653 Buffer DC Characteristics By Signal (See Note 1) (Continued)

Signal	Pin	I/O	Book, Pad (2)	Type (3)	P/L (2)	I_{OL} (mA)	I_{OH} (mA)	DPC (4)	
								Min	Max
PCI_AD_PAR	63	O	CBNO,12	TTL	C	4	4	2.9	3.5
PCI_CLK	70	I	CBJD,B0	TTL	A	—	—	.80	.95
PCI_OE#	11	I	CBJE,B0	TTL	F	—	—	.80	.95
PCI_SEL#	74	I	CBJD,B0	TTL	A	—	—	.80	.95
RASHI/CASLO	203	I	CBJD,B0	TTL	A	—	—	.80	.95
REFRESH_SEL#	76	I	CBJD,B0	TTL	A	—	—	.80	.95
ROM_SEL#	66	I	CBJE,B0	TTL	F	—	—	.80	.95
TEST#	222	I	CBSW,B3	TTL	A	—	—	.80	.95
TSIZ [0:2]		I	CBJD,B0	TTL	A	—	—	.80	.95
V_{DD} (27 pins)		V_{DD}	—	—	—	—	—	—	—

Notes for Table 6–6:

1) Values apply over recommended operating conditions.

2) The Book, Pad, and P/L (performance level) define the I/O pin driver/receiver type, characteristics, and speed. More information on these items is contained in the *IBM CMOS4LP Logic Products Databook (8/93)*, Document Number ADCC4LDBU–01.

3) See Section 6.3, Common Characteristics.

4) Die Pad Capacitance. The equivalent capacitance to ground of the die pad attachment as a function of the I/O book circuitry. To model the electrical path from the I/O book to the circuit board pad, see Section 6.4.

6.6 654 Controller DC Characteristics By Signal

Table 6–7. 654 Controller DC Characteristics By Signal

Signal	Pin	I/O	Book,	Type	P/L	I_{OL}	I_{OH}	DPC (4)		POR
			Pad (3)	(2)	(3)	(mA)	(mA)	Min	Max	(5)
Processor Signals										
AACK#	113	I/O	CBND,10	TTL	C	24	18	3.6	4.4	Z
ARTRY#	112	I/O	CBND,10	TTL	C	24	18	3.6	4.4	Z
CPU_ADDR [0:8, 19, 29–31]		I	CBJE,B0	TTL	F	—	—	.80	.95	X
CPU_GNT#	133	O	CBMZ,13	TTL	C	24	18	3.6	4.4	0
CPU_REQ#	126	I	CBJE,B0	TTL	F	—	—	.80	.95	1
DPE#	139	I	CBJE,B0	TTL	F	—	—	.80	.95	1
INT_CPU#	83	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
MASK_TEA#	77	I	CBJE,B0	TTL	F	—	—	.80	.95	X
SRESET_CPU#	37	O	CBNQ,16	TTL	B	8	6	2.7	3.3	1
TA#	134	I/O	CBND,10	TTL	C	24	18	3.6	4.4	Z
TBST#	127	I	CBNU,16	TTL	B	—	—	3.7	4.3	1
TEA#	135	O	CBMZ,13	TTL	C	24	18	3.6	4.4	Z
TS#	132	I/O	CBND,10	TTL	C	24	18	3.6	4.4	Z
TSIZ[0:2]		I/O	CBNU,16	TTL	B	8	6	2.9	3.5	Z
TT[0,2,3]		I/O	CBNU,16	TTL	B	8	6	2.9	3.5	Z
TT[1]	122	I/O	CBNU,16	TTL	C	8	6	3.7	4.3	Z
XATS#	125	I	CBJE,B0	TTL	F	—	—	.80	.95	1
L2 Cache Signals										
L2_CACHE_GNT#	114	O	CBNQ,B0	TTL	C	4	4	2.9	3.5	1
L2_CACHE_REQ#	109	I	CBJE,B0	TTL	F	—	—	.80	.95	1
L2_CLAIM#	124	I	CBJE,B0	TTL	F	—	—	.80	.95	1
L2_PRESENT#	82	I	CBJE,B0	TTL	F	—	—	.80	.95	1
PCI Sideband Signals (Incident Wave)										
PCI_CLK	129	I	CBUM,10	PCI	B	—	—	3.1	3.7	RUN
PCI_GNT[1:5]#		O	CBNQ,17	TTL	C	8	6	3.5	4.1	1
PCI_REQ[1:5]#		I	CBUM,10	PCI	B	—	—	3.1	3.7	1
PCI Bus Signals (Reflected Wave)										
PCI_C/BE[3:0]#		I/O	CBUM,10	PCI	B	6	6	3.1	3.7	Z
PCI_DEVSEL#	28	I/O	CBUM,10	PCI	B	6	6	3.1	3.7	Z

Table 6–7. 654 Controller DC Characteristics By Signal (Continued)

Signal	Pin	I/O	Book,	Type	P/L	I_{OL}	I_{OH}	DPC (4)		POR
			Pad (3)	(2)	(3)	(mA)	(mA)	Min	Max	(5)
PCI_FRAME#	29	I/O	CBUM,10	PCI	B	6	6	3.1	3.7	Z
PCI_IRDY#	52	I/O	CBUM,10	PCI	B	6	6	3.1	3.7	Z
PCI_PAR	53	O	CBUI,10	PCI	C	6	6	2.8	3.4	X
PCI_STOP#	16	I/O	CBUM,10	PCI	B	6	6	3.1	3.7	Z
PCI_TRDY#	27	I/O	CBUM,10	PCI	B	6	6	3.1	3.7	Z
I/O Bus and I/O Bridge Signals										
IO_BRDG_GNT#	7	O	CBNQ,17	TTL	C	8	6	3.5	4.1	1
IO_BRDG_HOLD#	45	I	CBUM,10	PCI	B	—	—	3.1	3.7	1
IO_BRDG_IRQ	32	I	CBJE,B0	TTL	F	—	—	.80	.95	0
IO_BRDG_REQ#	39	I	CBUM,10	PCI	B	—	—	3.1	3.7	1
ISA_MASTER#	79	I	CBJE,B0	TTL	F	—	—	.80	.95	X
NMI_IRQ	12	I	CBJE,B0	TTL	F	—	—	.80	.95	0
Test Signals										
DI#	117	I	CBJE,B0	TTL	B	—	—	.80	.95	1
RI#	75	I	CBJE,B0	TTL	E	—	—	.80	.95	1
TEST#	136	I	CBJE,B0	TTL	F	—	—	.80	.95	1
DRAM Memory Subsystem Signals										
BE_PAR_EN#	84	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
CAS[7:0]#		I/O	CBNU,10	TTL	C	12	8	3.7	4.3	1
LE_PAR_EN#	85	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
RAS[7:0]#		O	CBNQ,13	TTL	C	12	8	3.5	4.1	1
WE[1:0]#		O	CBNQ,13	TTL	C	12	8	3.5	4.1	1
Boot ROM Device Signals										
ROM_CS#	74	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
ROM_OE#	73	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
ROM_WE#	72	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
653 Buffer Signals										
ADDRHI/DATALO	147	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
ALL_ONES_SEL#	2	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
BURST_CLK#	8	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
CPU_ADDR_OE#	144	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1

Table 6–7. 654 Controller DC Characteristics By Signal (Continued)

Signal	Pin	I/O	Book,	Type	P/L	I_{OL}	I_{OH}	DPC (4)		POR
			Pad (3)	(2)	(3)	(mA)	(mA)	Min	Max	(5)
CPU_ADDR_SEL#	159	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
CPU_DATA_OE#	145	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
CPU_DATA_SEL#	158	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
DPE_ERR#	43	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
ERR_ADDR_SEL#	76	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
L_PCI_DATA#	146	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
LE_MODE_SEL#	92	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
MEM_DATA_OE#	154	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
MEM_DATA_SEL#	153	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
MEM_PAGE_HIT#	149	I	CBJE,B0	TTL	F	—	—	.80	.95	1
MEM_PAR_ERR#	42	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
MEM_PAR_GOOD	155	I	CBJE,B0	TTL	F	—	—	.80	.95	X
NO_TRANS	13	O	CBNQ,15	TTL	C	4	4	2.9	3.5	0
PCI_AD_PAR	152	I	CBJE,B0	TTL	F	—	—	.80	.95	X
PCI_OE#	148	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
PCI_SEL#	157	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
RASHI/CASLO	156	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
REFRESH_SEL#	9	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
ROM_SEL#	14	O	CBNQ,15	TTL	C	4	4	2.9	3.5	1
TT_ERR#	44	O	CBNQ,17	TTL	B	8	6	2.7	3.3	1
System Interface and Miscellaneous Signals										
CPU_CLK	142	I	CBJE,B0	TTL	F	—	—	.80	.95	RUN
LE_MODE_REQ#	150	I	CBJE,B0	TTL	F	—	—	.80	.95	X
MC_SETUP#	46	I	CBJE,B0	TTL	F	—	—	.80	.95	1
REFRESH_REQ#	87	I	CBJE,B0	TTL	F	—	—	.80	.95	1
RESET#	86	I	CBJE,B0	TTL	F	—	—	.80	.95	—
SRESET_REQ#	89	I	CBJE,B0	TTL	F	—	—	.80	.95	1

Notes for Table 6–7:

1) Values apply over recommended operating conditions.

2) See Section 6.3.

3) The Book, Pad, and P/L (performance level) define the I/O pin driver/receiver type, characteristics, and speed. More information on these items is contained in the *IBM CMOS4LP Logic Products Databook (8/93)*, Document Number ADCC4LDBU–01.

4) Die Pad Capacitance. The equivalent capacitance to ground of the die pad attachment as a function of the I/O book circuitry. To model the electrical path from the I/O book to the circuit board pad, see Section 6.4.

5) For inputs, this state must be held on the input during the inputs required interval of the POR sequence (the letter X indicates a don't care). For outputs, the 653 Buffer will drive this state onto the output pin during the outputs valid interval of the POR sequence (the symbol Z indicates that the output is tri-stated during POR). The 650 Bridge power on conditions are designed to be consistent with conditions produced by properly functioning 60X CPU, PCI bus agents, L2 cache, and other system components.

6.7 Output V—I Curves

6.7.1 PCI Local Bus Compatible Drivers

Inputs and outputs (drives) in the PCI group have input/output characteristics that comply with the DC specifications of both the 3.3v and 5v signalling environments defined for PCI Local Bus components in the *PCI Local Bus Specification, Revision 2.0.* These signals are identified in the DC Characteristics Tables as type PCI.

The following tables show V–I curves for the PCI bus output drivers contained in the 650 Bridge chipset. The curves labeled WC show driver characteristics of a worst case process variation device operating at 85 deg C with V_{DD} = 3v. The curves labeled BC show driver characteristics of a best case process variation device operating at 10 deg C with V_{DD} = 3.6v. The curves labeled NOM show driver characteristics of a nominal process device operating at 25 deg C with V_{DD} = 3.3v.

6.7.1.1 Pull Up Curves, PCI Drivers, P/L = A

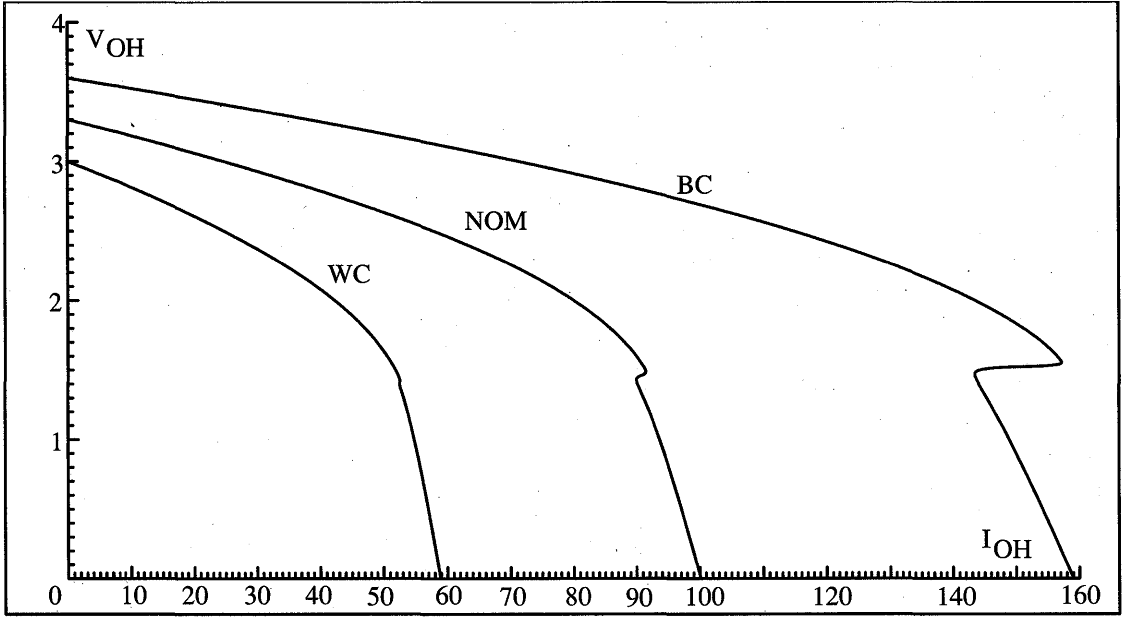


Figure 6–3. Pull Up Curves, PCI Drivers, P/L = A.

6.7.1.2 Pull Up Curves, PCI Drivers, P/L = B

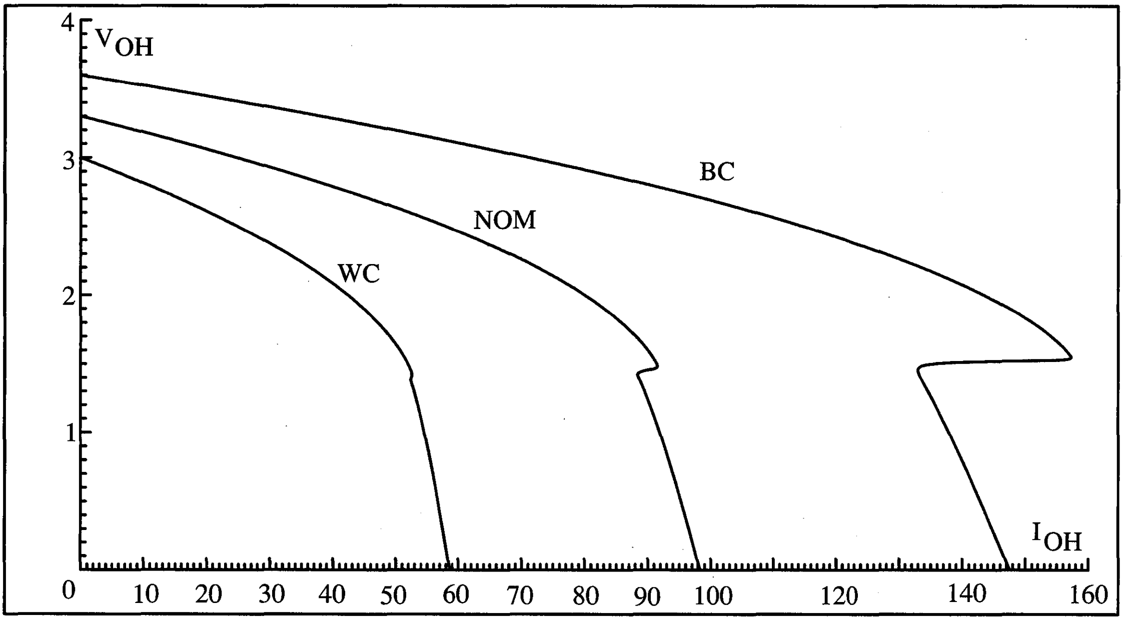


Figure 6–4. Pull Up Curves, PCI Drivers, P/L = B.

6.7.1.3 Pull Up Curves, PCI Drivers, P/L = C

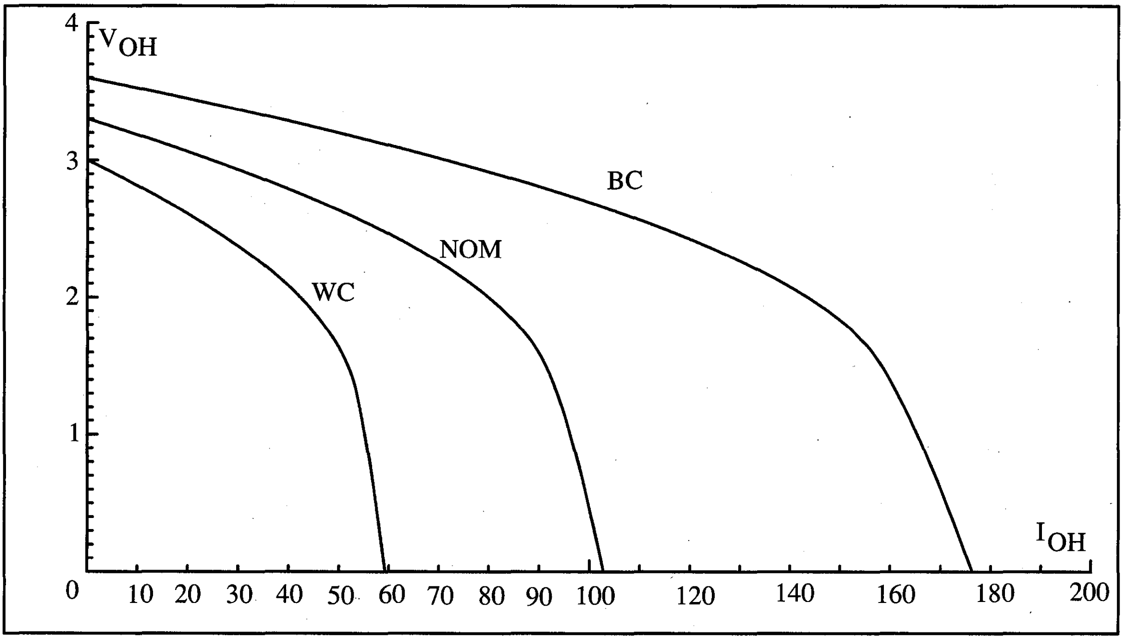


Figure 6–5. Pull Up Curves, PCI Drivers, P/L = C.

6.7.1.4 Pull Down Curves, PCI Drivers, P/L = A, B, and C

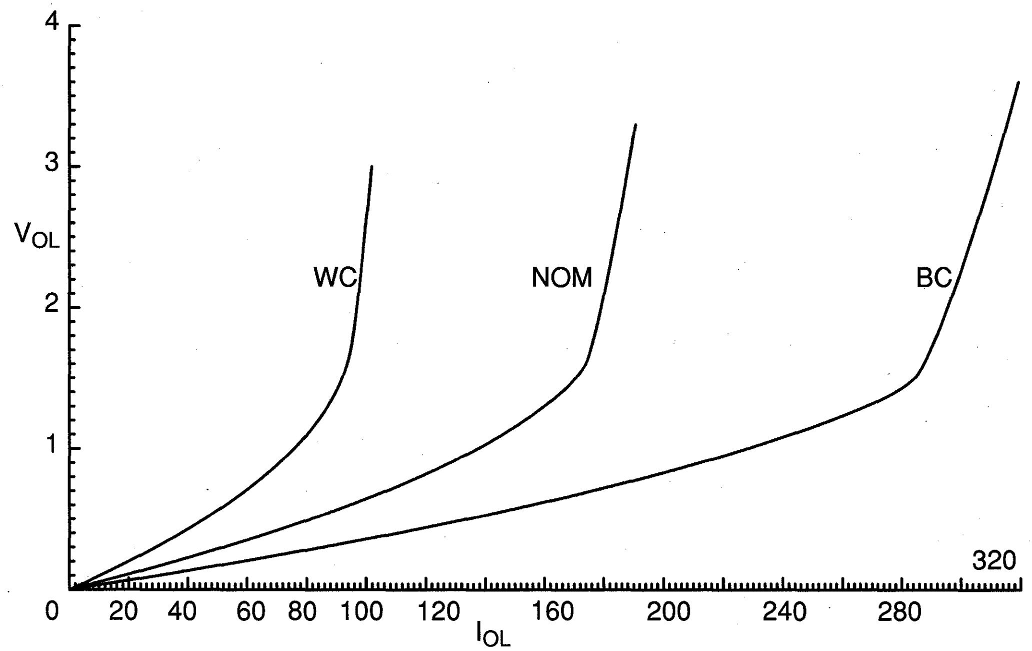


Figure 6–6. Pull Down Curves, PCI Drivers, P/L = A, B, and C.

6.7.2 TTL Driver Output Curves

Inputs and outputs (drives) in the TTL group have input/output characteristics that comply with common TTL switching levels, as shown in Table 6–5. The following tables show V–I curves for the TTL output drivers contained in the 650 Bridge chipset. The curves labeled WC show driver characteristics of a worst case process variation device operating at 85 deg C with V_{DD} = 3v. The curves labeled BC show driver characteristics of a best case process variation device operating at 10 deg C with V_{DD} = 3.6v. The curves labeled NOM show driver characteristics of a nominal process device operating at 25 deg C with V_{DD} = 3.3v.

6.7.2.1 Pull Down Curves, TTL Driver, I_{OL} = 4mA, P/L = A

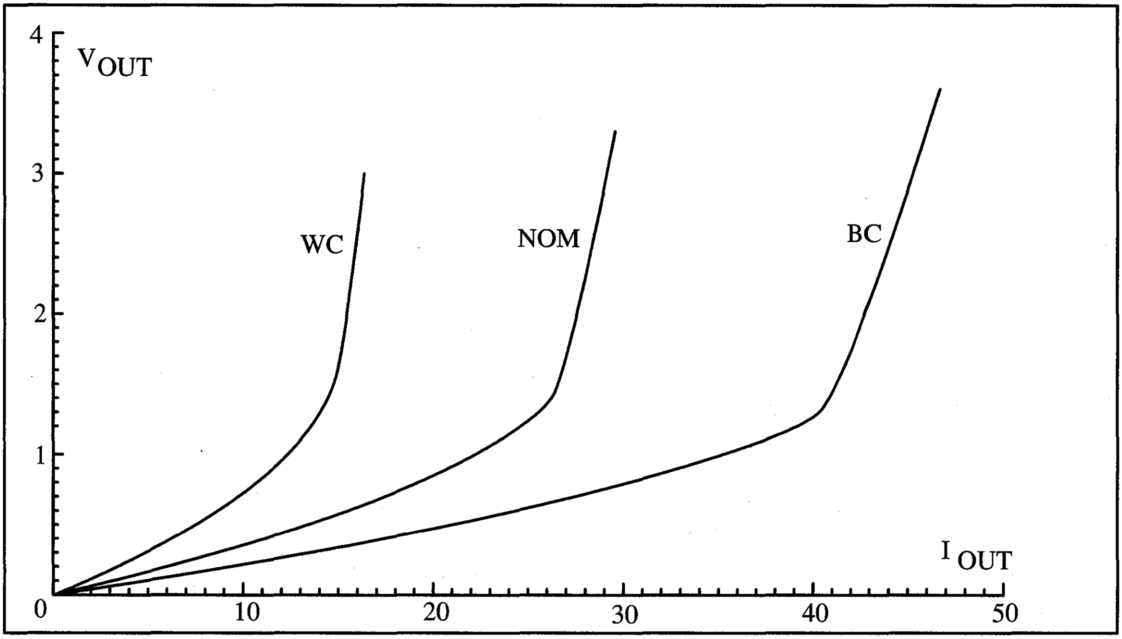


Figure 6–7. Pull Down Curves, TTL Driver, I_{OL} = 4mA, P/L = A

6.7.2.2 Pull Down Curves, TTL Driver, I_{OL} = 4mA, P/L = B

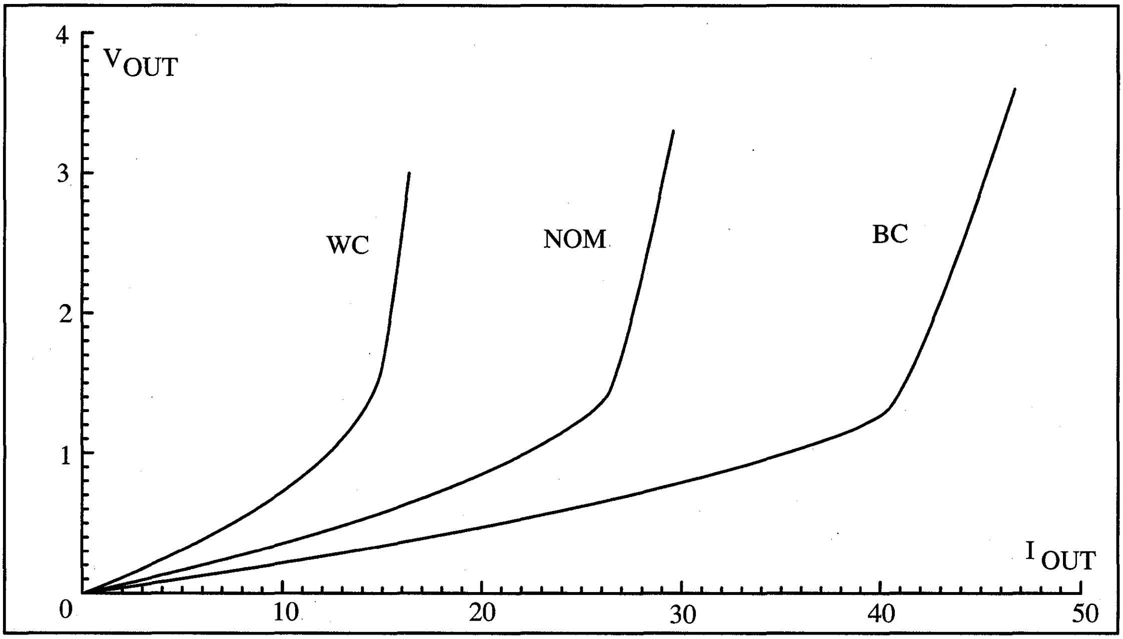


Figure 6–8. Pull Down Curves, TTL Driver, I_{OL} = 4mA, P/L = B

6.7.2.3 Pull Down Curves, TTL Driver, I_{OL} = 4mA, P/L = C

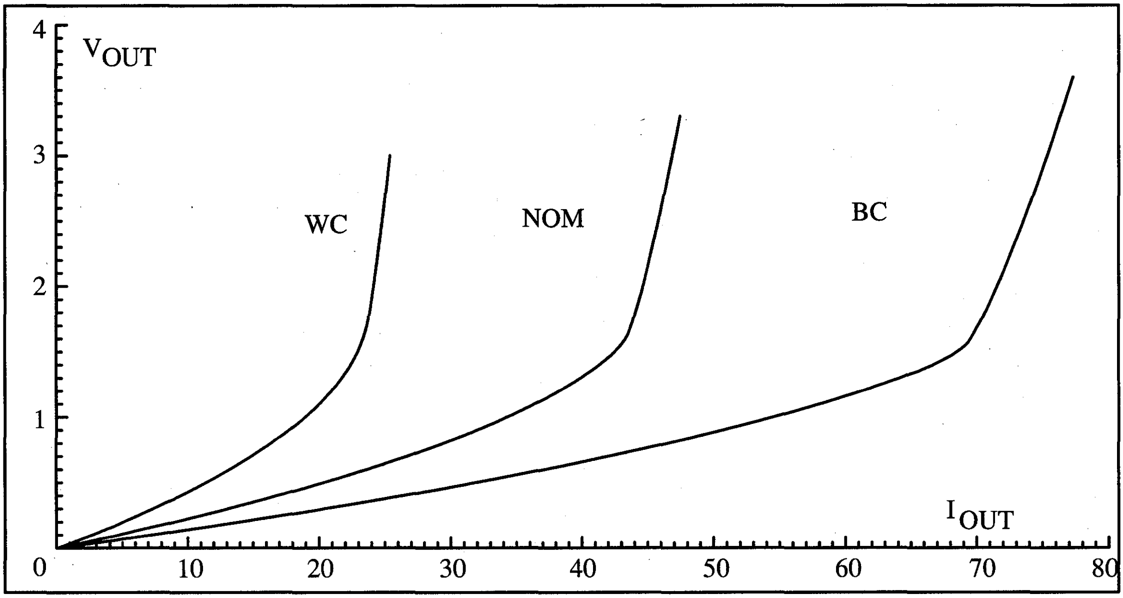


Figure 6–9. Pull Down Curves, TTL Driver, I_{OL} = 4mA, P/L = C

6.7.2.4 Pull Up Curves, TTL Driver, I_{OL} = 4mA, P/L = A, B, and C

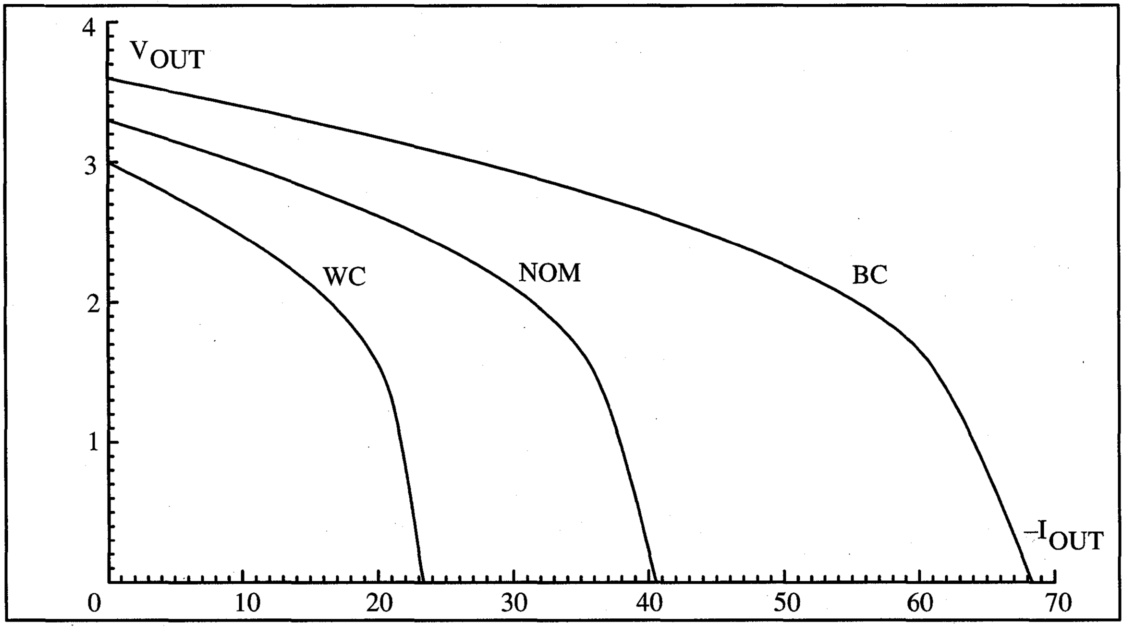


Figure 6–10. Pull Up Curves, TTL Driver, I_{OL} = 4mA, P/L = A, B, and C

6.7.2.5 Pull Down Curves, TTL Driver, I_{OL} = 6mA, P/L = A

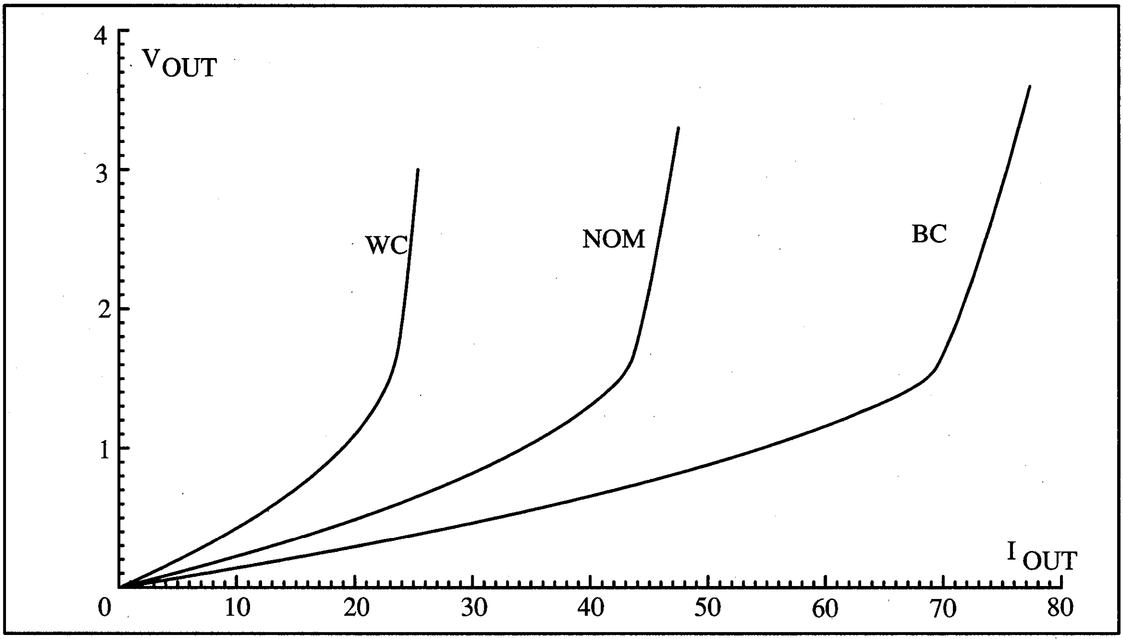


Figure 6–11. Pull Down Curves, TTL Driver, I_{OL} = 6mA, P/L = A

6.7.2.6 Pull Down Curves, TTL Driver, I_{OL} = 6mA, P/L = B

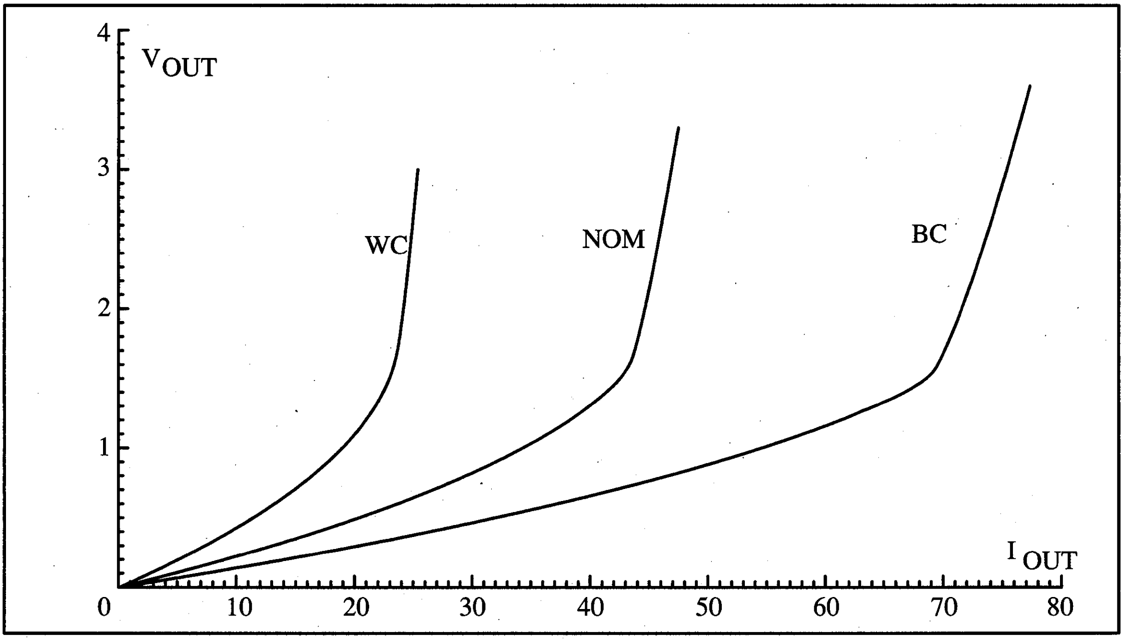


Figure 6–12. Pull Down Curves, TTL Driver, I_{OL} = 6mA, P/L = B

6.7.2.7 Pull Down Curves, TTL Driver, I_{OL} = 6mA, P/L = C

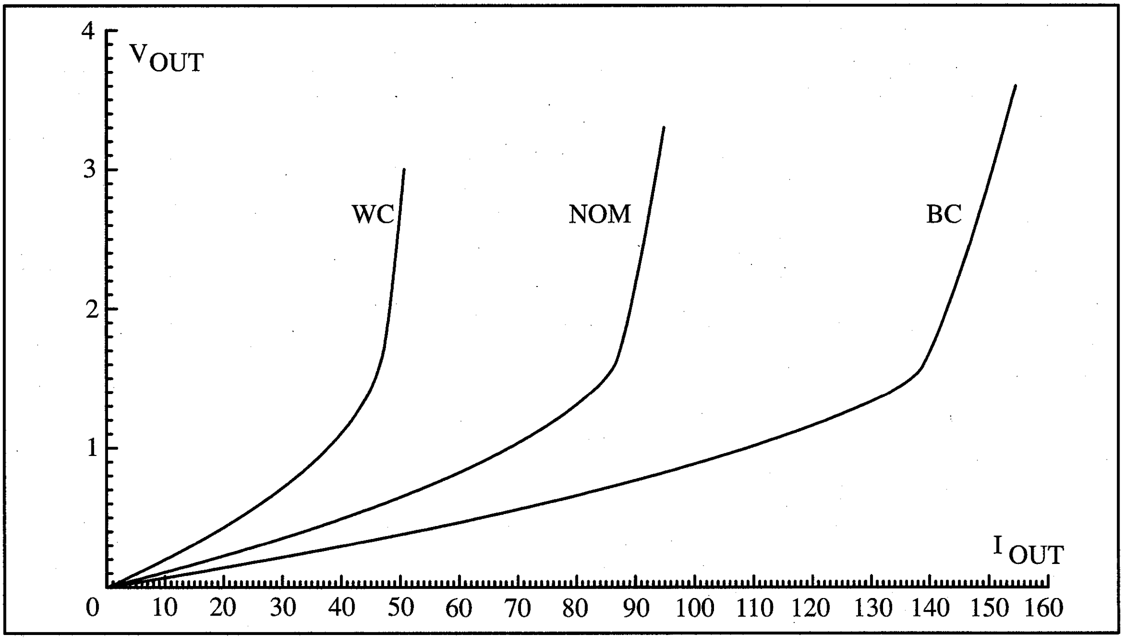


Figure 6–13. Pull Down Curves, TTL Driver, I_{OL} = 6mA, P/L = C

6.7.2.8 Pull Up Curves, TTL Driver, I_{OL} = 6mA, P/L = A, B, and C

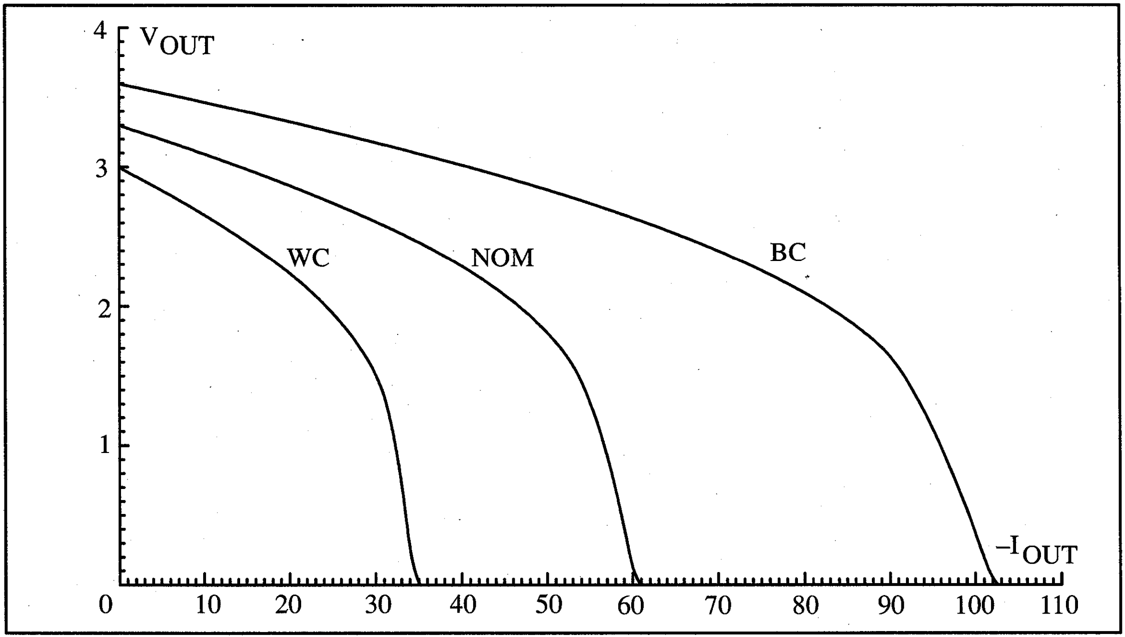


Figure 6–14. Pull Up Curves, TTL Driver, I_{OL} = 6mA, P/L = A, B, and C

6.7.2.9 Pull Down Curves, TTL Driver, I_{OL} = 8mA, P/L = A

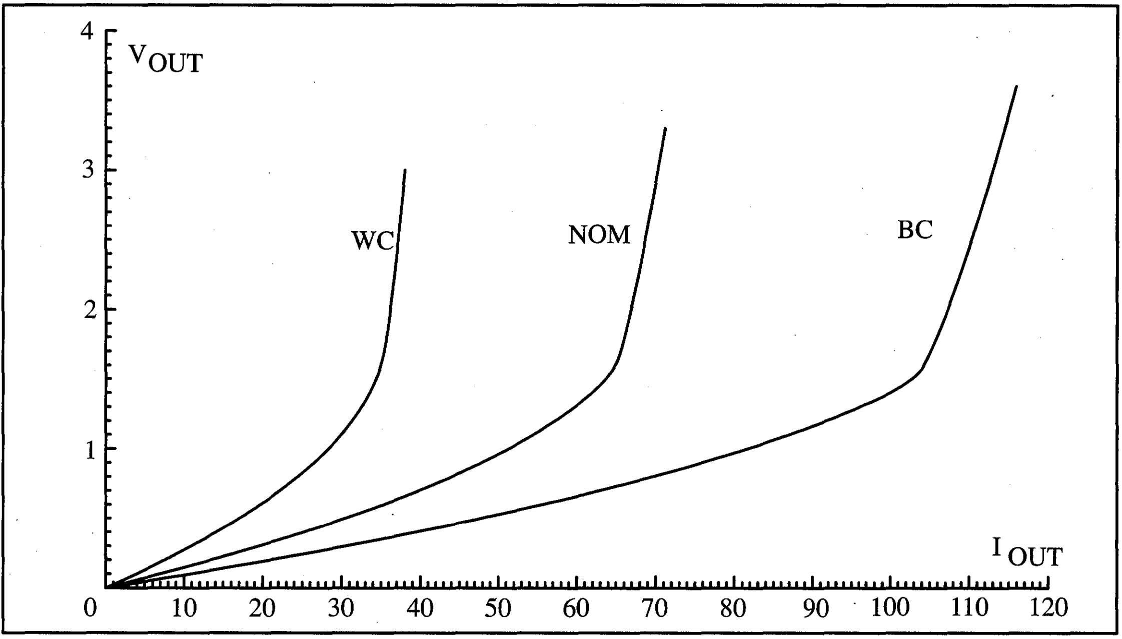


Figure 6–15. Pull Down Curves, TTL Driver, I_{OL} = 8mA, P/L = A

6.7.2.10 Pull Down Curves, TTL Driver, I_{OL} = 8mA, P/L = B

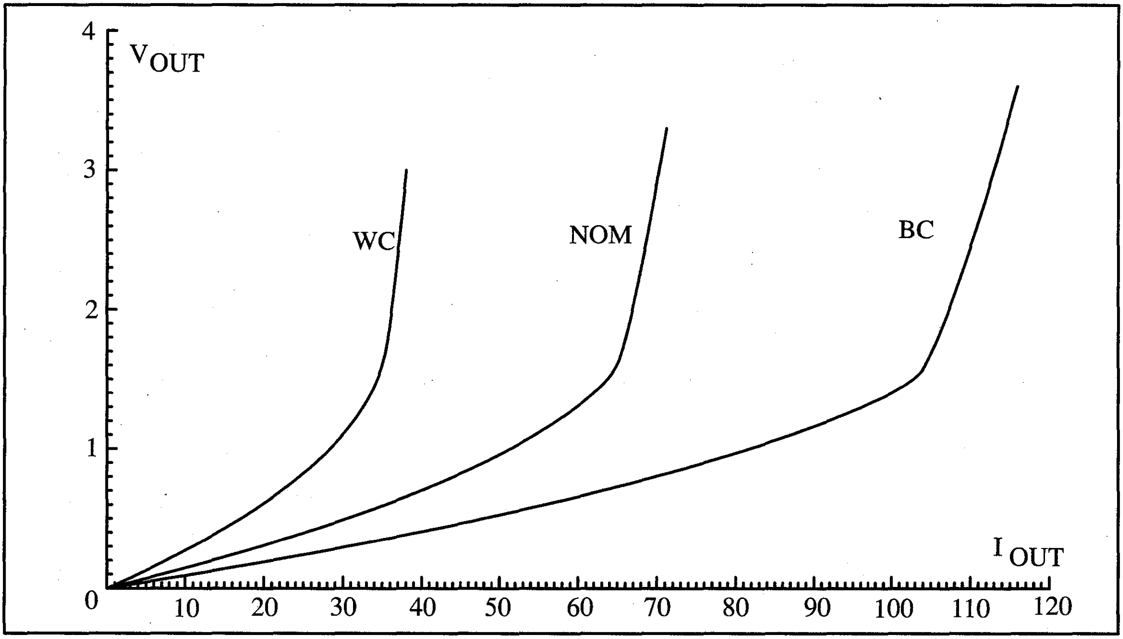


Figure 6–16. Pull Down Curves, TTL Driver, I_{OL} = 8mA, P/L = B

6.7.2.11 Pull Down Curves, TTL Driver, I_{OL} = 8mA, P/L = C

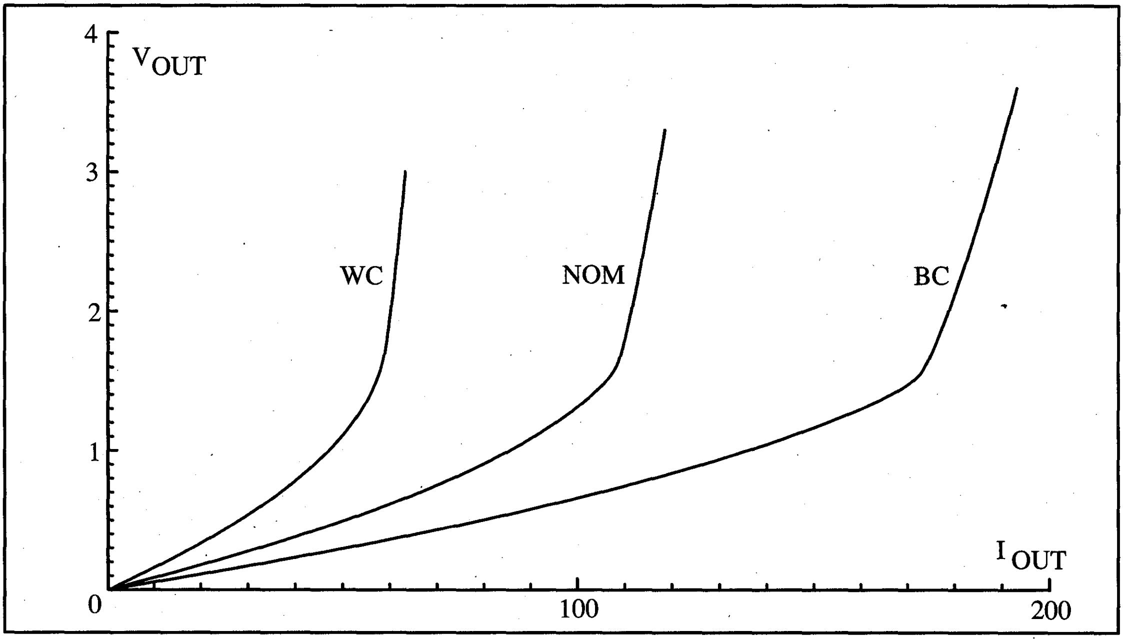


Figure 6–17. Pull Down Curves, TTL Driver, I_{OL} = 8mA, P/L = C

6.7.2.12 Pull Up Curves, TTL Driver, I_{OL} = 8mA, P/L = A, B, and C

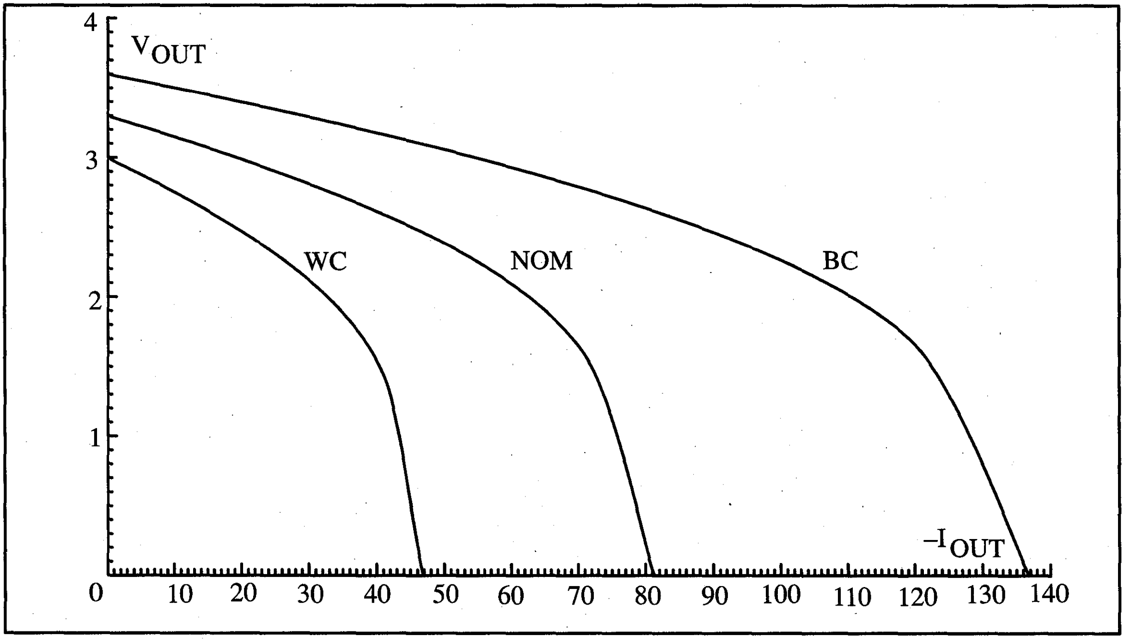


Figure 6–18. Pull Up Curves, TTL Driver, I_{OL} = 8mA, P/L = A, B, and C

6.7.2.13 Pull Down Curves, TTL Driver, I_{OL} = 12mA, P/L = A

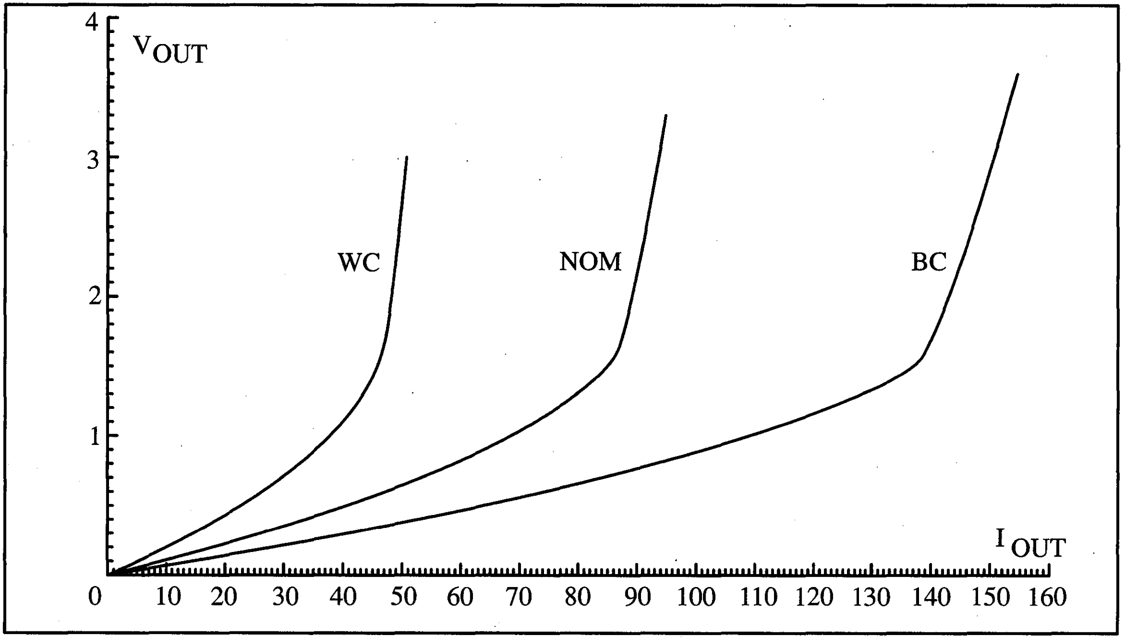


Figure 6–19. Pull Down Curves, TTL Driver, I_{OL} = 12mA, P/L = A

6.7.2.14 Pull Down Curves, TTL Driver, I_{OL} = 12mA, P/L = B

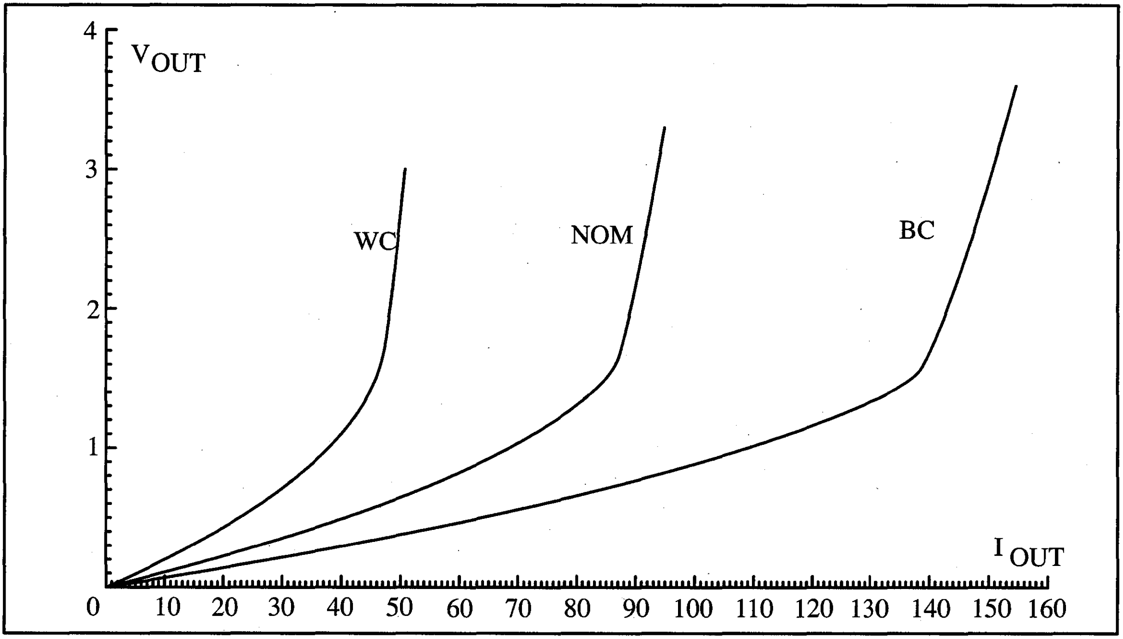


Figure 6–20. Pull Down Curves, TTL Driver, I_{OL} = 12mA, P/L = B

6.7.2.15 Pull Down Curves, TTL Driver, I_{OL} = 12mA, P/L = C

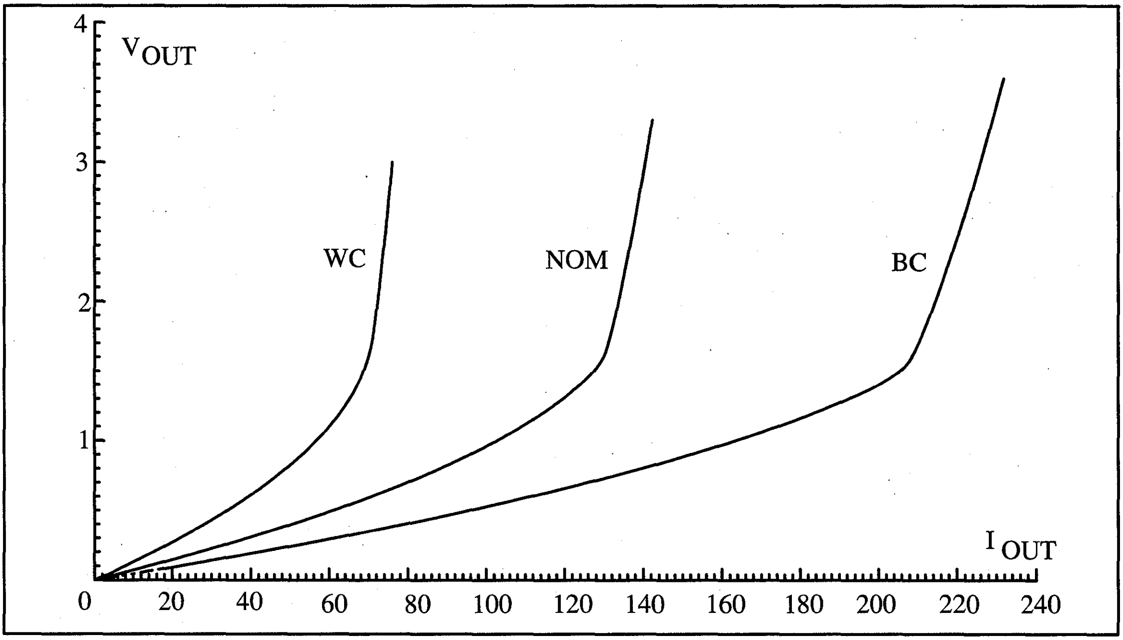


Figure 6–21. Pull Down Curves, TTL Driver, I_{OL} = 12mA, P/L = C

6.7.2.16 Pull Up Curves, TTL Driver, I_{OL} = 12mA, P/L = A, B, and C

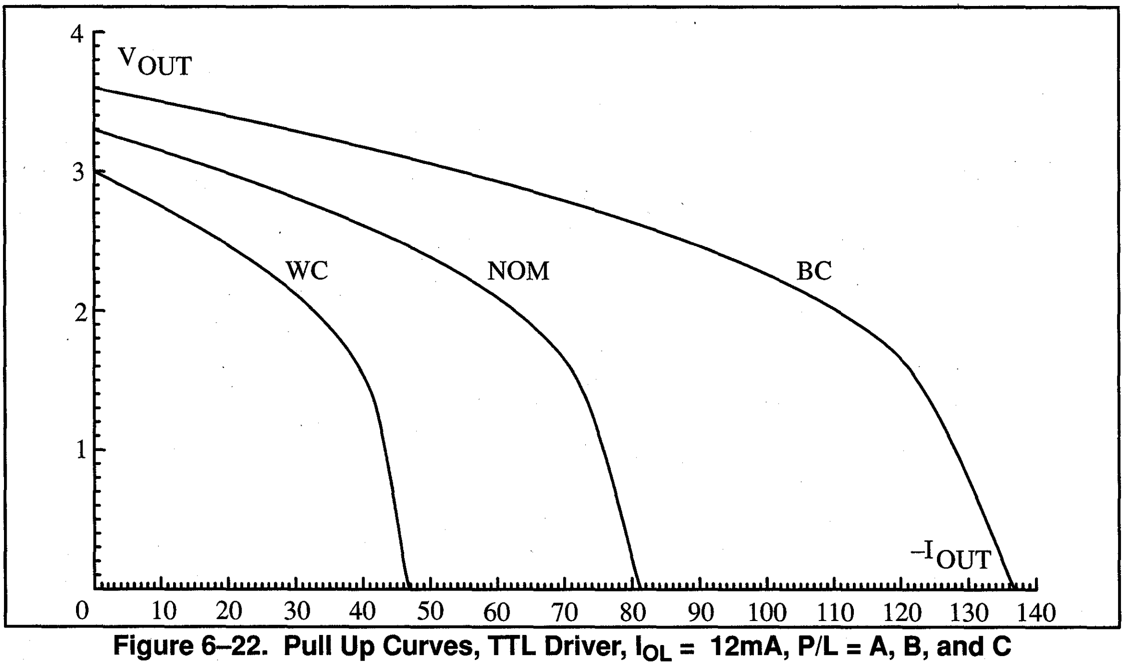


Figure 6–22. Pull Up Curves, TTL Driver, I_{OL} = 12mA, P/L = A, B, and C

6.7.2.17 Pull Up Curves, TTL Driver, I_{OL} = 24mA, P/L = A

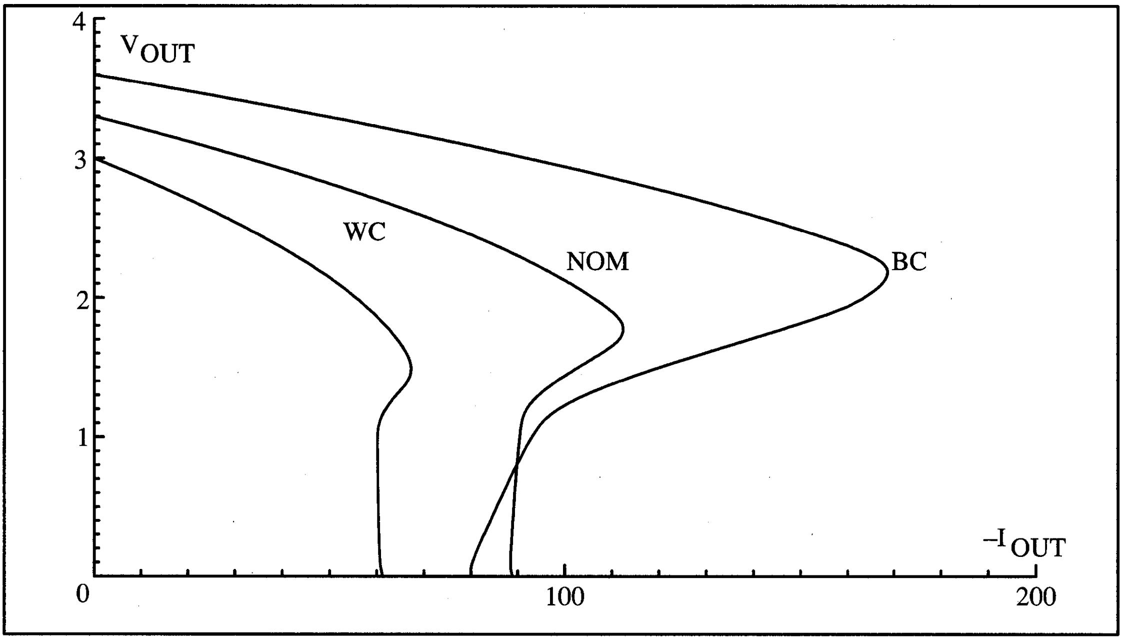


Figure 6–23. Pull Up Curves, TTL Driver, I_{OL} = 24mA, P/L = A

6.7.2.18 Pull Up Curves, TTL Driver, I_{OL} = 24mA, P/L = B

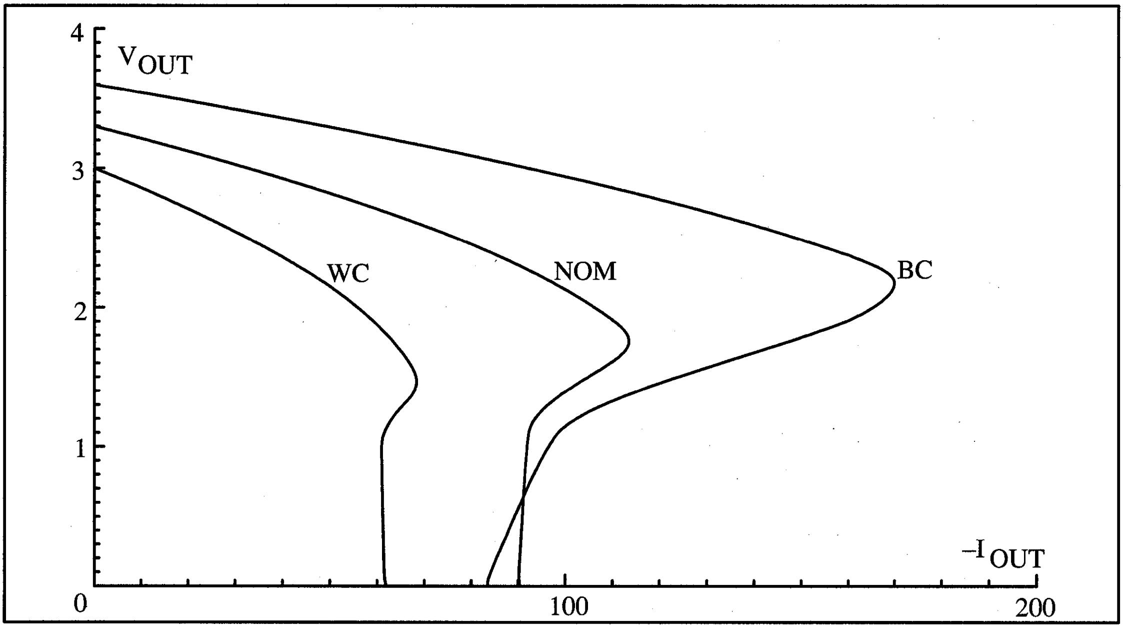


Figure 6–24. Pull Up Curves, TTL Driver, I_{OL} = 24mA, P/L = B

6.7.2.19 Pull Up Curves, TTL Driver, I_{OL} = 24mA, P/L = C

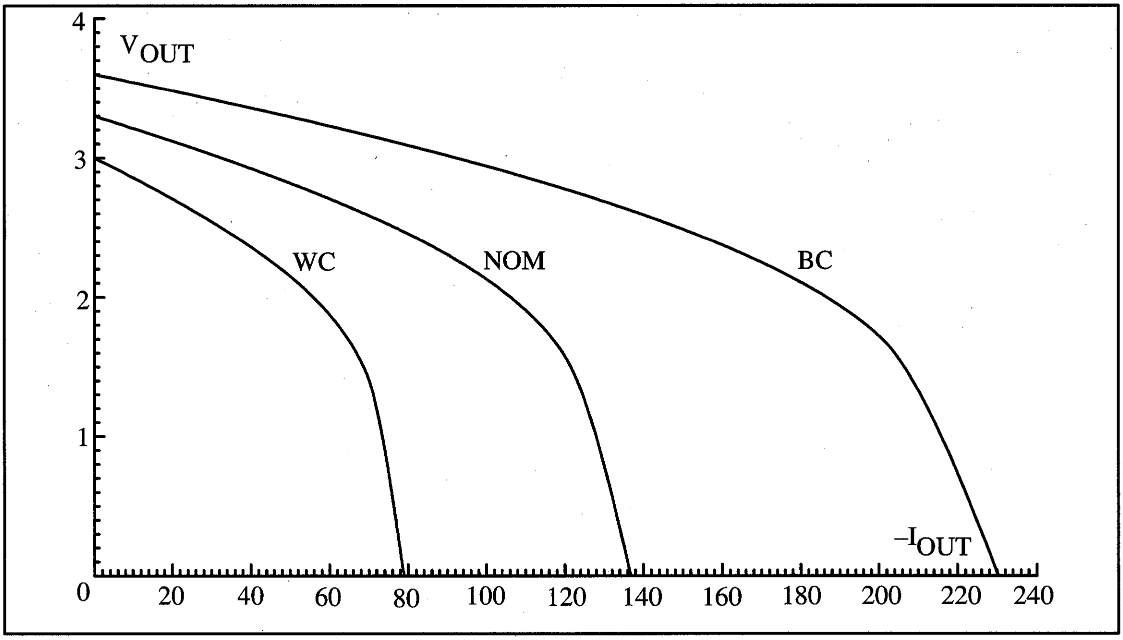


Figure 6–25. Pull Up Curves, TTL Driver, I_{OL} = 24mA, P/L = C

6.7.2.20 Pull Down Curves, TTL Driver, I_{OL} = 24mA, P/L = A, B, and C

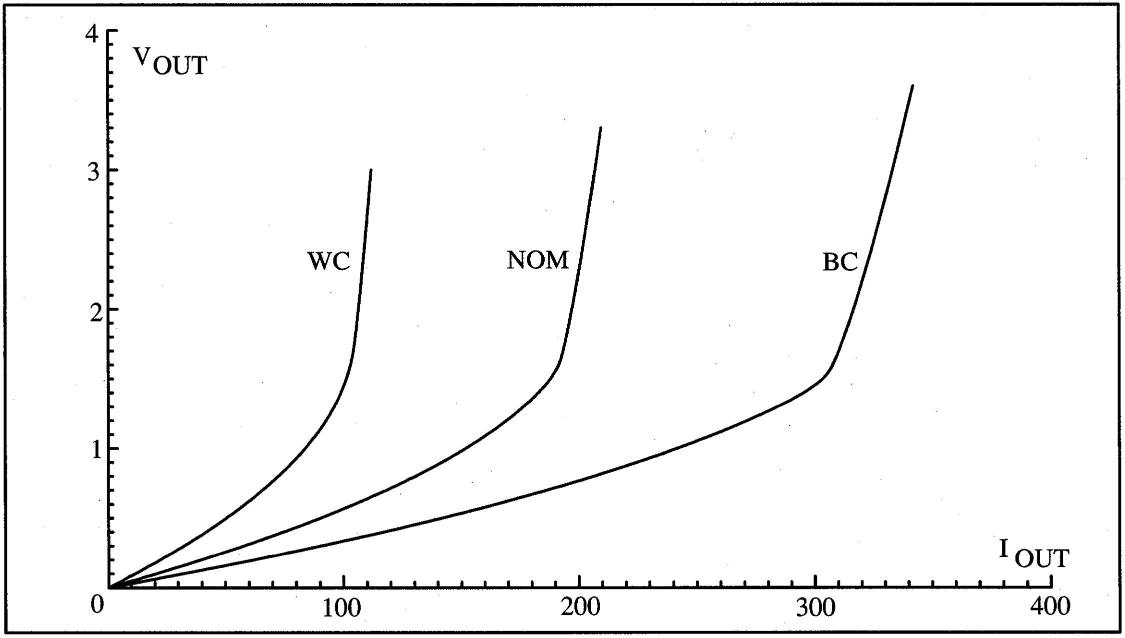


Figure 6–26. Pull Down Curves, TTL Driver, I_{OL} = 24mA, P/L = A, B, and C

Section 7
Timings

Unless otherwise indicated, all specifications in this section apply equally to the 653 Buffer and the 654 Controller.

7.1 Timing Conventions

7.1.1 Board Delays

Unless otherwise indicated, all timing specifications refer to events at the pins of the chip under discussion. In systems operating at speeds typical of the 60X family, propagation delays from point to point on a circuit board can be significant. The timing diagrams make no assumptions about board delays. No board or system propagation delays have been included in the timing diagrams or in the timing charts. The timing diagrams assume that there is zero propagation delay from pins on the 654 to pins on the 653, and to pins on the DRAMs, and to pins on the 60X, and to the PCI bus. For example, the delay from BURST_CLK# fall (at the 654 Controller pin) to MEM_ADDR valid (at the 653 Buffer pin) is shown as *tmab,* and this time is called out in the timing tables. However, *tmab* does not include the time required for BURST_CLK# to travel from the 654 Controller to the 653 Buffer. Likewise, the delay imposed upon TBST# between the 60X CPU and the 654 Controller is neither specified nor included in any of the timing information presented. Allow for delays between components while constructing timing diagrams for the design of an actual system.

7.1.2 Terms and Definitions

7.1.2.1 Signal Range Names

Signal range names used without range indicators refer to the entire group of signals. For example, CPU_DATA refers to the 653 signals CPU_DATA[0:63]. Ranges are expressed as [most-significant bit : least-significant bit].

7.1.2.2 Signal Group Names

Some signals are referred to in a group in the timing diagrams. For example, CPU_ADDR refers to 60X address and address transfer attribute signals. Particular signals in the group may be shown separately for emphasis (TBST#, for example).

7.1.2.3 Timing Diagram and Timing Chart Definitions

Table 7–1 shows the terms that are used in this section to describe signals.

Table 7–1. Timing Diagram and Timing Chart Definitions

Term	Definition
in or **I**	Input only pin
out or **O**	Output only pin. Output driver is totem-pole unless otherwise noted.
I/O	Input/output pin, tri-state capable unless otherwise noted.
CLK	The rising edge of CPU_CLK.
asserted/active	In the logic TRUE state.
deasserted/inactive/ negated	In the logic FALSE state.
valid	The voltage of the signal is above V_H or below V_L. Valid does not imply that the signal is TRUE or asserted or active.

7.1.3 Transaction Clock Cycle Nomenclature

Following the 601 convention, CPU_CLK cycles are labeled according to the cycle number. A rising edge of CPU_CLK is referred to by using the numbers of the cycles on either side of it. The rising edge labeled A in Figure 7–1 is called the 1/2 rising edge of CPU_CLK.

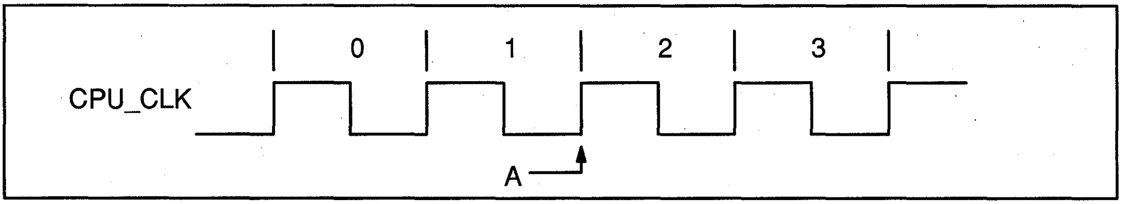


Figure 7–1. CPU_CLK Cycle Nomenclature

Figure 7–2 shows the nomenclature used in the PCI Specification to refer to the rising edges of PCI_CLK. During a defined PCI transaction, each rising edge of the PCI clock is numbered (PCI_FRAME# is asserted on PCI_CLK rising edge 1, which is also called PCI_CLK 1).

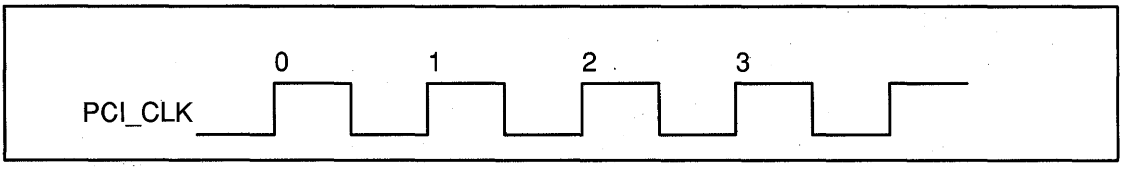


Figure 7–2. PCI_CLK Cycle Nomenclature

7.1.4 Signal Switching Levels for Timing Analysis

Figure 7–3 shows typical timing analysis signal switching levels, where V_H and V_L (see *valid* in Table 7–2) are the valid logic levels used for all input and output signals except CPU_CLK. Unless otherwise indicated, all input and output signal (not clock) switching specifications refer to the point in time at which the signal crosses one of these levels. These levels are used for timing analysis only, and do not imply anything about the DC characteristics of the device.

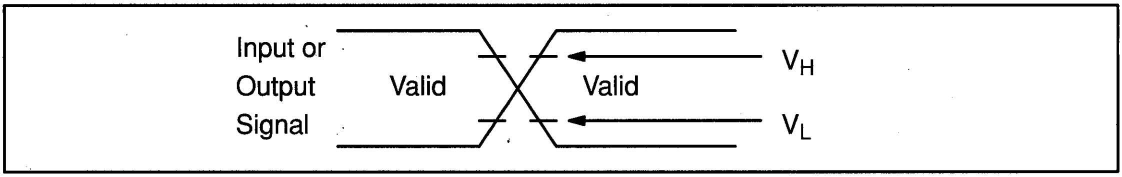


Figure 7–3. Switching Levels

Table 7–2. Valid Logic High and Low Levels for 650 Bridge Timing Specifications

Level Name	Symbol	Voltage
Logic High Level	V_H	2v
Logic Low Level	V_L	.8v
Midpoint Voltage	VM	1.5v

7.1.5 Input Setup Time

Input setup time is the amount of time that an input signal is required to be stable at a valid logic level immediately prior to an event. Input setup time (T_{IS} in Figure 7–4) from a signal to the clock is measured from the point in time at which the input becomes valid to the the point in time at which the clock rising edge crosses the VM level. Input setup time from a signal to an input strobe is measured from the point in time at which the input becomes valid to the the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active-going direction).

7.1.6 Input Hold Time

Input hold time is the amount of time that an input signal is required to remain stable at a valid logic level immediately following an event. Input hold time (T_{IH} in Figure 7–4) from the clock to an input signal is measured from the point in time at which the clock rising edge crosses the VM level to the point in time at which the input goes invalid (crosses the valid logic level in the invalid-going direction). Input hold time from an input strobe to an input signal is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active-going direction) to the point in time at which the input goes invalid.

7.1.7 Output Hold Time

Output hold time is the amount of time that an output signal remains stable at a valid logic level immediately following an event which may cause the output to change state. Output hold time (T_{OH} in Figure 7–4) from the clock is measured from the point in time at which the rising edge of the clock crosses the VM level to the point in time at which the output signal becomes invalid (crosses the valid logic level in the invalid-going direction). Output hold time from an input strobe is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active-going direction) to the point in time at which the output signal becomes invalid.

7.1.8 Output Valid Delay Times

Output valid delay time is the amount of time required for an output signal to change to a stable valid state following an event. Output valid delay time (T_{OD} in Figure 7–4) from the clock is measured from the point in time at which the rising edge of the clock crosses the VM level to the point in time at which the output signal becomes valid (crosses the valid logic level in the valid-going direction). Output valid delay time from an input strobe is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active-going direction) to the point in time at which the output signal becomes valid.

7.1.9 Output Tri-State Hold Time

Output tri-state hold time is the amount of time that an output signal remains driven to a valid logic level immediately following an event which may cause the output to tri-state (go to a high–impedance state). Output tri-state hold time (T_{3SH} in Figure 7–4) from the clock is measured from the point in time at which the rising edge of the clock crosses the VM level to the point in time at which the output signal becomes invalid (is no longer guaranteed to be actively driven to a valid logic level). Output hold time from an input strobe is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active-going direction) to the point in time at which the output signal becomes invalid. Note that this specification deals with the time that the output driver remains active following an event which may turn it off. The actual output signal may remain valid for some time after this, depending on other conditions.

7.1.10 Output Tri-State Delay Time

Output valid delay time is the amount of time required for an output signal driver to turn off (go to a high impedance state) following an event. Output valid delay time (T_{3SD} in Figure 7–4) from the clock is measured from the point in time at which the rising edge of the clock crosses the VM level to the point in time at which the output signal driver turns off (is no longer driving the output). Output valid delay time from an input strobe is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active-going direction) to the point in time at which the output signal driver turns off. Note that this specification deals with the time that it takes the output driver to stop driving the output signal line following an event which may turn it off. The actual output signal may remain valid for some time after this, depending on other conditions.

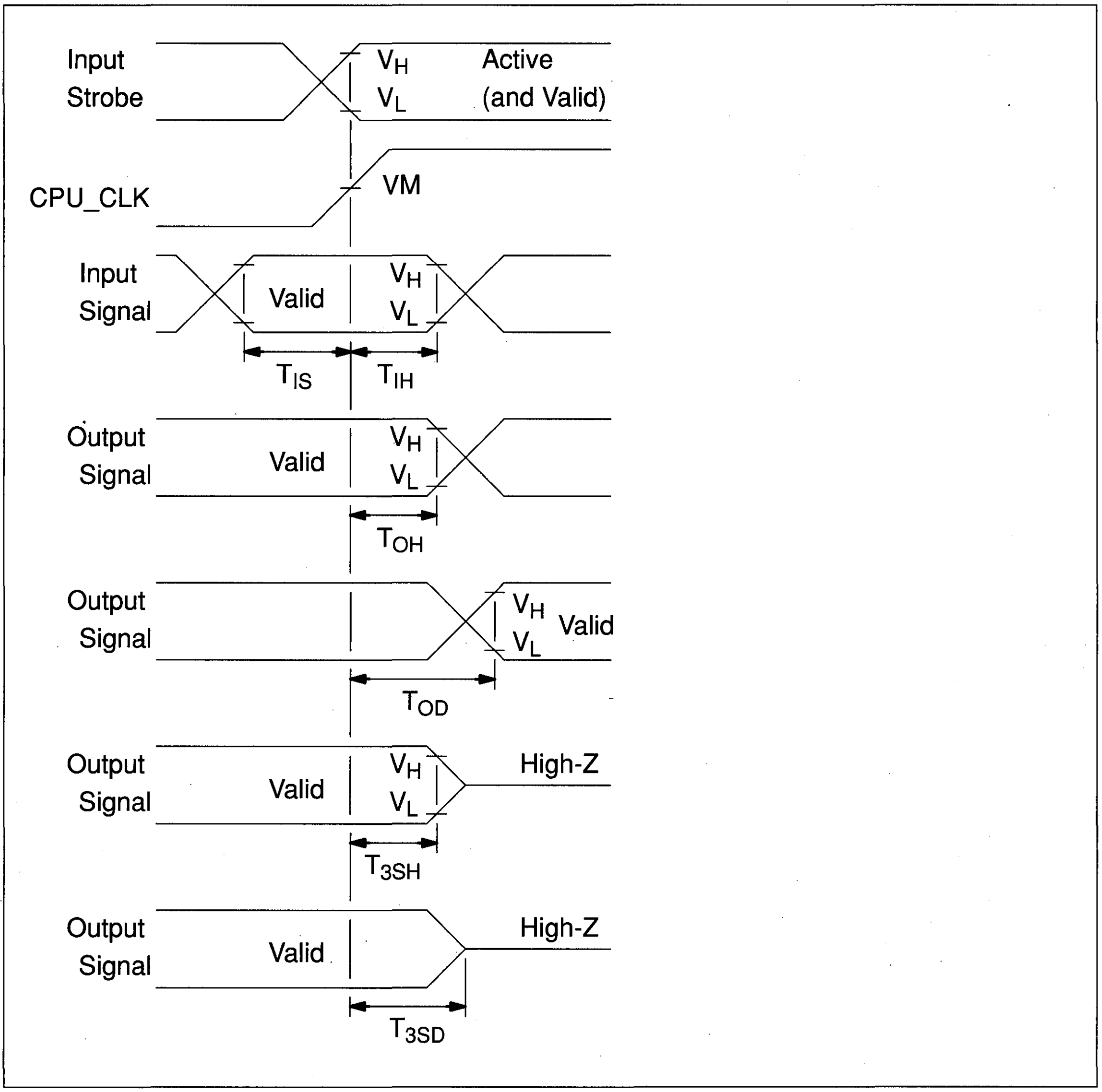


Figure 7–4. Signal Timing Conventions

7.2 Clock Considerations

To maintain synchronization between the 654 Controller and the 601 CPU, certain constraints are placed on the relationship between 2X_PCLK and CPU_CLK. There are also constraints placed on the relationship between PCI_CLK and CPU_CLK.

7.2.1 Clock Switching Levels

Unless otherwise indicated, all references to CPU_CLK clock timing refer to the point in time at which the CPU_CLK crosses the VM level. See Figure 7–4, Table 7–2, and Figure 7–5.

7.2.2 The CPU_CLK

The CPU_CLK is shown (at the CPU_CLK pin of the 654 Controller) in Figure 7–5, where T_{CH} is the time that CPU_CLK is high, and T_{CL} is the time that CPU_CLK is low. The duty cycle of CPU_CLK is shown in Table 7–3 as $T_{D(CPU\_CLK)}$. If the period of CPU_CLK is 15ns, then T_{CH} may range from 5.25ns to 9.75ns.

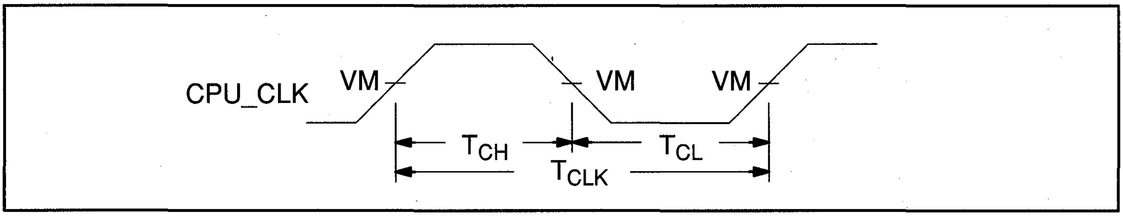


Figure 7–5. CPU_CLK Timing

In general, 654 Controller inputs are sampled on the rising edge of CPU_CLK, and outputs are updated on the rising edge of CPU_CLK.

Table 7–3. CPU_CLK Timing Constraints

Symbol	Description	Min	Max	Units	Note
$T_{D(CPU\_CLK)}$	CPU_CLK Duty cycle $T_{CH}/(T_{CH} + T_{CL})$	35	65	%	(1)
$T_{S(2X—CPU)}$	Allowed skew of 2X_PCLK wrt CPU_CLK	–1	+1	ns	(2)

Notes for Table 7–3.

1) $10°C \leq T_J \leq 85°C$, $3.0v \leq V_{DD} \leq 3.8v$. See Figure 7–5.

2) $10°C \leq T_J \leq 85°C$, $3.0v \leq V_{DD} \leq 3.8v$. See Figure 7–6.

7.2.3 The 654 Controller Clock and the 601 Clocks

Nominally, each CPU_CLK rising edge is exactly aligned with a rising edge of 2X_PCLK. The falling edge of CPU_CLK is not defined with respect to 2X_PCLK. It is defined by the duty cycle constraint. Figure 7–6 shows the required relationship between the 2X_PCLK (a 601 signal) at the pin of the CPU, and CPU_CLK at the pin of the 654. Note that the allowed skew $T_{S(2X—CPU)}$, is shown as ±1ns in Table 7–3.

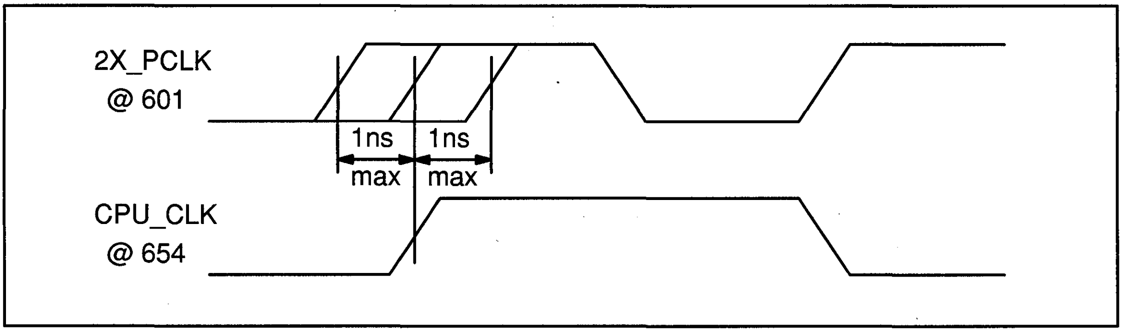


Figure 7–6. CPU_CLK Timing

When the CPU_CLK is running at 66MHz, as shown in Figure 7–7, the 601 BCLK_EN# signal must be tied low, and CPU_CLK must be in phase with the 601 PCLK_EN# signal. As shown in Figure 7–8, when CPU_CLK is running at 33MHz, BCLK_EN# runs at 66MHz, and the phase relationships between all three clocks must be maintained. In each case, CPU_CLK is required to up-transition on the 2X_PCLK up-transition indicated by the arrow. This is the 2X_PCLK edge on which the 601 samples inputs and issues outputs. Enforcing these constraints synchronizes the 654 Controller to the 601 CPU.

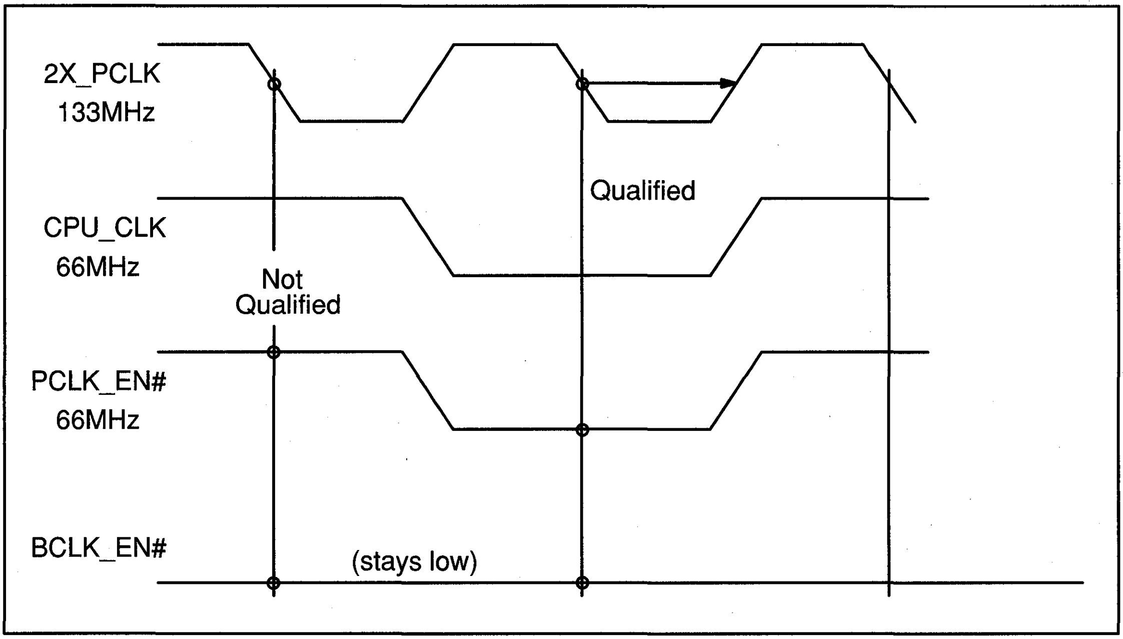


Figure 7–7. CPU_CLK Phase Relationships at 66MHz

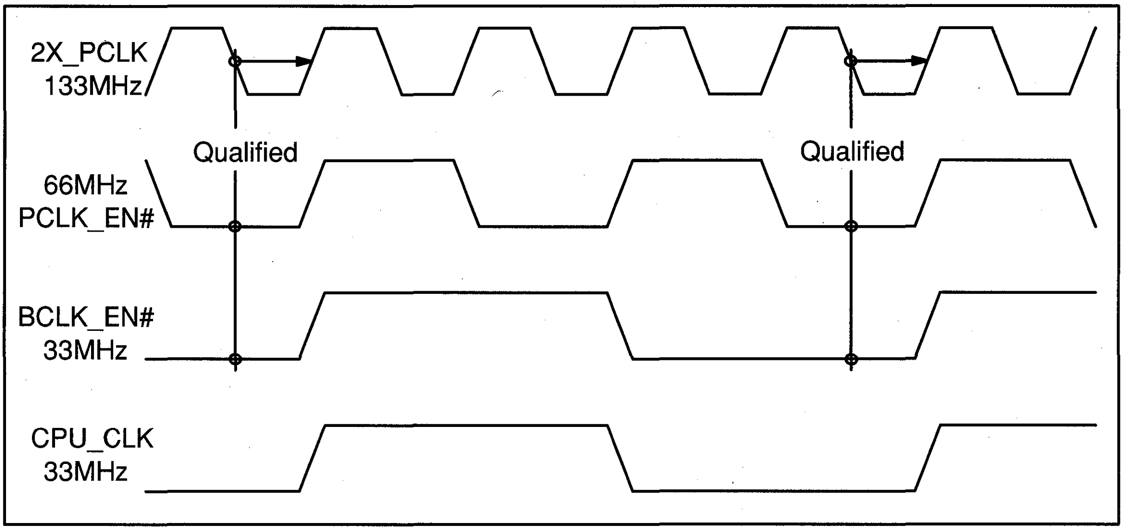


Figure 7–8. CPU_CLK Phase Relationships at 33MHz

7.2.4 CPU_CLK to PCI_CLK Skew

The 654 Controller is clocked by the CPU_CLK signal. The 654 also receives a PCI_CLK signal, which it treats as a signal input as opposed to a clock. These signals are typically generated in two different chips and travel two different paths to the 654. This implementation typically generates some skew between the two signals. This skew must not exceed Tcpcs, the allowed CPU_CLK to PCI_CLK skew specification shown in Table 7–4. Tcpcs is independent of CPU_CLK speed and is measured at the pins of the 654 Controller.

Table 7–4. PCI_CLK Timing Constraints

Symbol	Description	Min	Max	Units	Note
T_{CPCS} @ 3.6v	Allowed skew of PCI_CLK wrt CPU_CLK	–.5	4	ns	(1)
T_{CPCS} @ 3.3v	Allowed skew of PCI_CLK wrt CPU_CLK	–.5	3.3	ns	(2)

Notes for Table 7–4.
1) 10°C $\leq T_J \leq$ 85°C, 3.4v $\leq V_{DD} \leq$ 3.8v. See Figure 7–9 and Figure 7–10.
2) 10°C $\leq T_J \leq$ 85°C, 3.0v $\leq V_{DD} \leq$ 3.6v. See Figure 7–9 and Figure 7–10.

7.2.4.1 Clocking In 2:1 Mode

In 2:1 clocking mode (see Figure 7–9) the CPU_CLK is running at twice the speed of the PCI_CLK. (The time scale shown is for reference only, for a system running with a 66MHz CPU_CLK and a 33MHz PCI_CLK.) PCI_CLK is required to change state on the rising edge of CPU_CLK. Tcpcs applies to both the rising and the falling edges of PCI_CLK.

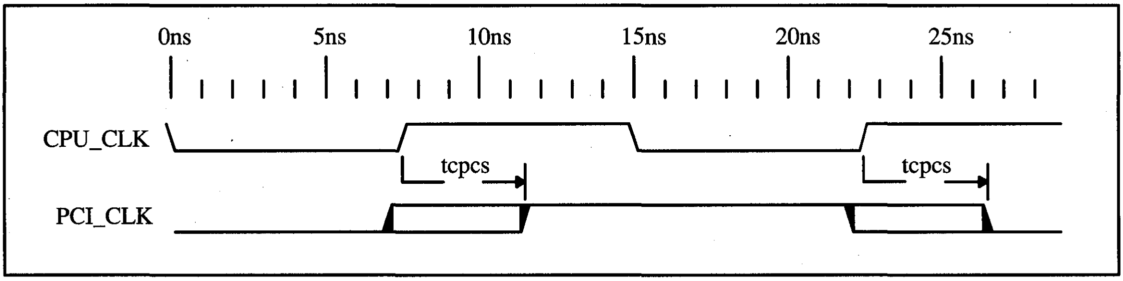


Figure 7–9. Timing Diagram, CPU_CLK to PCI_CLK Skew, 2:1 Mode

7.2.4.2 Clocking In 1:1 Mode

In 1:1 clocking mode (see Figure 7–10), the CPU_CLK is running at the same speed as the PCI_CLK. (The time scale shown is for reference only, for a system running with a 33MHz CPU_CLK and a 33MHz PCI_CLK.) PCI_CLK is required to be in phase with CPU_CLK and to change state when CPU_CLK changes state. Tcpcs applies to both the rising and the falling edges of PCI_CLK.

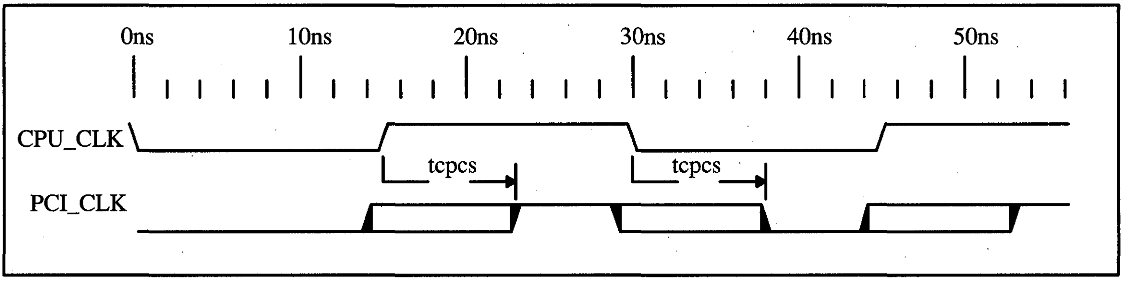


Figure 7–10. Timing Diagram, CPU_CLK to PCI_CLK Skew, 1:1 Mode

7.3 Power-On Considerations

The 650 Bridge is designed to impose no additional power-on-reset or power supply behavior constraints on a system that contains a 60X CPU and a PCI bus. The 650 works properly in a system designed to correctly support the 60X CPU and the PCI bus.

The 654 Controller requires RESET# to be asserted at power-on for a minimum of 1 us past power-good, and for a minimum of 10 CPU_CLK cycles past the point in time at which CPU_CLK is stable and within specification. The inputs to the 654 Controller are not required to be in any special state during the reset period, but the 654 will start to respond to control inputs immediately following the deassertion of RESET#. This design fully supports a properly functioning 60X CPU, L2 cache, and PCI agents.

The 653 Buffer requires no reset signal, as it is controlled by the 654 Controller. When used without the 654 Controller, the only requirement is for the designer to arrange for the CPU_ADDR_SEL# signal to be asserted and then deasserted during the power-on reset sequence.

7.4 654 Controller Timing

7.4.1 654 Controller Synchronous Input Timing Characteristics

Table 7–5. 654 Controller Input Timing Characteristics By Signal

654 Input Signal	I/O	3.3v V_{DD} System (2)		3.6v V_{DD} System (1)		Note
		Setup Time (ns) Min (3)	Hold Time (ns) Min (4)	Setup Time (ns) Min (3)	Hold Time (ns) Min (4)	
Processor Signals						
AACK#	I/O	.20	2.0	0.2	2.0	
ARTRY#	I/O	2.9	2.0	0.25	2.0	
CPU_ADDR[0]	I	0.6	2.0	1.1	2.0	
CPU_ADDR[1]	I	6.9	2.0	8.5	2.0	
CPU_ADDR[2]	I	6.7	2.0	8.2	2.0	
CPU_ADDR[3]	I	6.2	2.0	7.7	2.0	
CPU_ADDR[4]	I	6.6	2.0	7.7	2.0	
CPU_ADDR[5]	I	6.2	2.0	7.7	2.0	
CPU_ADDR[6]	I	6.6	2.0	8.1	2.0	
CPU_ADDR[7]	I	6.6	2.0	8.1	2.0	
CPU_ADDR[8]	I	6.6	2.0	8.1	2.0	
CPU_ADDR[19]	I	3.4	.20	3.9	.20	(5)
CPU_ADDR[29]	I	6.2	2.0	7.7	2.0	
CPU_ADDR[30]	I	6.8	2.0	8.1	2.0	
CPU_ADDR[31]	I	7.3	2.0	8.8	2.0	
CPU_REQ#	I	1.7	2.0	1.3	2.0	
DPE#	I	1.3	2.0	1.0	2.0	
MASK_TEA#	I	1.3	2.0	1.3	2.0	
TA#	I/O	0	2.0	0	2.0	
TBST#	I	6.8	2.0	5.9	2.0	
TS#	I/O	1.5	2.0	1.0	2.0	
TSIZ[0]	I/O	7.4	2.0	8.8	2.0	
TSIZ[1]	I/O	7.1	2.0	8.4	2.0	
TSIZ[2]	I/O	7.2	2.0	8.8	2.0	
TT[0]	I/O	0.9	2.0	1.4	2.0	
TT[1]	I/O	5.0	2.0	6.1	2.0	

Table 7–5. 654 Controller Input Timing Characteristics By Signal (Continued)

654 Input Signal	I/O	3.3v V_{DD} System (2)		3.6v V_{DD} System (1)		Note
		Setup Time (ns) Min (3)	Hold Time (ns) Min (4)	Setup Time (ns) Min (3)	Hold Time (ns) Min (4)	
TT[2]	I/O	0.9	2.0	1.3	2.0	
TT[3]	I/O	6.4	2.0	8.0	2.0	
XATS#	I	1.6	2.0	1.0	2.0	
L2 Cache Signals						
L2_CACHE_REQ#	I	2.7	2.0	2.3	2.0	
L2_CLAIM#	I	2.5	2.0	2.2	2.0	
L2_PRESENT#	I	3.6	2.0	3.0	2.0	
PCI Sideband Signals (Incident Wave)						
PCI_CLK	I	See Section 7.2.4.		See Section 7.2.4.		
PCI_REQ[1:5]#	I	7.0	0	7.0	0	
PCI Bus Signals (Reflected Wave)						
PCI_C/BE[3:0]#	I/O	7.0	0	7.0	0	
PCI_DEVSEL#	I/O	8.0	0	7.0	0	
PCI_FRAME#	I/O	7.0	0	7.0	0	
PCI_IRDY#	I/O	7.0	0	7.0	0	
PCI_STOP#	I/O	7.0	0	7.0	0	
PCI_TRDY#	I/O	7.0	0	7.0	0	
I/O Bus and I/O Bridge Signals						
IO_BRDG_HOLD#	I	0	2.0	0	2.0	
IO_BRDG_IRQ	I	0.5	2.0	0.5	2.0	
IO_BRDG_REQ#	I	7.0	0	7.0	0	
ISA_MASTER#	I	0	2.0	0	2.0	
NMI_IRQ	I	7.9	2.0	6.4	2.0	
653 Buffer Signals						
MEM_PAGE_HIT#	I	3.1	2.0	2.3	2.0	
MEM_PAR_GOOD	I	0.8	2.0	0.7	2.0	
PCI_AD_PAR	I	0	2.0	0	2.0	

Table 7–5. 654 Controller Input Timing Characteristics By Signal (Continued)

654 Input Signal	I/O	3.3v V_{DD} System (2)		3.6v V_{DD} System (1)		Note
		Setup Time (ns) Min (3)	Hold Time (ns) Min (4)	Setup Time (ns) Min (3)	Hold Time (ns) Min (4)	
System Interface and Miscellaneous Signals						
LE_MODE_REQ#	I	0.8	2.0	0.3	2.0	
REFRESH_REQ#	I	0	2.0	0	2.0	
SRESET_REQ#	I	0	2.0	0	2.0	

Notes for Table 7–5.

1) 10°C $\leq T_J \leq$ 85°C, 3.4v $\leq V_{DD} \leq$ 3.8v.
2) 10°C $\leq T_J \leq$ 85°C, 3.0v $\leq V_{DD} \leq$ 3.6v.
3) From signal valid to CPU_CLK rise.
4) From CPU_CLK rise to signal invalid.
5) Fast logic path chosen to accomodate delays imposed on A19 during PCI bus master transactions that are broadcast (A19 goes through the 653 Buffer) to the CPU bus for snooping. A19 is used to detect page misses during PCI to memory burst transactions.
6) TEST#, RI#, and DI# timing is not specified.

7.4.2 654 Controller Synchronous Output Timing Characteristics

Table 7–6. 654 Controller Output Timing Characteristics By Signal

654 Output Signal	I/O	3.3v V_{DD} System (2)			3.6v V_{DD} System (1)		
		Output Hold Time (3) (ns) Min	Output Tri-state Delay (4) (ns) Max	Output Valid Delay (5) (ns) Max	Output Hold Time (3) (ns) Min	Output Tri-state Delay (4) (ns) Max	Output Valid Delay (5) (ns) Max
Processor Signals							
AACK#	I/O	2.2	10.8	10.5	2.1	10.3	9.2
ARTRY#	I/O	2.4	10.4	10.4	2.3	9.3	9.3
CPU_GNT#	O	2.5	NA	9.7	2.4	NA	8.7
INT_CPU#	O	3.3	NA	15.1	3.0	NA	12.7
SRESET_CPU#	O	3.3	NA	14.9	3.0	NA	12.6
TA#	I/O	2.2	10.7	10.1	2.0	10.2	9.1
TBST#	I/O	3.6	15.6	15.6	3.3	14.5	14.5
TEA#	O	2.5	NA	9.6	2.3	NA	8.6
TS#	I/O	2.2	12.2	8.6	2.0	10.9	7.8
TSIZ[0:2]	I/O	3.6	15.6	15.6	3.3	14.5	14.5
TT[0]	I/O	3.6	15.7	15.7	3.3	14.5	14.4
TT[1]	I/O	2.8	14.9	13.4	2.4	13.2	11.9
TT[2]	I/O	3.8	15.7	15.7	3.4	14.5	14.4
TT[3]	I/O	3.6	15.7	15.7	3.3	14.5	14.4
L2 Cache Signals							
L2_CACHE_GNT#	O	2.8	NA	14.0	2.4	NA	12.7
PCI Sideband Signals (Incident Wave)							
PCI_GNT[1:5]#	O	2.5	NA	13.0	2.0	NA	12.0
PCI Bus Signals (Reflected Wave)							
PCI_C/BE[3:0]#	I/O	2.0	13.0	12.1	2.0	11.5	11.0
PCI_DEVSEL#	I/O	2.0	11.3	11.8	2.0	10.1	11.0
PCI_FRAME#	I/O	2.0	12.8	11.5	2.0	11.4	11.0
PCI_IRDY#	I/O	2.0	26.7	11.5	2.0	23.1	11.0
PCI_PAR	O	2.0	13.0	12.3	2.0	11.5	11.0
PCI_STOP#	I/O	2.0	11.2	12.2	2.0	10.0	11.0
PCI_TRDY#	I/O	2.0	11.3	12.0	2.0	10.0	11.0

Table 7–6. 654 Controller Output Timing Characteristics By Signal (Continued)

654 Output Signal	I/O	3.3v V_{DD} System (2)			3.6v V_{DD} System (1)		
		Output Hold Time (3) (ns) Min	Output Tri-state Delay (4) (ns) Max	Output Valid Delay (5) (ns) Max	Output Hold Time (3) (ns) Min	Output Tri-state Delay (4) (ns) Max	Output Valid Delay (5) (ns) Max
I/O Bus and I/O Bridge Signals							
IO_BRDG_GNT#	O	2.5	NA	12.7	2.0	NA	12.0
DRAM Memory Subsystem Signals							
BE_PAR_EN#	O	3.5	NA	15.7	3.1	NA	13.9
CAS[7:0]#	I/O	2.4	NA	12.3	2.3	NA	10.8
LE_PAR_EN#	O	3.5	NA	15.7	3.1	NA	13.9
RAS[7:0]#	O	2.5	NA	12.4	2.3	NA	10.9
WE[1:0]#	O	2.9	NA	17.2	2.6	NA	15.0
Boot ROM Device Signals							
ROM_CS#	O	3.7	NA	16.5	3.4	NA	13.9
ROM_OE#	O	3.4	NA	15.7	3.1	NA	13.2
ROM_WE#	O	3.4	NA	15.6	3.1	NA	13.5
653 Buffer Signals							
ADDRHI/DATALO	O	3.2	NA	16.0	2.8	NA	14.3
ALL_ONES_SEL#	O	3.0	NA	15.0	2.7	NA	13.5
BURST_CLK#	O	3.0	NA	14.8	2.6	NA	13.3
CPU_ADDR_OE#	O	3.0	NA	14.6	2.6	NA	13.2
CPU_ADDR_SEL#	O	3.1	NA	15.0	2.7	NA	13.5
CPU_DATA_OE#	O	2.9	NA	15.0	2.5	NA	13.5
CPU_DATA_SEL#	O	3.0	NA	15.0	2.6	NA	13.5
DPE_ERR#	O	3.4	NA	15.4	3.1	NA	13.0
ERR_ADDR_SEL#	O	3.7	NA	16.1	3.3	NA	13.8
L_PCI_DATA#	O	2.8	NA	13.9	2.4	NA	12.5
LE_MODE_SEL#	O	3.4	NA	15.4	3.1	NA	13.0
MEM_DATA_OE#	O	3.2	NA	18.2	2.8	NA	16.2
MEM_DATA_SEL#	O	3.0	NA	18.8	2.6	NA	16.8
MEM_PAR_ERR#	O	3.6	NA	15.5	3.2	NA	13.7
NO_TRANS	O	3.0	NA	15.0	2.6	NA	13.5
PCI_OE#	O	3.1	NA	16.7	2.7	NA	15.0

Table 7–6. 654 Controller Output Timing Characteristics By Signal (Continued)

654 Output Signal	I/O	3.3v V_{DD} System (2)			3.6v V_{DD} System (1)		
		Output Hold Time (3) (ns) Min	Output Tri-state Delay (4) (ns) Max	Output Valid Delay (5) (ns) Max	Output Hold Time (3) (ns) Min	Output Tri-state Delay (4) (ns) Max	Output Valid Delay (5) (ns) Max
PCI_SEL#	O	3.3	NA	16.0	2.9	NA	14.4
RASHI/CASLO	O	3.0	NA	15.7	2.6	NA	14.1
REFRESH_SEL#	O	2.9	NA	14.3	2.5	NA	12.9
ROM_SEL#	O	3.1	NA	14.9	2.7	NA	13.4
TT_ERR#	O	3.4	NA	15.3	2.8	NA	12.9

Notes for Table 7–6.

1) 10°C $\leq T_J \leq$ 85°C, 3.4v $\leq V_{DD} \leq$ 3.8v

2) 10°C $\leq T_J \leq$ 85°C, 3.0v $\leq V_{DD} \leq$ 3.6v.

3) Minimum output delay from CPU_CLK rising edge to signal invalid. See Section 7.1.7. Values shown reflect fastest process variables, VDD = max (3.6v for V_{DD} = 3.3v nominal, or 3.8v for V_{DD} = 3.6v nominal), and T_J = 10°C. This value applies to signals that are being driven to a new logic level by the CPU_CLK rising edge and to signals that are being tri-stated by the CPU_CLK rising edge (synchronous output disables).

4) Maximum output delay from CPU_CLK rising edge to signal tri-state (output driver turned off). Value shown reflects low V_{DD} (3v for V_{DD} = 3.3v nominal, or 3.4v for V_{DD} = 3.6v nominal), T_J = 85°C and slowest process variables. Value derived by placing two of the same type of drivers in contention, gating one off (on the rising edge of CPU_CLK), and determining the time required for the other driver to drive the line to a valid logic level.

5) Maximum output delay from CPU_CLK rising edge to signal valid. Value shown is slowest of up-transition from 0v to 2v or down-transition from low V_{DD} (3v for V_{DD} = 3.3v nominal, or 3.4v for V_{DD} = 3.6v nominal) to .8v, and reflects T_J = 85°C and slowest process variables.

NA) These outputs are never synchronously tri-stated.

7.4.3 Asynchronous Signals in the 654

The 654 Controller generally operates as a synchronous state machine. However, there are asynchronous paths from some of the 654 inputs to some of the 654 outputs. Some of these paths are provided to speed up system operation by allowing the bridge to enable pathways before they are officially selected by a qualifying clock edge. In all cases, these asynchronous paths are benign as long as the 60X CPU and other system components are operating properly. The following sections describe these asynchronous paths in detail. The effects of the test inputs (TEST#, DI#, and RI#) are not included in this section.

7.4.3.1 AACK#

Asynchronously affected by—L2_PRESENT#.

The 654 Controller drives AACK# continuously while L2_PRESENT# is high. If L2_PRESENT# is low, then AACK# is normally (at bus idle) tri-stated. During CPU mastered transactions, AACK# is output enabled during CPU_CLK cycle 4. After it is driven low for one cycle at the end of the transfer, it is driven high for one cycle, and then tri-stated. During snoop cycles (PCI bus master to memory transactions) AACK# is output enabled only while CPU_ADDR_OE# is asserted. AACK# is enabled during address-only CPU cycles.

7.4.3.2 ALL_ONES_SEL#

Asynchronously affected by—CPU_ADDR[0:8,19,29:31], TBST#, TSIZ[0:2], TT[0:3].

The 654 Controller normally asserts and negates this output on the rising edge of CPU_CLK. There are asynchronous paths to this output from the inputs shown above. These inputs must remain stable during transactions in order to guarantee that this output does not glitch. This condition is normally satisfied if the 60X CPU is operating correctly. This output is continuously output enabled.

7.4.3.3 CAS[7:0]#

Asynchronously affected by—MC_SETUP#, TT[1].

The 654 Controller normally asserts and negates these signals on the rising edge of CPU_CLK. There is an asynchronous path to these signals from TT[1] which must remain stable during transactions in order to guarantee that these signals do not glitch. This condition is normally satisfied if the 60X CPU is operating correctly. For the relationship between CAS[7:0]# and MC_SETUP# see Section 5.2.3.

7.4.3.4 CPU_ADDR_SEL#

Asynchronously affected by—RESET#.

While RESET# is asserted, the 654 Controller forces CPU_ADDR_SEL# high. While RESET# is high, CPU_ADDR_SEL# is asserted and negated on the rising edge of CPU_CLK. This output is continuously output enabled.

7.4.3.5 CPU_DATA_SEL#

Asynchronously affected by—TT[1].

The 654 controls CPU_DATA_SEL# by gating it (controlling its assertion & negation) with TT[1] while the CPU has control of the bus. In other words, CPU_DATA_SEL# is controlled asynchronously by TT[1] while the CPU has a valid bus grant. At the end of a CPU mastered transaction during which CPU_GNT# is removed from the CPU, CPU_DATA_SEL# is negated on the rising edge of the CPU_CLK on which AACK# is negated. This output is continuously output enabled.

7.4.3.6 ERR_ADDR_SEL#

Asynchronously affected by—CPU_ADDR[0:8].

The 654 Controller normally asserts and negates this output on the rising edge of CPU_CLK. There are asynchronous paths to this output from the inputs shown above. These inputs must remain stable during transactions in order to guarantee that this output does not glitch. This condition is normally satisfied if the 60X CPU is operating correctly. This output is continuously output enabled.

7.4.3.7 MEM_DATA_OE#

Asynchronously affected by—CPU_ADDR[0:8,29:31], TBST#, TSIZ[0:2], TT[1,3].

The 654 Controller normally asserts and negates this output on the rising edge of CPU_CLK. There are asynchronous paths to this output from the inputs shown above. These inputs must remain stable during transactions in order to guarantee that this output does not glitch. This condition is normally satisfied if the 60X CPU is operating correctly. This output is continuously output enabled.

7.4.3.8 MEM_DATA_SEL#

Asynchronously affected by—CPU_ADDR[0:8,29:31], L2_CLAIM#, TBST#, TSIZ[0:2], TT[1,3].

The 654 Controller normally asserts and negates this output on the rising edge of CPU_CLK. There are asynchronous paths to this output from the inputs shown above. These inputs must remain stable during transactions in order to guarantee that this output does not glitch. This condition is normally satisfied if the 60X CPU is operating correctly. This output is continuously output enabled.

7.4.3.9 PCI_C/BE[3:0]#

Asynchronously affected by—CPU_ADDR[30,31], TSIZ[1:2], TT[1].

The 654 Controller normally asserts and negates these outputs on the rising edge of CPU_CLK that corresponds to a rising edge of the PCI_CLK. There are asynchronous paths to this output from the inputs shown above. These inputs must remain stable during transactions in order to guarantee that this output does not glitch. This condition is normally satisfied if the 60X CPU is operating correctly. This output is continuously output enabled.

7.4.3.10 PCI_PAR

Asynchronously affected by—PCI_AD_PAR, MEM_PAR_GOOD, TT[1].

PCI_PAR is generated asynchronously from the PCI_AD_PAR and MEM_PAR_GOOD signals and is also affected by the type of transaction occurring in the system. Figure 7–6 shows the 654 Controller asserting PCI_PAR on the rising edge of PCI_CLK following the data phase from which it is generated. During CPU to PCI write data phases and PCI to memory read data phases, PCI_PAR specifications meet or exceed those required by the PCI specification. The state of this output at other times is not specified, but is designed to be benign.

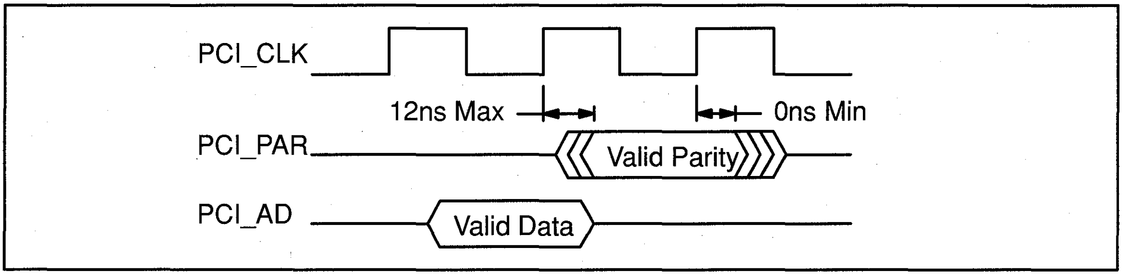


Figure 7–11. Timing of PCI_PAR

7.4.3.11 TA#

Asynchronously affected by—L2_PRESENT#.

The 654 Controller drives TA# continuously while L2_PRESENT# is high. If L2_PRESENT# is low then TA# is normally (at bus idle) tri-stated. During CPU mastered transactions, TA# is output enabled during CPU_CLK cycle 4. After it is driven low for one cycle with AACK# at the end of the transfer, it is driven high for one cycle, and then tri-stated. During snoop cycles (PCI bus master to memory transactions) TA# stays tri-stated.

7.4.3.12 WE[1:0]

Asynchronously affected by—CPU_ADDR[0:8,29:31], TBST#, TSIZ[0:2], TT[1,3].

The 654 Controller normally asserts and negates this output on the rising edge of CPU_CLK. There are asynchronous paths to this output from the inputs shown above. These inputs must remain stable during transactions in order to guarantee that this output does not glitch. This condition is normally satisfied if the 60X CPU is operating correctly. This output is continuously output enabled.

7.5 653 Buffer Timing Tables

Table 7–7. 653 Buffer Timing Tables

Des	**CPU Interface** (See Notes 1,2,3)	Note	V_{DD} = 3.3v		V_{DD} = 3.6v	
			Min (ns)	Max (ns)	Min (ns)	Max (ns)
t1	CPU_ADDR, TSIZ setup to CPU_ADDR_SEL# fall	6	2	—	2	—
t2	CPU_ADDR, TSIZ hold from CPU_ADDR_SEL# fall	6	3	—	3	—
t3	CPU_ADDR valid from CPU_ADDR_OE# fall	4,7	3	21	2	17
t4	CPU_ADDR float from CPU_ADDR_OE# rise	7	3	21	3	17
t4a	CPU_ADDR held valid from CPU_ADDR_OE# rise	7	3	21	3	17
t5,5a	CPU_ADDR valid (CPU snoop) from PCI_AD valid (addr phase)	16	3	17	2	14
t74	CPU_ADDR valid from PCI_SEL# fall (CPU snoop)	17	3	19	3	16
t74a	CPU_ADDR held valid from PCI_SEL# rise (CPU snoop)	17	3	19	3	16
t75	CPU_ADDR valid from BURST_CLK# fall (CPU snoop)	17	4	19	4	17
t76	CPU_ADDR valid from NO_TRANS rise/fall (CPU snoop)	17	4	17	3	14
t6	CPU_DATA valid from MEM_DATA valid	8	4	16	4	13
t7	CPU_DATA valid from MEM_DATA_SEL# fall	9	4	19	4	16
t7a	CPU_DATA held valid from MEM_DATA_SEL# rise	9	4	19	4	16
t8	CPU_DATA valid from CPU_DATA_OE# fall	4,10	4	19	5	20
t8a	CPU_DATA held valid from CPU_DATA_OE# rise	10	4	19	4	16
t9	CPU_DATA float from CPU_DATA_OE# rise	10	4	19	4	16
t11	CPU_DATA valid from PCI_AD valid (data phase)	25	4	18	4	15
t12	CPU_DATA valid from ERR_ADDR_SEL# fall	11	4	20	4	17
t13	CPU_DATA valid from ALL_ONES_SEL#	5	5	20	4	17
t14	L_ERR_ADDR# setup to CPU_ADDR_SEL#		8	—	8	—
t15	L_ERR_ADDR# hold from CPU_ADDR_SEL#		1	—	1	—

Des	**Memory Interface** (See Notes 1,2,3)	Note	V_{DD} = 3.3v		V_{DD} = 3.6v	
			Min (ns)	Max (ns)	Min (ns)	Max (ns)
t21	MEM_ADDR valid from CPU_ADDR valid	13	4	16	3	14
t22	MEM_ADDR valid from CPU_ADDR_SEL# fall	16	4	21	4	19
t23	MEM_ADDR valid from RASHI/CASLO change	18	4	16	3	14
t23a	MEM_ADDR held valid from RASHI/CASLO change	18	4	16	3	14
t24	MEM_ADDR valid from BURST_CLK# fall	14	5	19	4	16

Des	Memory Interface (See Notes 1,2,3)	Note	V_{DD} = 3.3v		V_{DD} = 3.6v	
			Min (ns)	Max (ns)	Min (ns)	Max (ns)
t25	MEM_ADDR valid from PCI_AD valid	17	4	17	3	14
t26	MEM_ADDR valid from BURST_CLK# fall	15	5	19	4	16
t77	MEM_ADDR valid from PCI_SEL# fall	17	4	19	4	16
t77a	MEM_ADDR held valid from PCI_SEL# rise	17	4	19	4	16
t78	MEM_ADDR valid from DRAMX9HI/X10LO rise/fall	18	4	17	3	15
t78a	MEM_ADDR valid from DRAMX9HI/X10LO rise/fall	22	4	17	3	15
t79	MEM_ADDR valid from NO_TRANS rise/fall	17	4	17	3	15
t27	MEM_ADDR valid (row address) from REFRESH_SEL# fall	22	4	20	3	17
t27a	MEM_ADDR held valid (row addr) from REFRESH_SEL# rise	22	4	20	3	17
t80	REFRESH_SEL# rise to valid & stable new refresh address	22	4	20	3	17
t81	REFRESH_SEL# high time (for correct counter operation)	22	6	—	6	—
t82	REFRESH_SEL# low time (for correct counter operation)	22	6	—	6	—
t82	REFRESH_SEL# period (for correct counter operation)	22	27	—	27	—
t28	MEM_PAGE_HIT# valid from CPU_ADDR valid	16	3	15	3	13
t28a	MEM_PAGE_HIT# valid from CPU_ADDR_SEL# fall	6	4	17	4	15
t29	MEM_PAGE_HIT# valid from RASHI/CASLO rise (row addr)	18	4	16	3	14
t30	MEM_PAGE_HIT# valid from PCI_AD valid	17	4	16	3	13
t83	MEM_PAGE_HIT# valid from PCI_SEL# fall	17	4	18	4	15
t31	MEM_PAGE_HIT# valid from ADDRHI/DATALO rise		4	19	7	17
t32	MEM_PAGE_HIT# valid from BURST_CLK# fall	21	5	21	5	18
t84	MEM_PAGE_HIT# valid from DRAMX9HI/X10LO rise/fall	18	4	15	3	13
t33	MEM_DATA valid from CPU_DATA valid	13	4	16	4	13
t34	MEM_DATA valid from CPU_DATA_SEL# fall	13	4	18	4	15
t35	MEM_DATA valid from MEM_DATA_OE# fall	4,12	5	21	4	17
t35a	MEM_DATA held valid from MEM_DATA_OE# rise	12	5	21	4	17
t36	MEM_DATA float from MEM_DATA_OE# rise	12	5	21	4	17
t37	MEM_DATA valid from PCI_AD valid	23	4	16	3	13
t85	MEM_DATA valid from PCI_SEL# fall	23	4	18	4	15
t86	MEM_PAR (out) valid from PCI_SEL# fall	23	4	18	4	16
t38	MEM_PAR (out) valid from CPU_DATA valid	13	4	16	3	14

Des	Memory Interface (See Notes 1,2,3)	Note	V_{DD} = 3.3v		V_{DD} = 3.6v	
			Min (ns)	Max (ns)	Min (ns)	Max (ns)
t87	MEM_PAR (out) valid from CPU_DATA_SEL# valid	13	4	19	4	16
t38a	MEM_PAR (out) valid from PCI_AD valid (data phase)	23	3	17	3	14
t39	MEM_PAR (out) valid from MEM_DATA_OE# fall	4,19	4	18	3	15
t39a	MEM_PAR (out) held valid from MEM_DATA_OE# rise	19	4	18	3	15
t40	MEM_PAR (out) float from MEM_DATA_OE# rise	19	4	18	3	15
t41a	MEM_PAR_GOOD valid from MEM_PAR (in) valid	20	3	16	3	13
t41b	MEM_PAR_GOOD valid from MEM_DATA (in) valid	20	3	16	3	13
t88	MEM_PAR_GOOD valid from MEM_DATA_SEL# fall	20	3	13	3	11
t88a	MEM_PAR_GOOD forced high from MEM_DATA_SEL# rise		3	13	3	11

Des	PCI Interface (See Notes 1,2,3)	Note	V_{DD} = 3.3v		V_{DD} = 3.6v	
			Min (ns)	Max (ns)	Min (ns)	Max (ns)
t51	PCI_AD valid (addr phase) from CPU_ADDR valid	26	4	17	4	15
t89	PCI_AD valid (addr phase) from CPU_ADDR_SEL# fall	26	4	22	4	19
t52	PCI_AD valid (data phase) from CPU_DATA valid	24	4	13	3	12
t53	PCI_AD valid (data phase) from CPU_DATA_SEL# fall	24	4	16	4	14
t90	PCI_AD valid from PCI_SEL# fall	25	4	20	4	17
t90a	PCI_AD held valid from PCI_SEL# rise	25	4	20	4	17
t54	PCI_AD valid from PCI_OE# fall	4,27	3	13	3	11
t54a	PCI_AD held valid from PCI_OE# rise	27	3	13	3	11
t55	PCI_AD float from PCI_OE# rise	27	3	13	3	11
t91	PCI_AD valid (data phase) from ERR_ADDR_SEL#		4	17	4	15
t92	PCI_AD valid (address phase) from NO_TRANS rise/fall	26	4	18	4	14
t106	PCI_AD valid (data phase) from MEM_DATA valid		3	12	3	11
t56	PCI_AD_PAR valid (address phase) from CPU_ADDR valid	26	5	29	5	21
t57	PCI_AD_PAR valid (addr phase) from CPU_ADDR_SEL# fall	26	6	33	5	25
t58	PCI_AD_PAR valid (data phase) from CPU_DATA valid	26	5	25	5	18
t59	PCI_AD_PAR valid (data phase) from CPU_DATA_SEL# fall	26	5	27	5	20
t93	PCI_AD_PAR valid (data phase) from MEM_DATA valid	29	4	19	4	16
t94	PCI_AD_PAR valid from PCI_AD valid		5	25	4	22
tA1	PCI_SEL# setup to PCI_CLK rise (address phase)	30	2	—	2	—

Des	**PCI Interface** (See Notes 1,2,3)	Note	V_{DD} = 3.3v		V_{DD} = 3.6v	
			Min (ns)	Max (ns)	Min (ns)	Max (ns)
tAx	PCI_SEL# hold from PCI_CLK rise (address phase)	30	2	—	2	—
t112	PCI_AD valid from BURST_CLK# fall (2nd 4-byte of 8-byte read)		3	12	3	12
t60	ADDRHI/DATALO setup to PCI_CLK rise or fall		2	—	2	—
t61	ADDRHI/DATALO hold from PCI_CLK rise or fall		2	—	2	—
t62	PCI_AD setup to PCI_CLK rise (all cycles, all phases)		0	—	0	—
t63	PCI_AD hold from PCI_CLK rise (all cycles, all phases)		4	—	3	—
tB2	L_PCI_DATA# setup to PCI_CLK rise or fall	31	1	—	0	—
tB1	L_PCI_DATA# hold from PCI_CLK rise or fall	31	2	—	1	—
tC1	ADDRHI/DATALO low setup to PCI_CLK rise (data phase)	32	2	—	1	—
tC2	ADDRHI/DATALO low hold from PCI_CLK rise (data phase)	32	2	—	0	—
tC3	PCI_CLK to PCI_AD (output data phase) valid	32	4	21	4	18
t64	L_PCI_DATA# setup to PCI_CLK		1	—	1	—
t65	L_PCI_DATA# hold from PCI_CLK		2	—	1	—
t66	PCI_AD valid (data phase) from MEM_DATA valid	29	3	12	3	11
t67	PCI_AD valid (data phase) from MEM_DATA_SEL# fall	29	4	16	4	14

Des	**ROM Engine** (See Notes 1,2,3)	Note	V_{DD} = 3.3v		V_{DD} = 3.6v	
			Min (ns)	Max (ns)	Min (ns)	Max (ns)
t68	PCI_AD[23:0] valid (ROM address) from ROM_SEL# fall	4, R/W	4	20	3	17
t68a	PCI_AD[23:0] held valid (ROM address) from ROM_SEL# rise	R/W	4	20	4	17
t69	PCI_AD[23:0] valid (ROM address) from BURST_CLK# fall	read	5	19	4	16
t70	PCI_AD[31:24] setup to BURST_CLK# fall	read	17	—	15	—
t71	PCI_AD[31:24] hold from BURST_CLK# fall	read	0	—	0	—
t10	CPU_DATA valid (last byte) from BURST_CLK# fall (ROM_SEL# low)	read	5	21	5	17
t95	CPU_DATA valid from ROM_SEL# fall	read	4	19	4	16
t95a	CPU_DATA held valid from ROM_SEL# rise	read	4	19	4	16

Des	**System Interface** (See Notes 1,2,3)	Note	V_{DD} = 3.3v		V_{DD} = 3.6v	
			Min (ns)	Max (ns)	Min (ns)	Max (ns)
t96	ALL_ONES_SEL# fall to PCI_AD valid (all 1's)	33	4	17	4	14
t97	ALL_ONES_SEL# fall to MEM_DATA valid		4	19	4	16

Des	**System Interface** (See Notes 1,2,3)	Note	V_{DD} = 3.3v		V_{DD} = 3.6v	
			Min (ns)	Max (ns)	Min (ns)	Max (ns)
t98	BURST_CLK# high pulse width		6	—	6	—
t99	BURST_CLK# low pulse width		6	—	6	—
t107	BURST_CLK# period		27	—	27	—
t100	CONTIG_IO rise or fall to PCI_AD valid (address phase)		—	13	—	12
t101	CPU_ADDR setup to L_ERR_ADDR# fall	34	4	—	3	—
t102	CPU_ADDR_SEL# fall to L_ERR_ADDR# fall	34	8	—	8	—
t103	PCI_AD setup to L_ERR_ADDR# fall	34	4	—	4	—
t104	PCI_SEL# fall to L_ERR_ADDR# fall	34	5	—	5	—
t105	REFRESH_SEL# fall to L_ERR_ADDR# fall	34	6	—	6	—
t72	Any PCI, memory, or bridge activity setup to LE_MODE_SEL#	35	30	—	30	—
t73	Any PCI, memory, or bridge activity hold to LE_MODE_SEL#	35	30	—	30	—

Notes:

1. For the times given for V_{DD} = 3.3v the following applies:
Up transitions are from 0 to 2v.
Slowest times are given for 3.0v V_{DD}, 85 deg C ambient, and slowest process variables.
Down transitions (unless noted) are from 3.0v to .8v.
Fastest times are given for 3.6v V_{DD}, 0 deg C ambient, and fastest process variables.
Down transitions (unless noted) are from 3.6v to .8v.

2. For the times given for V_{DD} = 3.6v the following applies:
Up transitions are from 0 to 2v.
Down transitions (unless noted) are from 3.6v to .8v.
Slowest times are given for 3.42v V_{DD}, 85 deg C ambient, and slowest process variables.
Fastest times are given for 3.6v V_{DD}, 0 deg C ambient, and fastest process variables.

3. All times are given for a 50pF capacitive load on each pin.

4. Bus valid from output enable active times are given for the slowest of:
(a) driving the bus down from 5.25v to .8v, or
(b) driving the bus up from 0v to 2v.

5. CPU to PCI configuration read with master abort, CPU interrupt acknowledge cycles.

6. CPU mastered transactions.

7. CPU snoop of PCI bus master transaction.

8. CPU to memory read.

9. CPU or PCI to memory read.

10. CPU read.

11. CPU to error address latch read.

12. CPU or PCI to memory write.

13. CPU to memory write.

14. CPU to memory burst.
15. PCI to memory burst.
16. CPU to memory transaction.
17. PCI to memory transaction.
18. Memory access.
19. Memory write.
20. Memory read.
21. PCI BM burst transfer
22. Memory refresh operation.
23. PCI to memory write.
24. CPU to PCI write.
25. CPU to PCI read.
26. CPU to PCI transaction.
27. CPU to PCI transaction address phase or PCI to memory read first data phase.
28. CPU to PCI read address to data phase transition or CPU to PCI write at end of data phase.
29. PCI to memory read.
30. PCI Bus Master address transaction address phase, showing the operation of the 653 Buffer PCI address latch. See Figure 7–12.
31. CPU to PCI read. PCI data latch operation. See Figure 7–13.
32. CPU to PCI write. PCI address/data MUX delay flip–flop operation. See Figure 7–14. With XADIO=1, the data is driven onto the PCI_AD lines 2 PCI_CLKs before IRDY# is sampled valid, yielding a maximum allowed PCI compliant specification of 1 PCI_CLK + 11ns.
33. PCI interrupt acknowledge address phase, etc..
34. Error address latch setup time from event to valid error address.
35. Including beginning any activity from either the CPU or a PCI bus master that involves a data or address path, or a data or address path control signal (CPU_ADDR_SEL#, MEM_DATA_SEL#, PCI_SEL#, etc.) in the 650 Bridge.

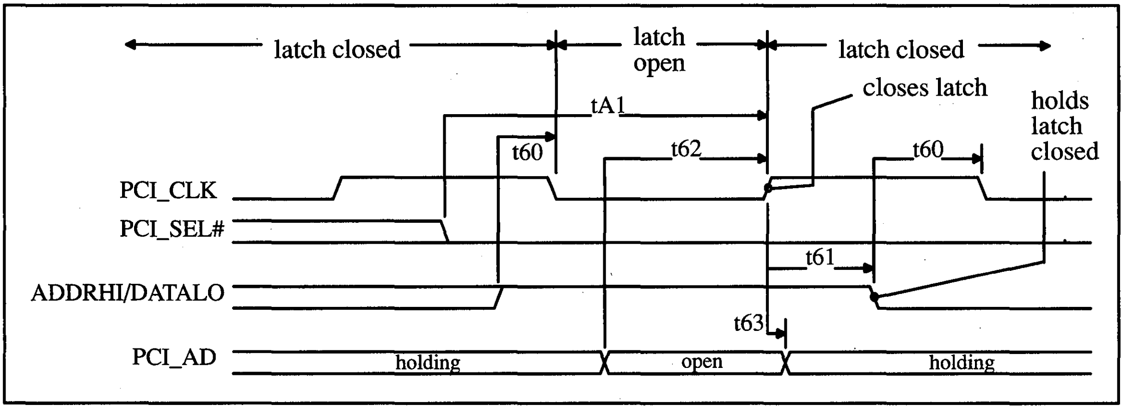


Figure 7–12. PCI Bus Master Transaction—Address Latch Operation

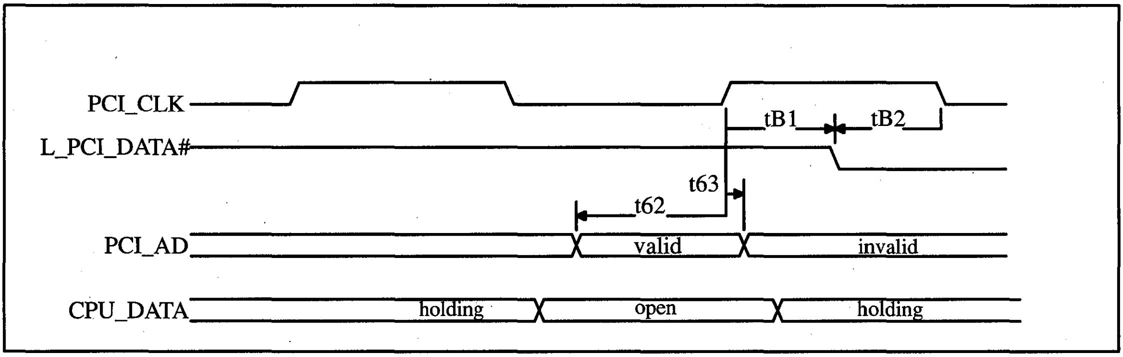


Figure 7–13. CPU to PCI Read—PCI Data Latch Operation

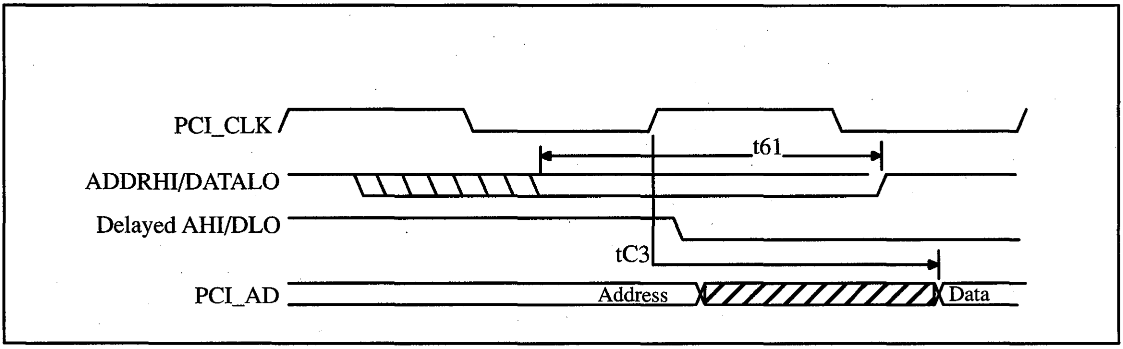


Figure 7–14. CPU to PCI Write—PCI Address/Data MUX

7.6 Detailed Timing Diagrams

This section contains timing diagrams of transactions and operations that can occur in the system.

Unless otherwise indicated, all timing specifications refer to events at the pins of the chip under discussion. Signals whose names are followed by a (C) are shown as if they were measured at the pin of the 654 Controller. Signal names followed by a (B) are shown as if they were measured at the pin of the 653 Buffer.

The source of some signals shown in the timing diagrams is indicated inside square brackets. For example, some signal names are followed by an [L2], and these signals are sourced by the L2 cache. They are based on the performance of an L2 cache built around an IBM27-82681-66 L2 Cache Controller chip. Some signal names are followed by a [target], and they are sourced by a PCI agent acting as a target. All signals shown that are sourced by a device other than the 650 Bridge are supplied for reference only, and they are not intended to specify the operation of the referenced device.

Some physical signal nets can be driven by more than one device. For example TA# can be driven by both the L2 cache controller and the memory controller during the same transaction. In this case the timing diagram line labeled TA# (C) [MC] shows the effects of the drivers in the memory controller (in the 654) on the TA# signal net (as measured at the pin of the 654 Controller), as if no other drivers were connected to the net. The timing diagram line labeled TA# (C) [L2] shows the effects of the drivers in the L2 cache on the TA# signal net (as measured at the pin of the 654 Controller), as if no other drivers were connected to the net. In this way, the activity of the various agents is fully described, and interactions between the drivers can be freely evaluated by the designer. The effects of the various drivers on the signal net can be derived by inspection.

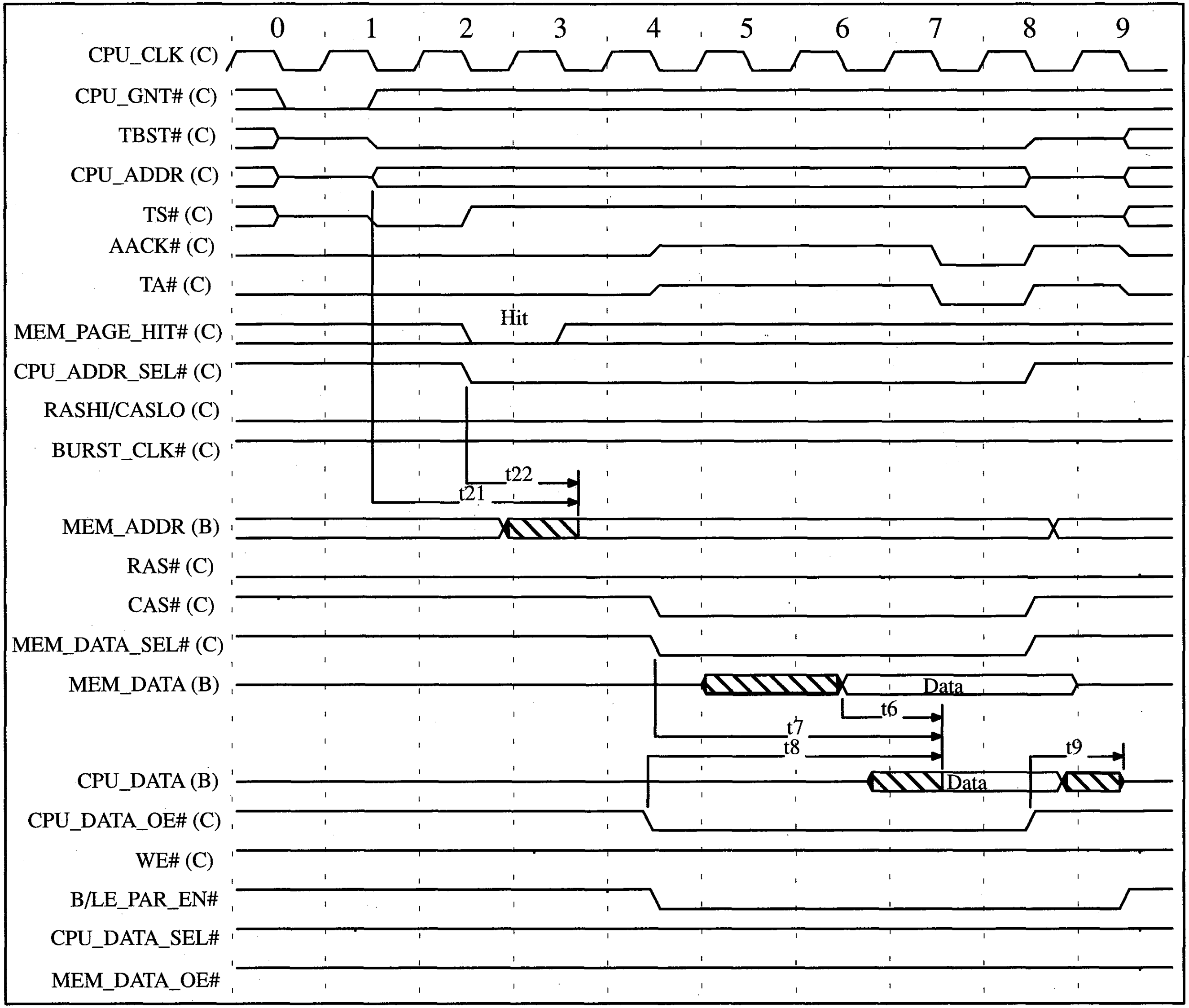


Figure 7–15. CPU To Memory Read – Single, Page Hit, XCAS=1

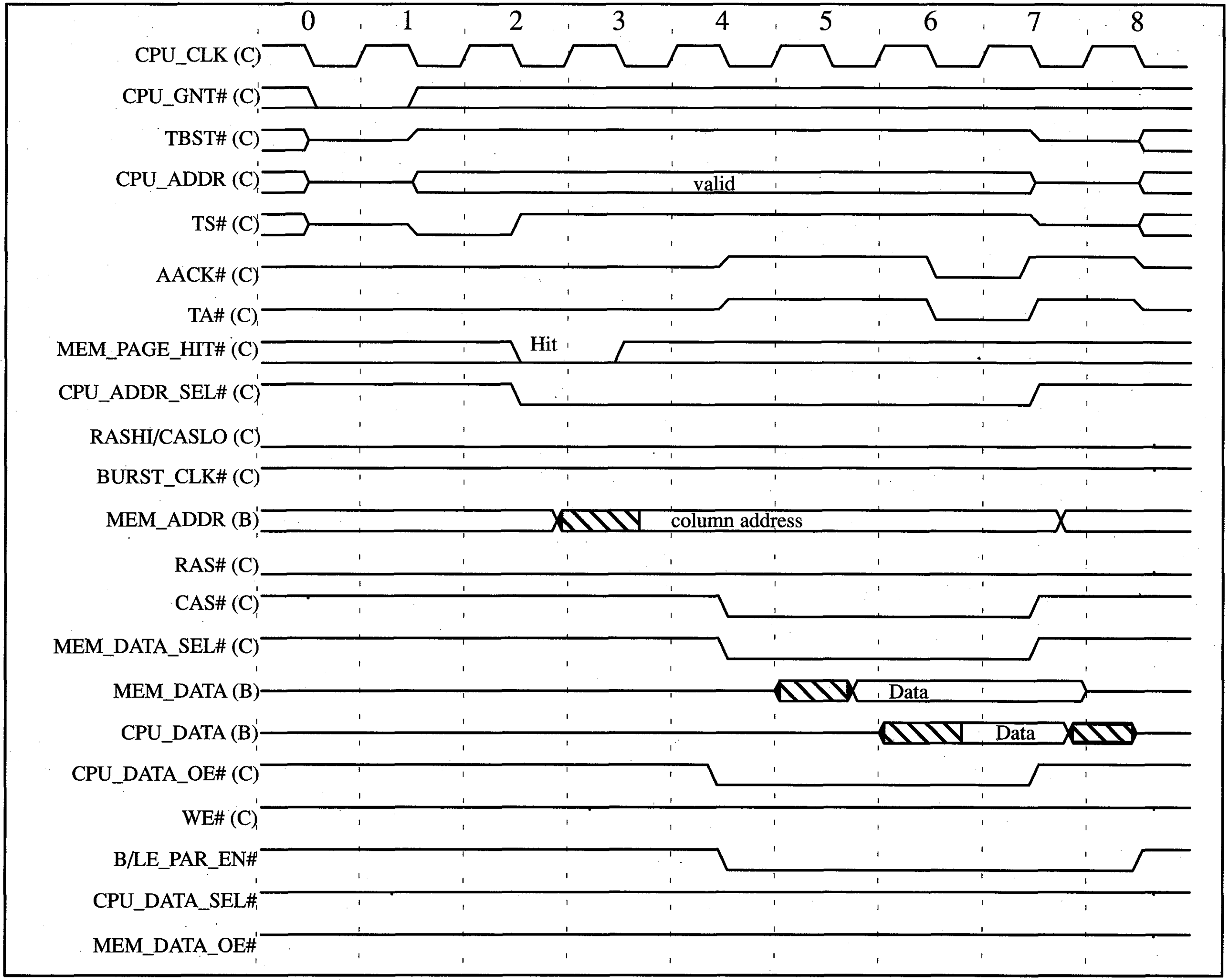


Figure 7–16. CPU To Memory Read – Single, Page Hit, XCAS=0

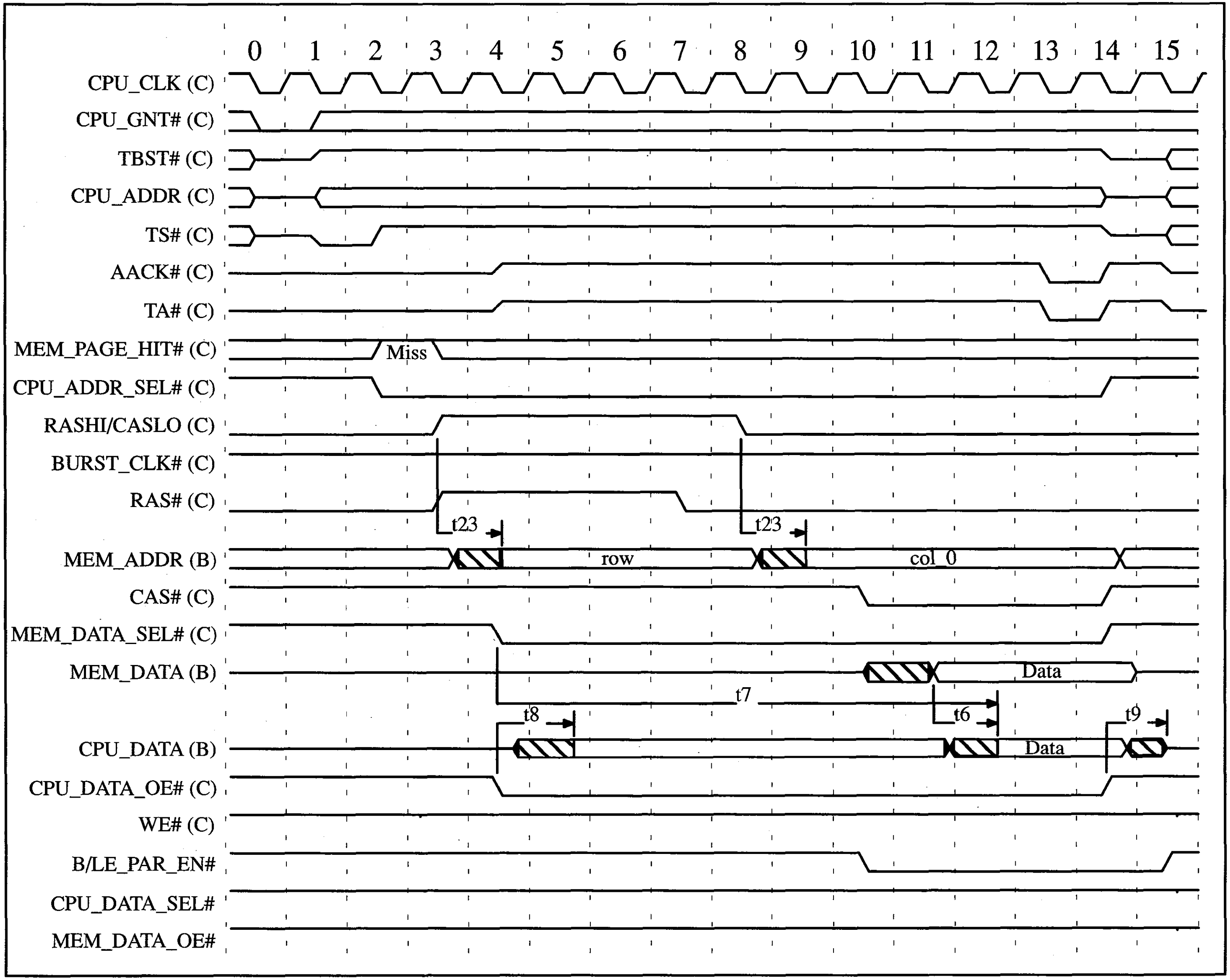


Figure 7–17. CPU To Memory Read – Single, Page Miss, XCAS=1

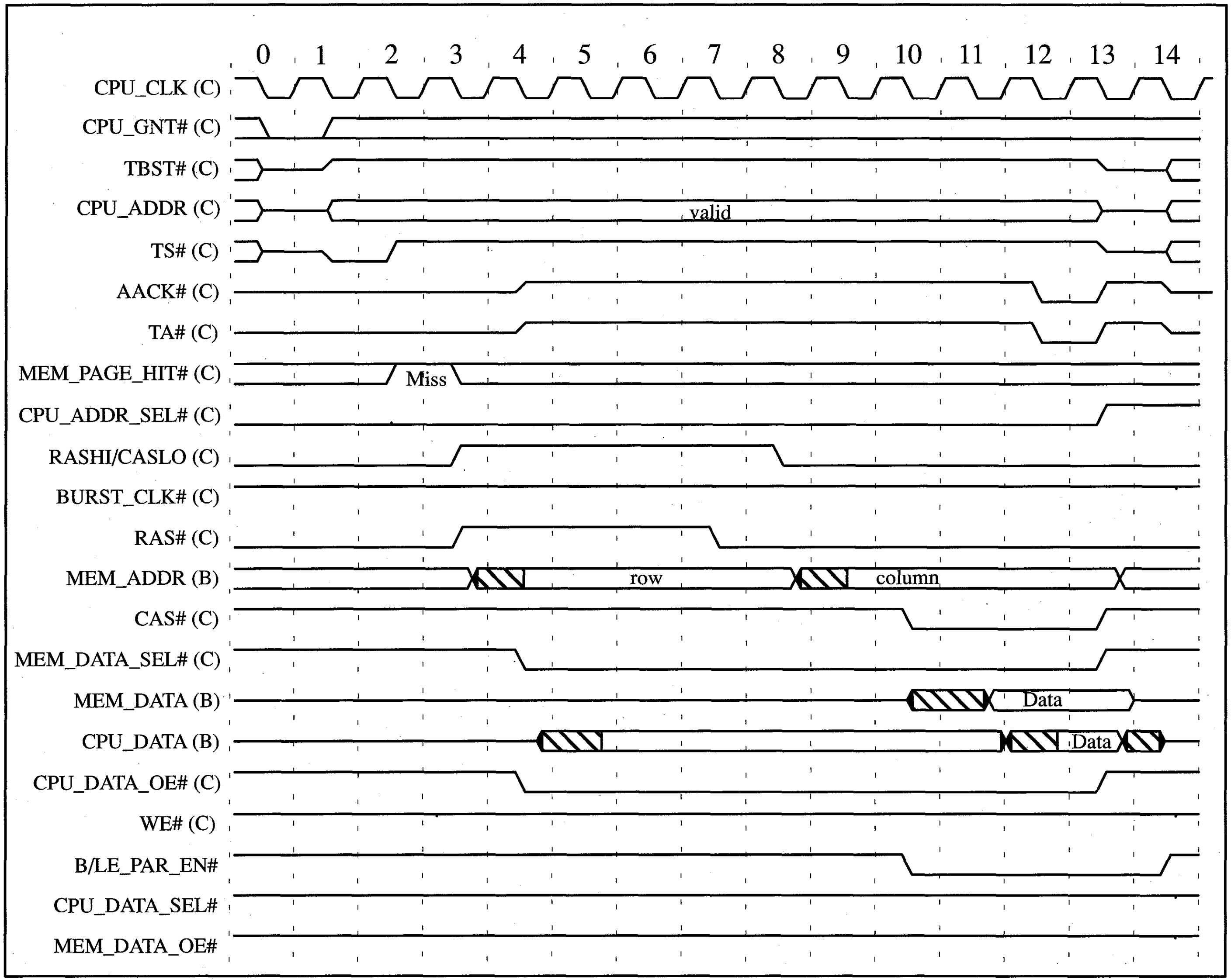


Figure 7–18. CPU To Memory Read – Single, Page Miss, XCAS=0

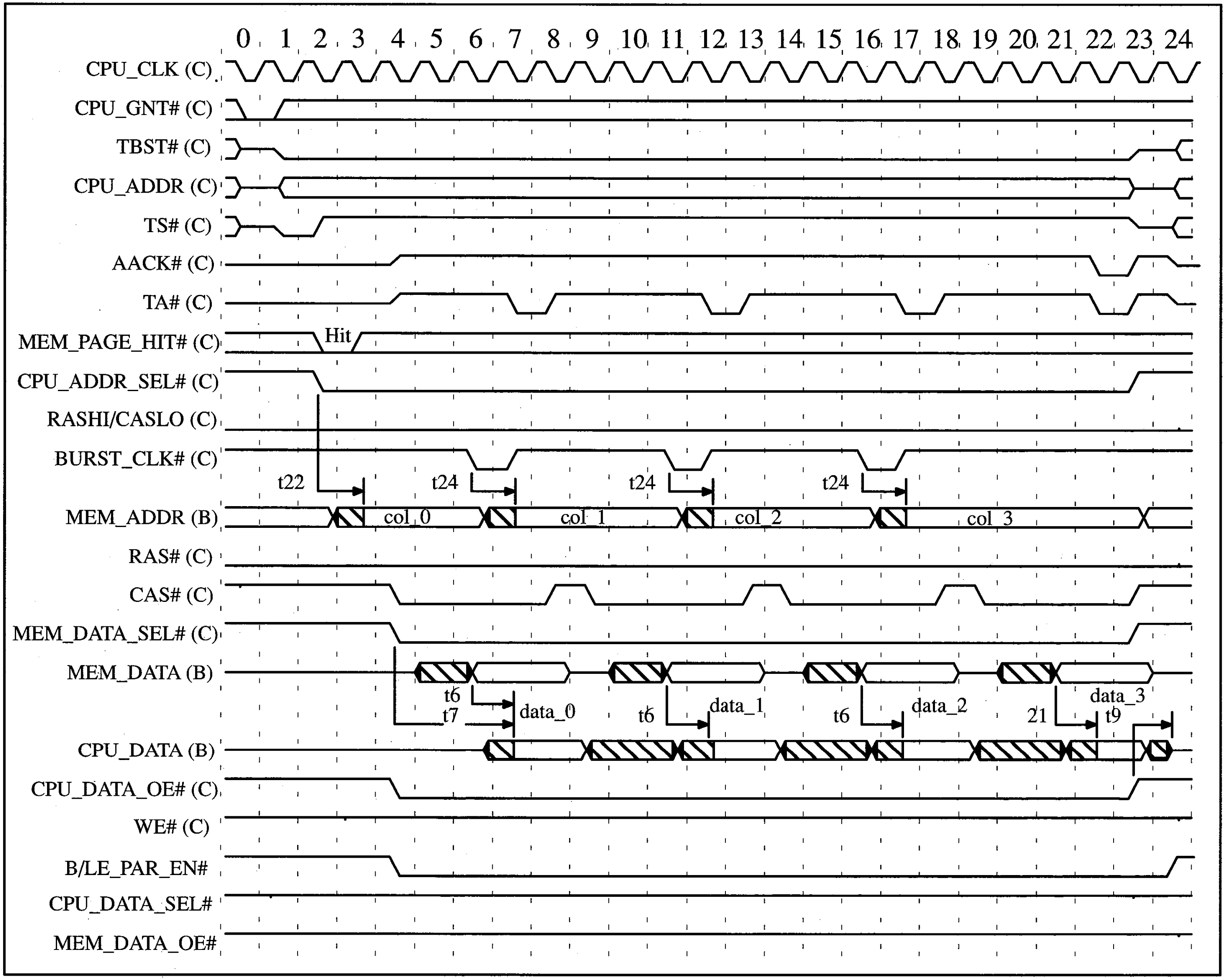


Figure 7–19. CPU To Memory Read – Burst, Page Hit, XCAS=1

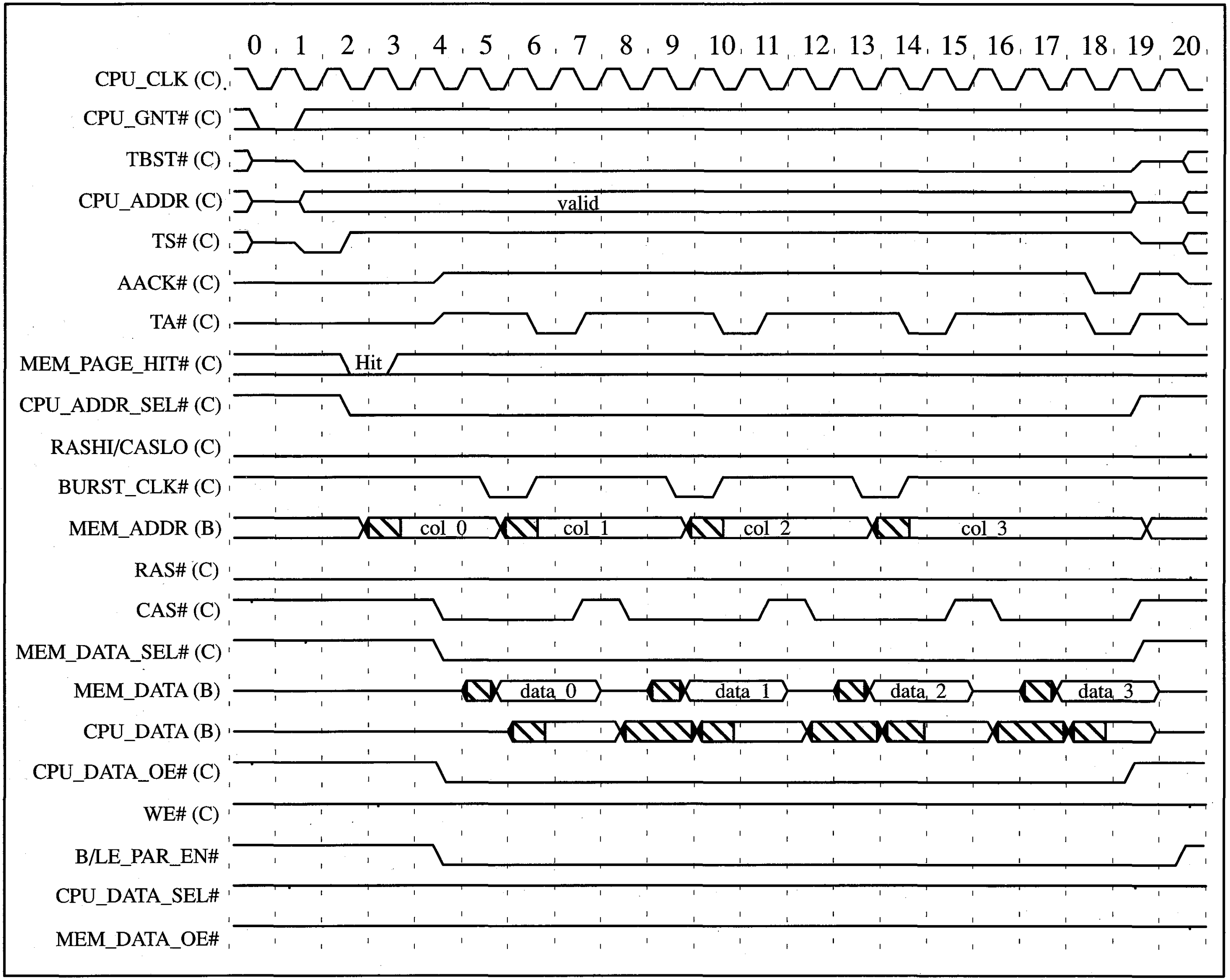


Figure 7–20. CPU To Memory Read – Burst, Page Hit, XCAS=0

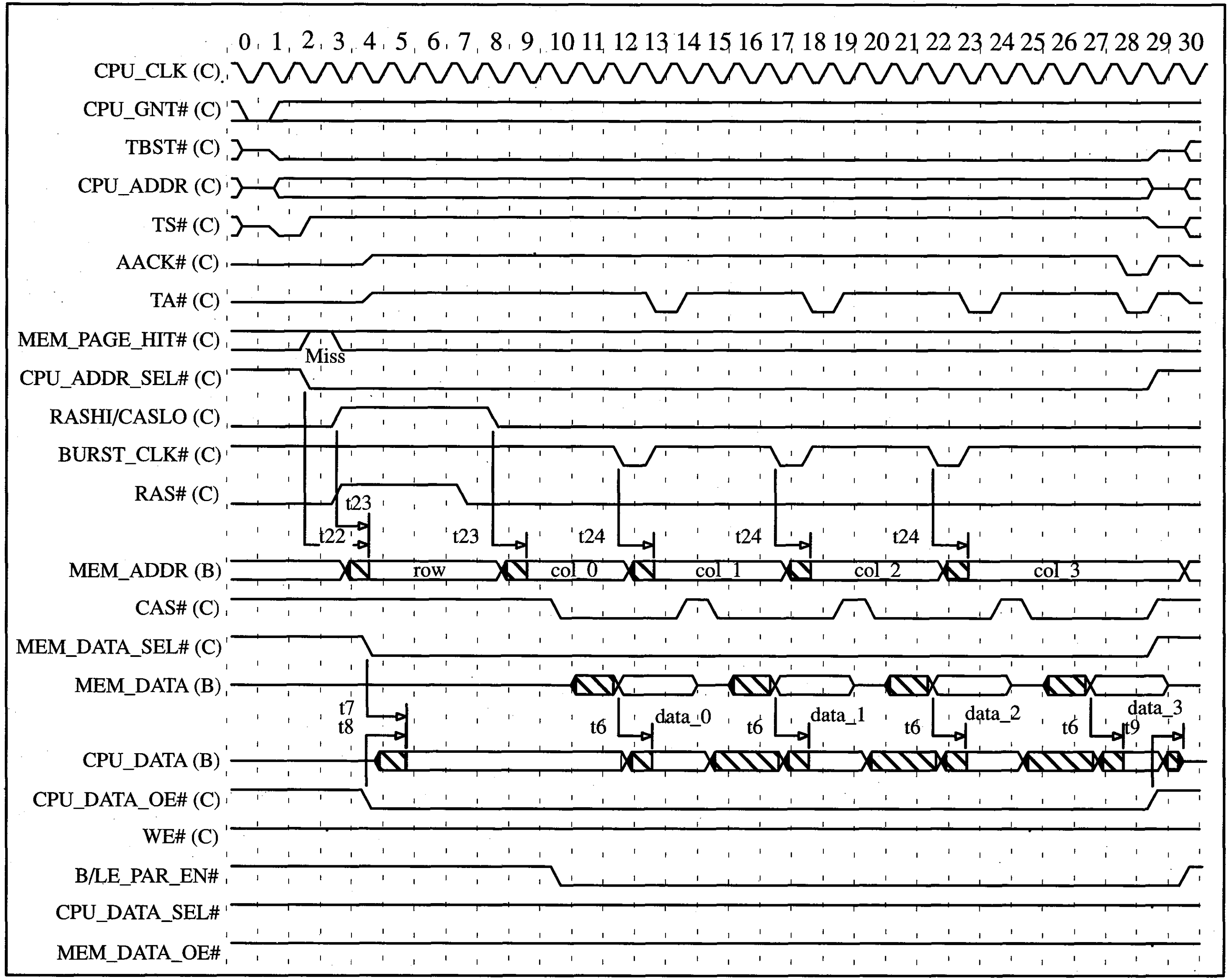


Figure 7–21. CPU To Memory Read – Burst, Page Miss, XCAS=1

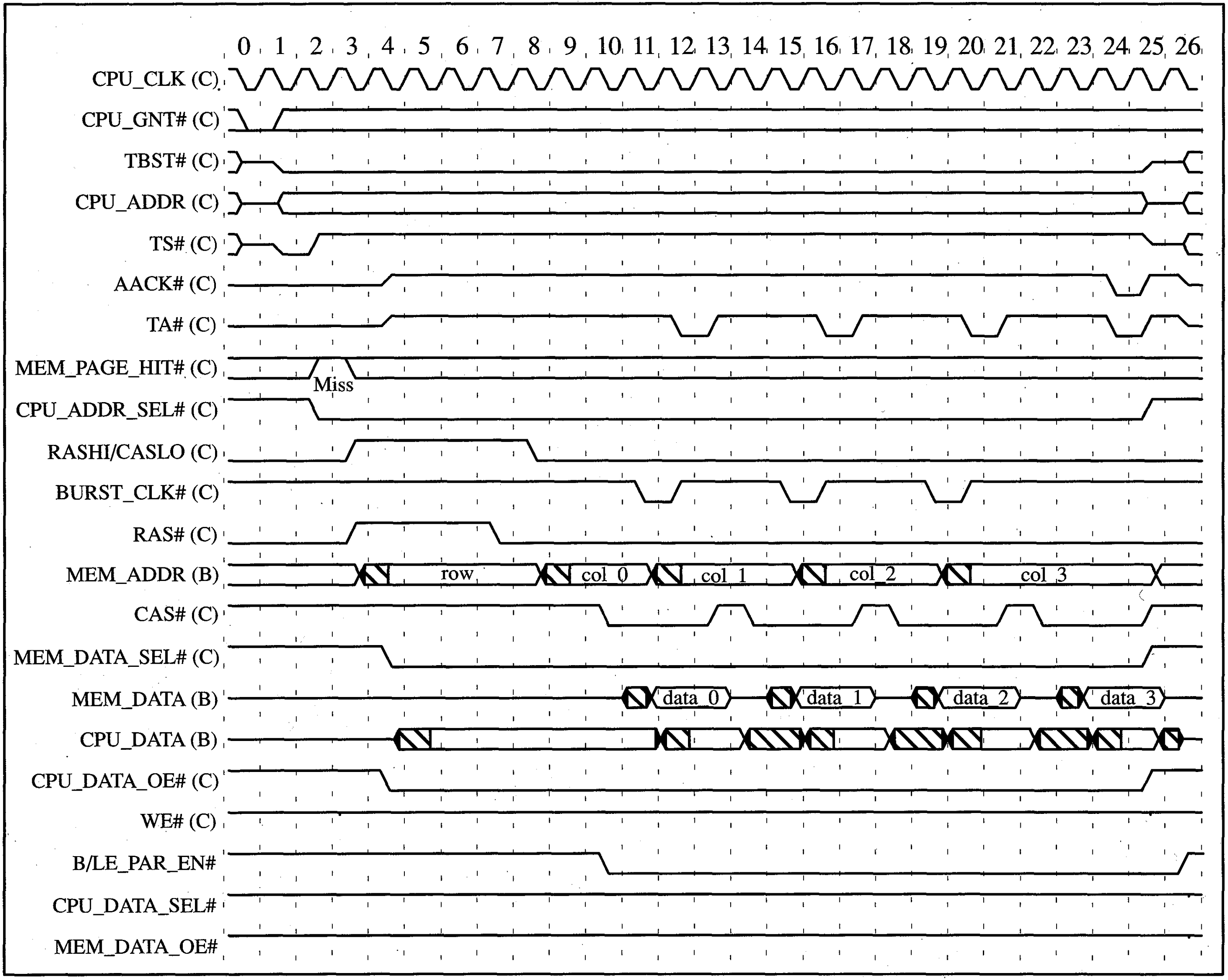
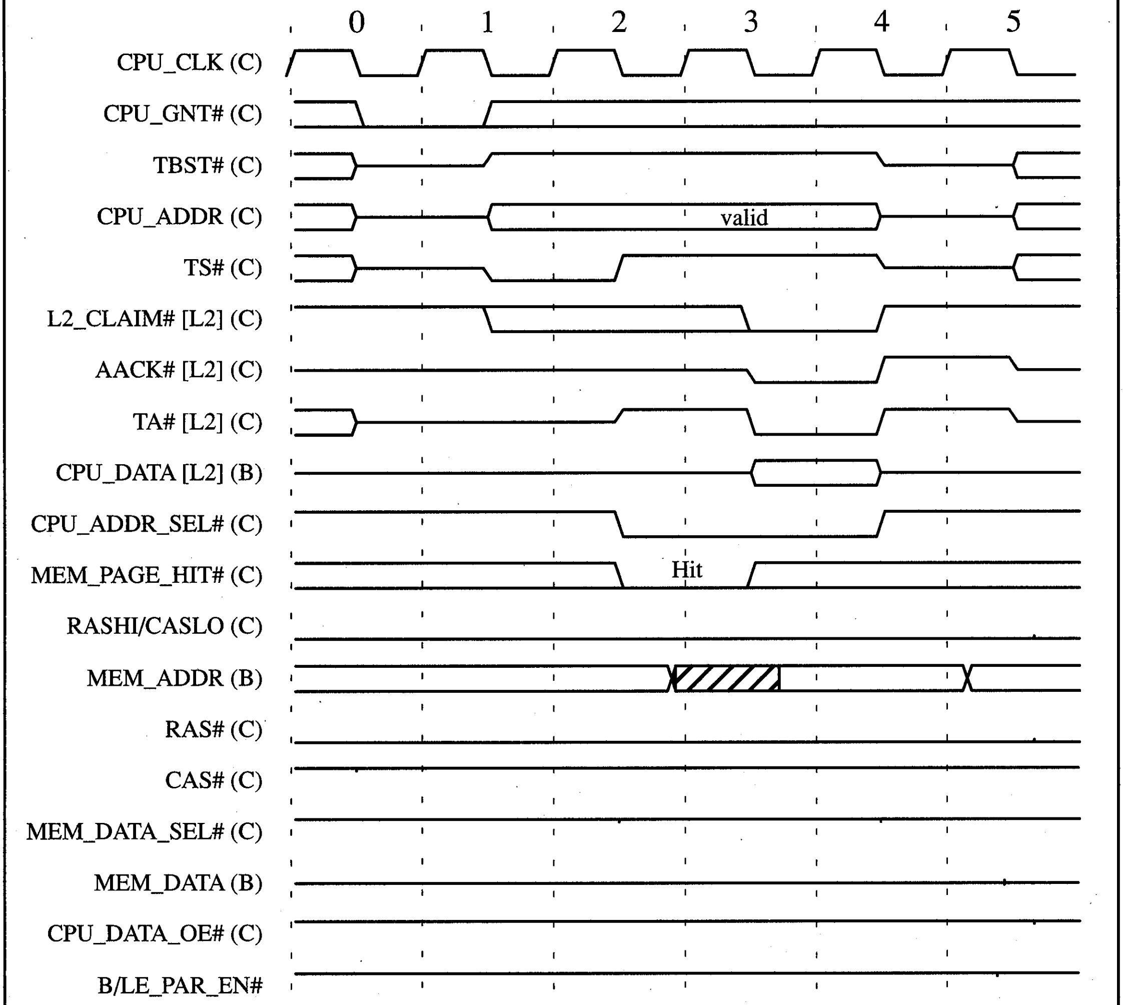


Figure 7–22. CPU To Memory Read – Burst, Page Miss, XCAS=0



654 does not enable TA#, AACK#
654 does not assert CPU_DATA_OE#

Figure 7–23. CPU To Memory Read – Single, Page Hit, L2 Cache Hit

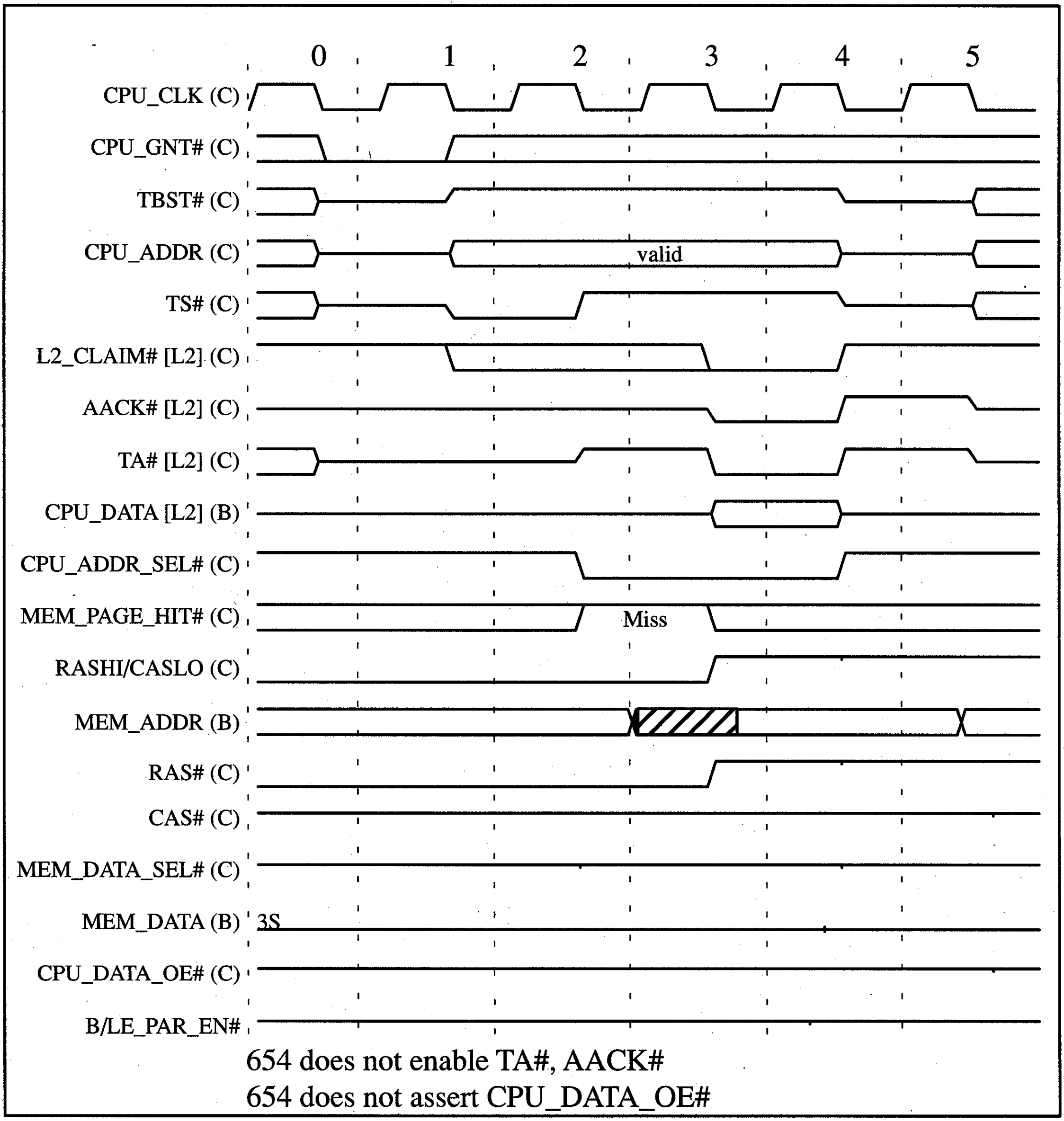


Figure 7–24. CPU To Memory Read – Single, Page Miss, L2 Cache Hit

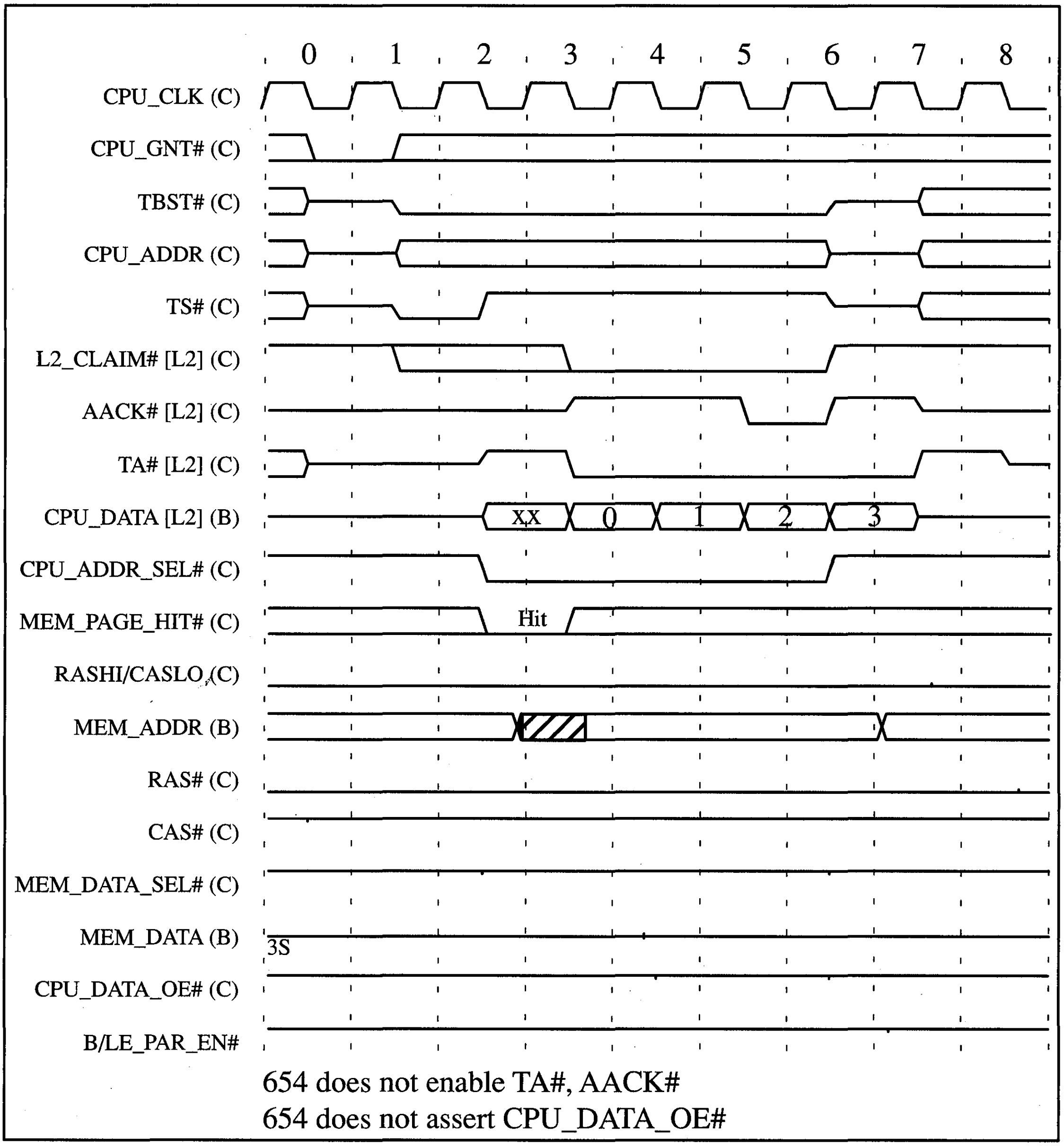


Figure 7–25. CPU To Memory Read – Burst, Page Hit, L2 Cache Hit

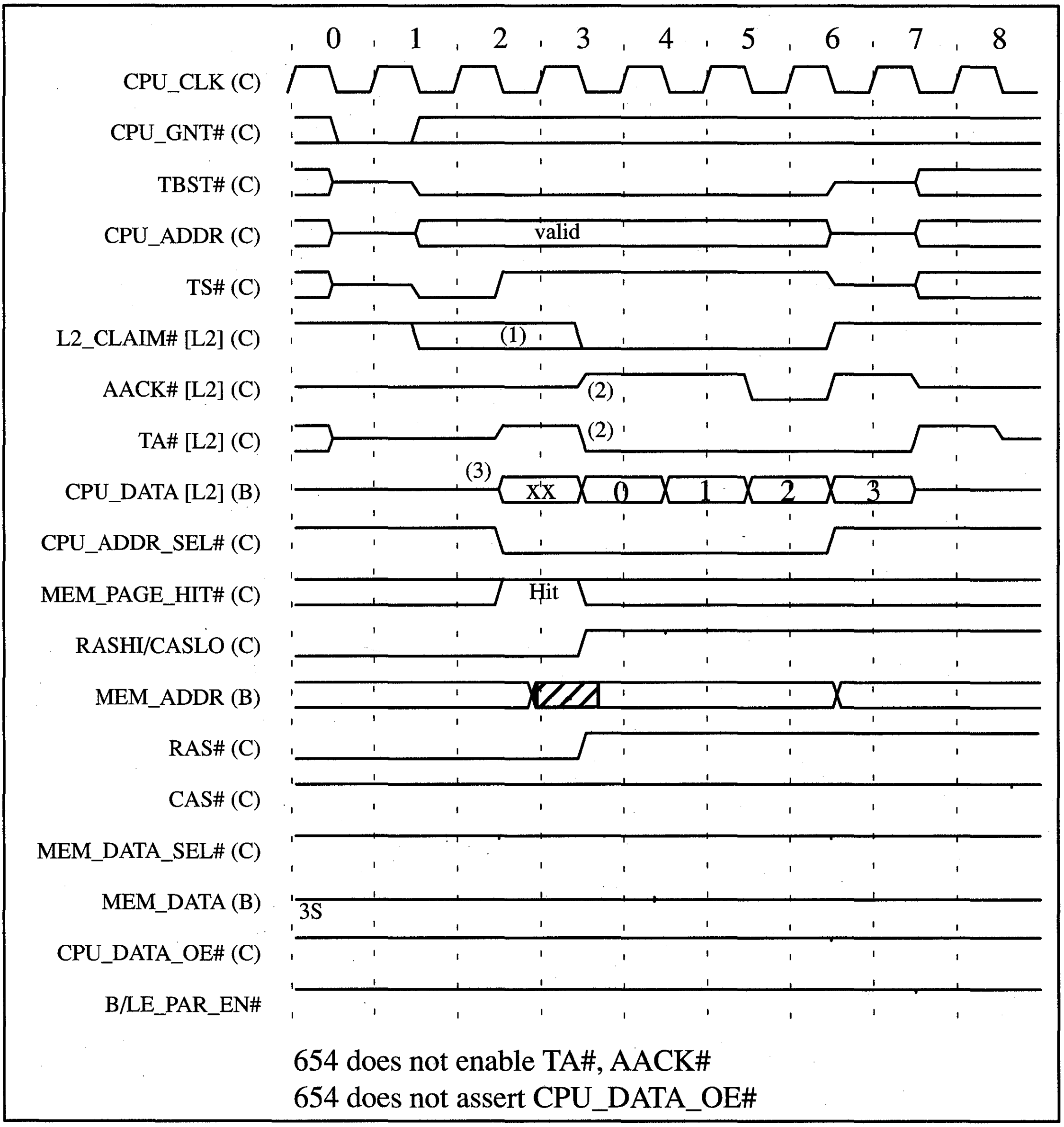


Figure 7–26. CPU To Memory Read – Burst, Page Miss, L2 Cache Hit

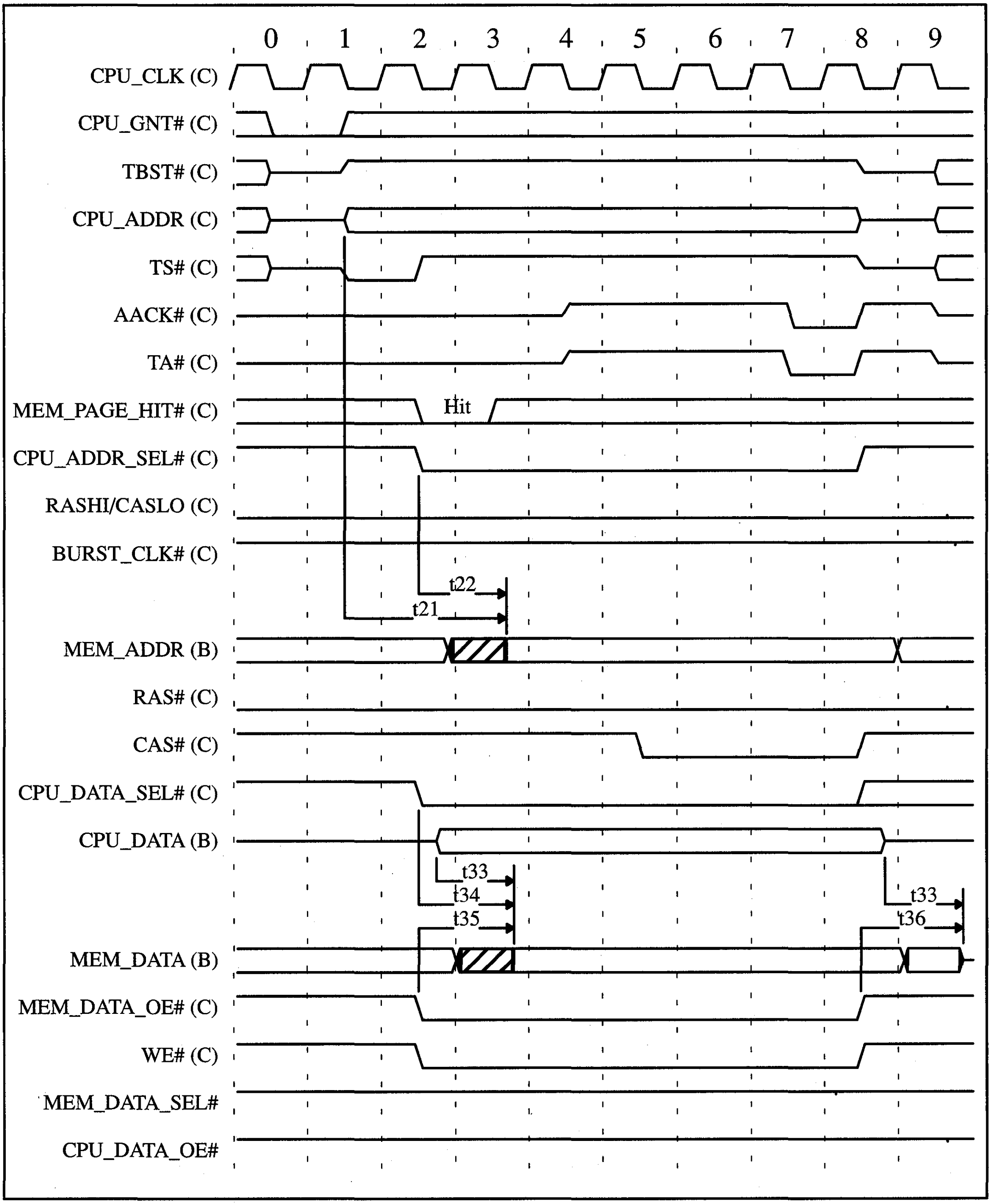


Figure 7–27. CPU To Memory Write – Single, Page Hit, XCAS=1

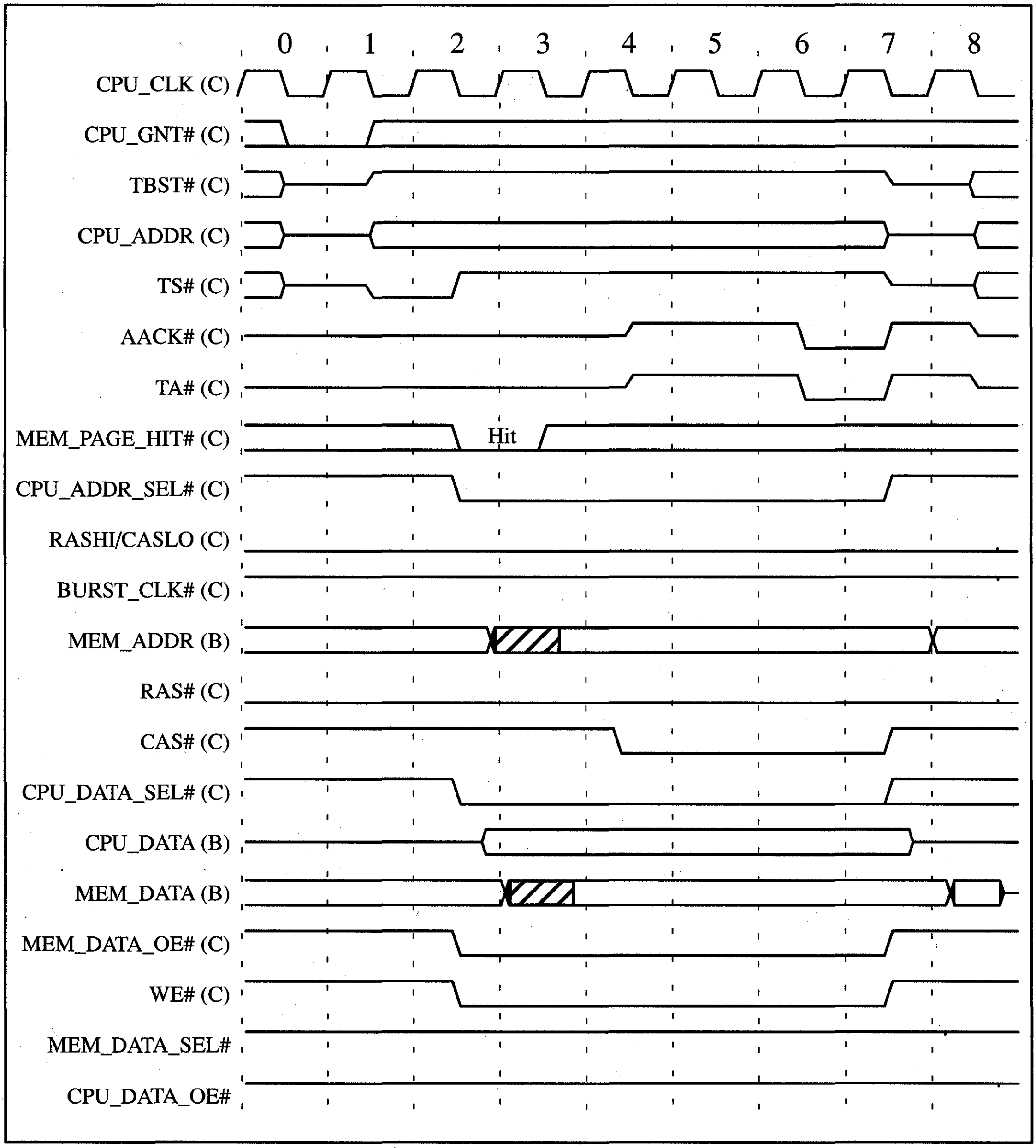


Figure 7–28. CPU To Memory Write – Single, Page Hit, XCAS=0

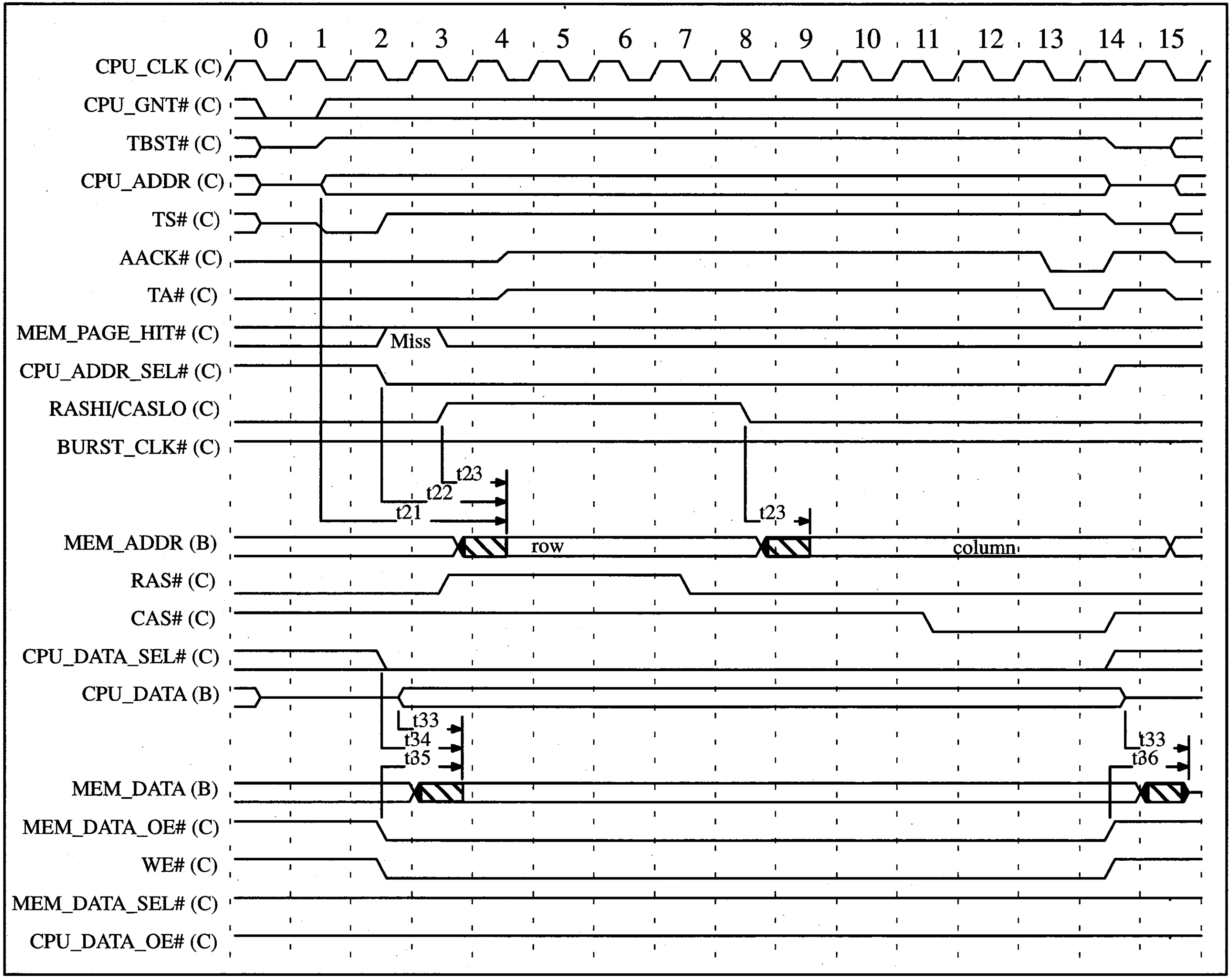


Figure 7–29. CPU To Memory Write – Single, Page Miss, XCAS=1

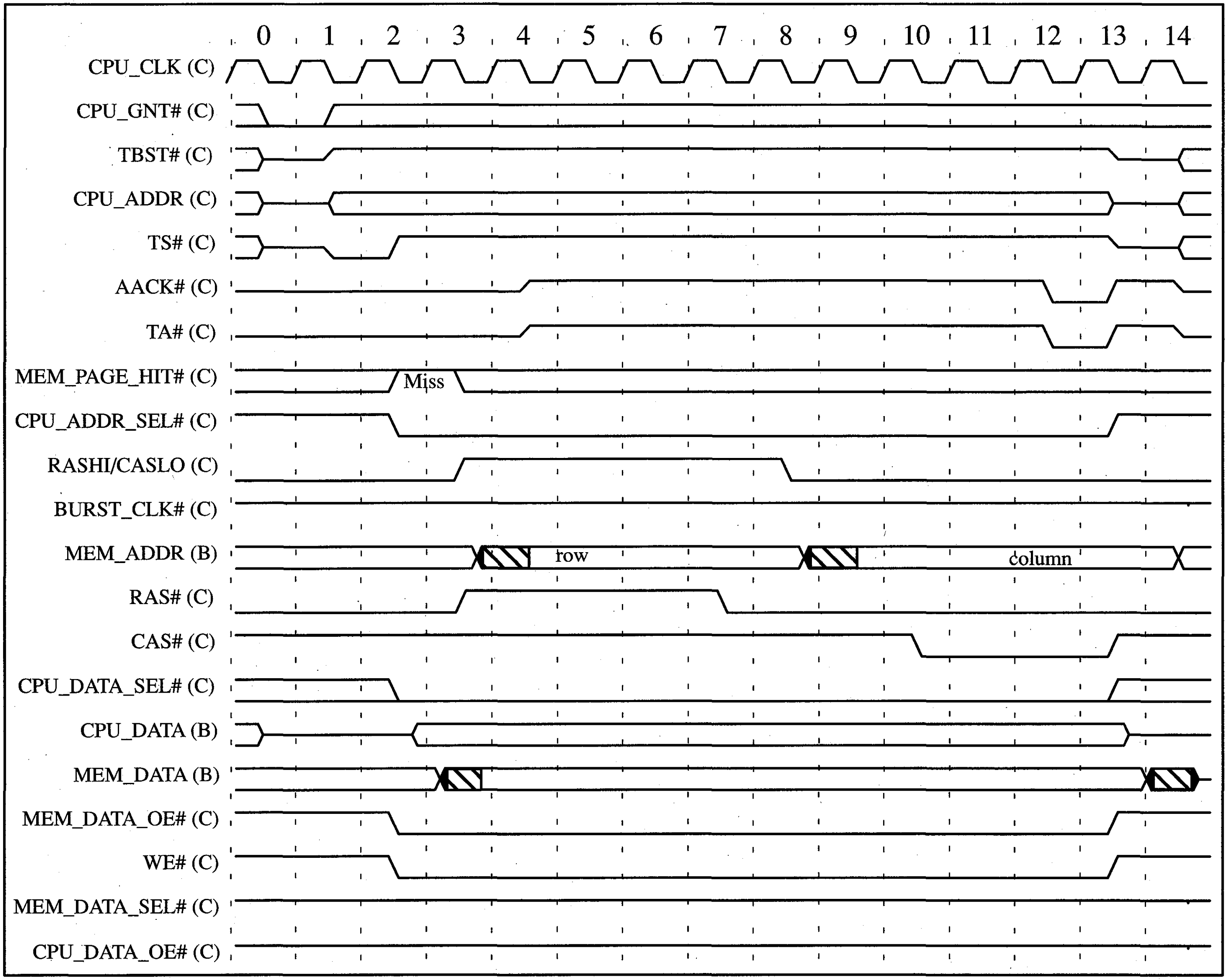


Figure 7–30. CPU To Memory Write – Single, Page Miss, XCAS=0

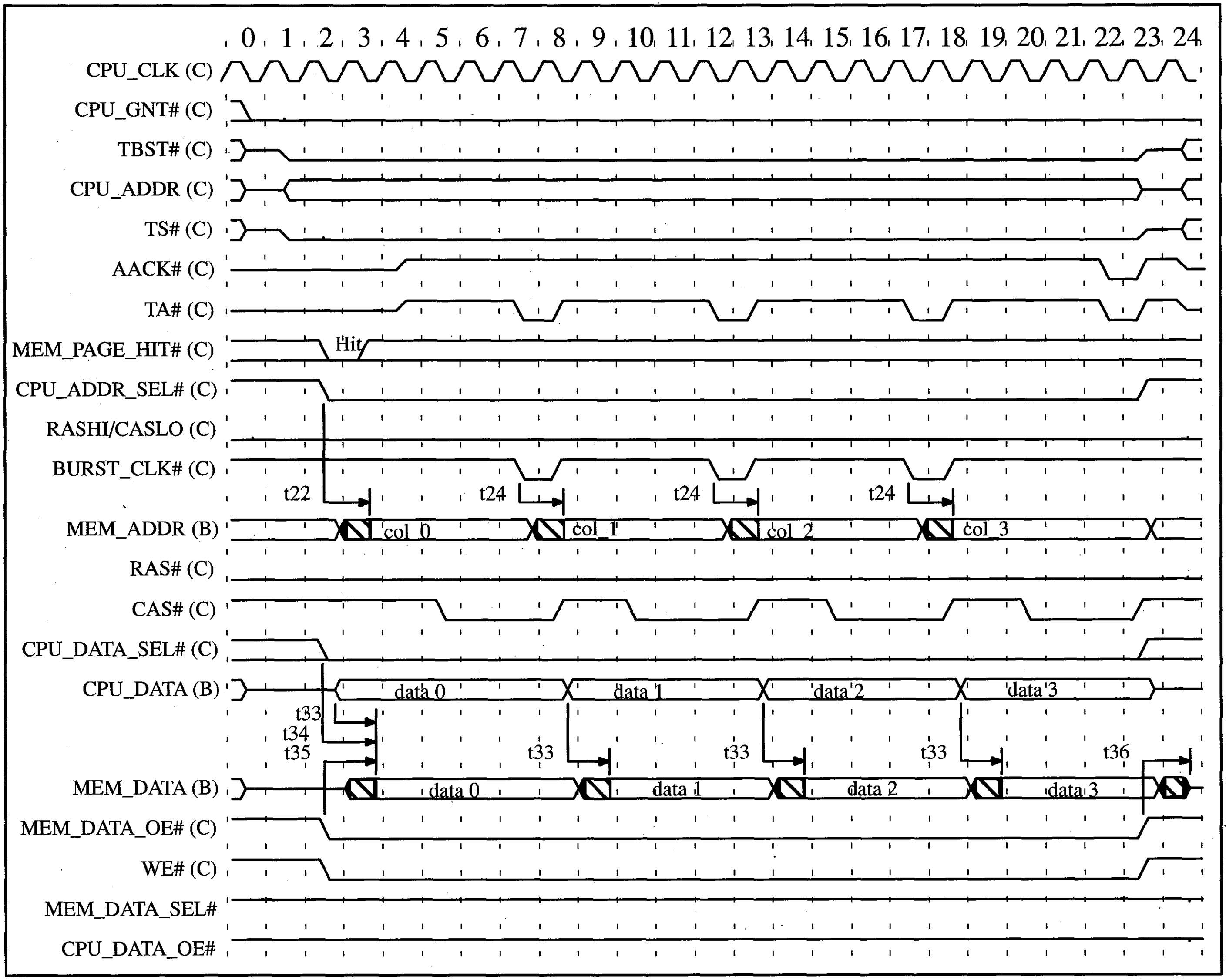


Figure 7–31. CPU To Memory Write – Burst, Page Hit, XCAS=1

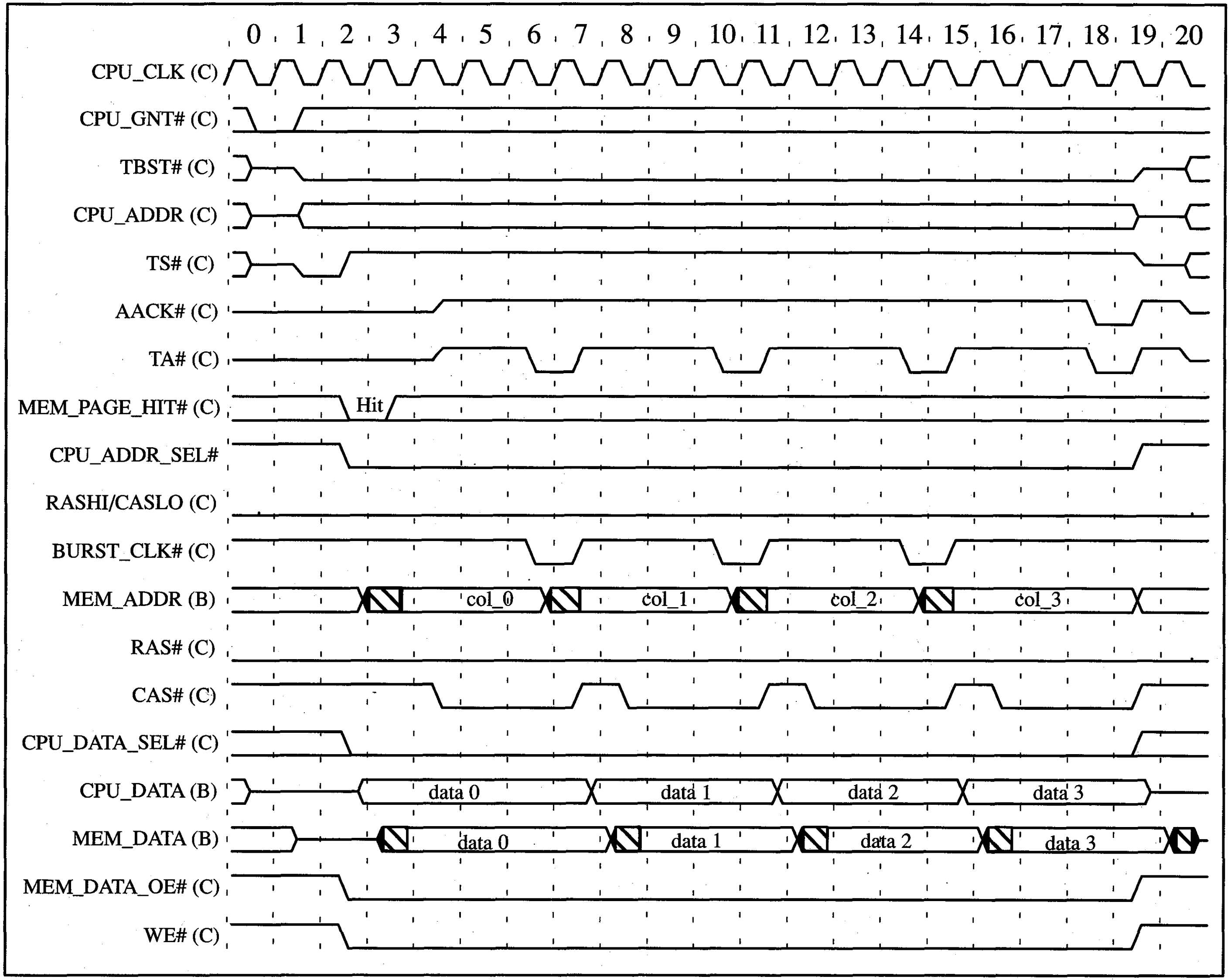


Figure 7–32. CPU To Memory Write – Burst, Page Hit, XCAS=0

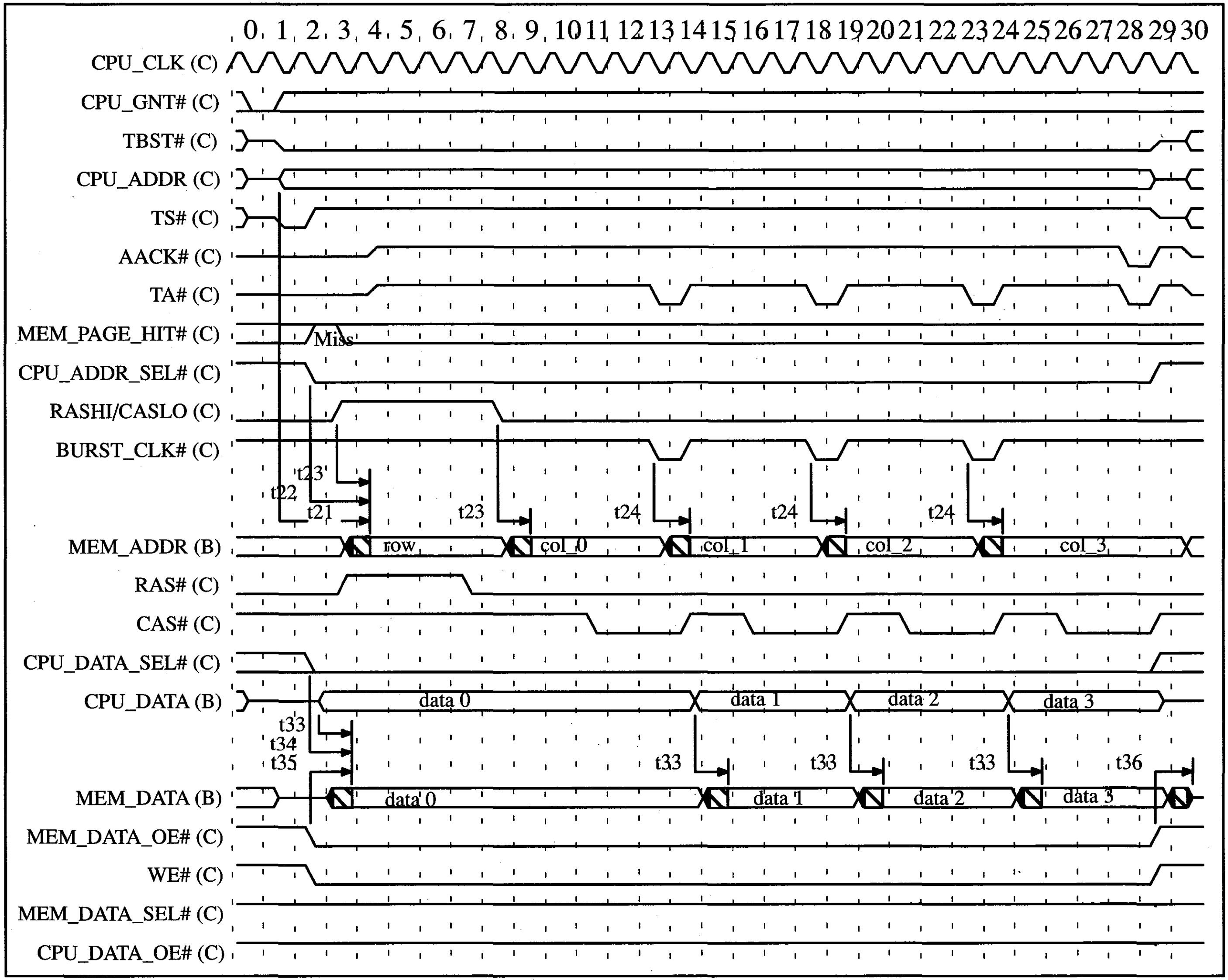


Figure 7–33. CPU To Memory Write – Burst, Page Miss, XCAS=1

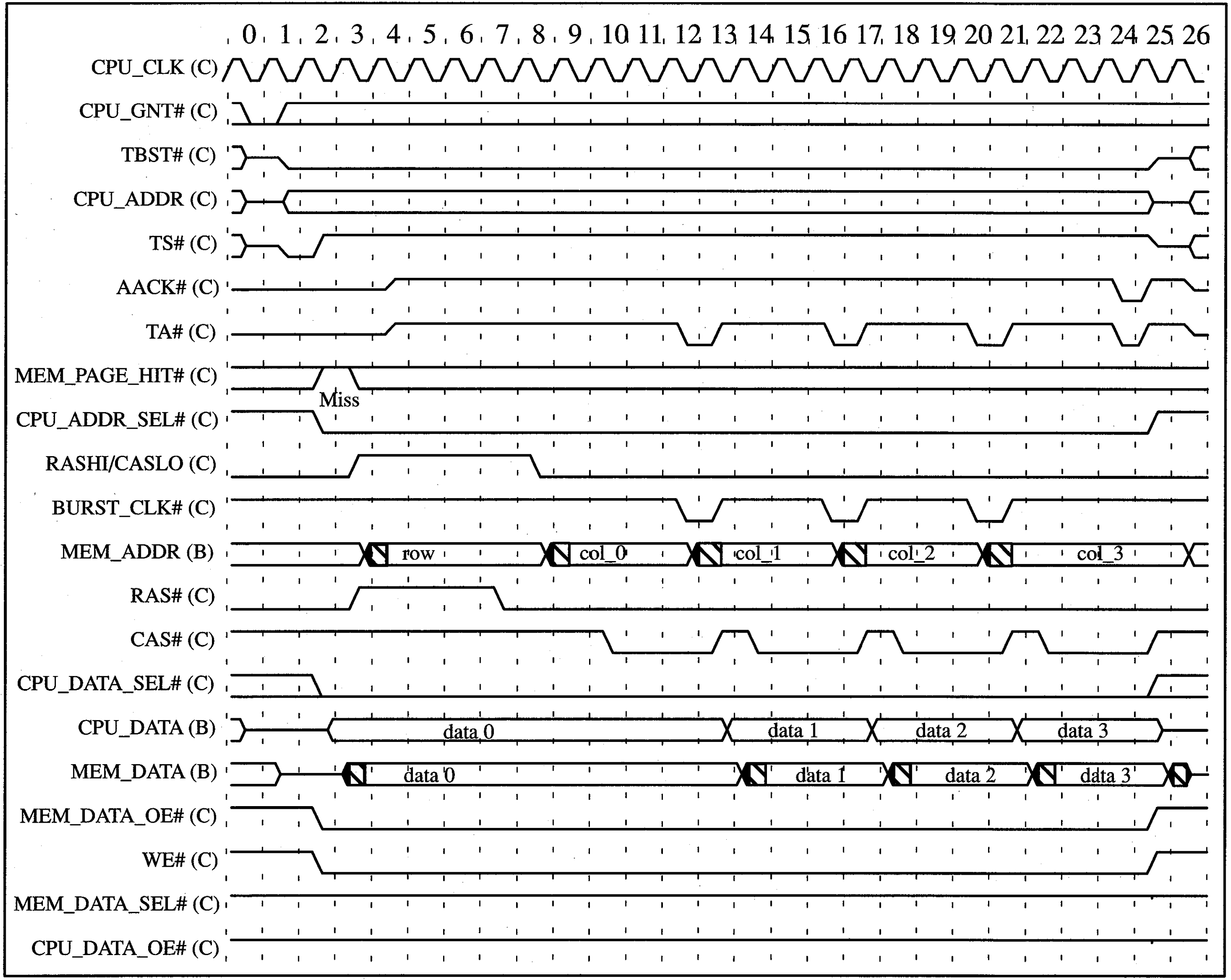


Figure 7–34. CPU To Memory Write – Burst, Page Miss, XCAS=0

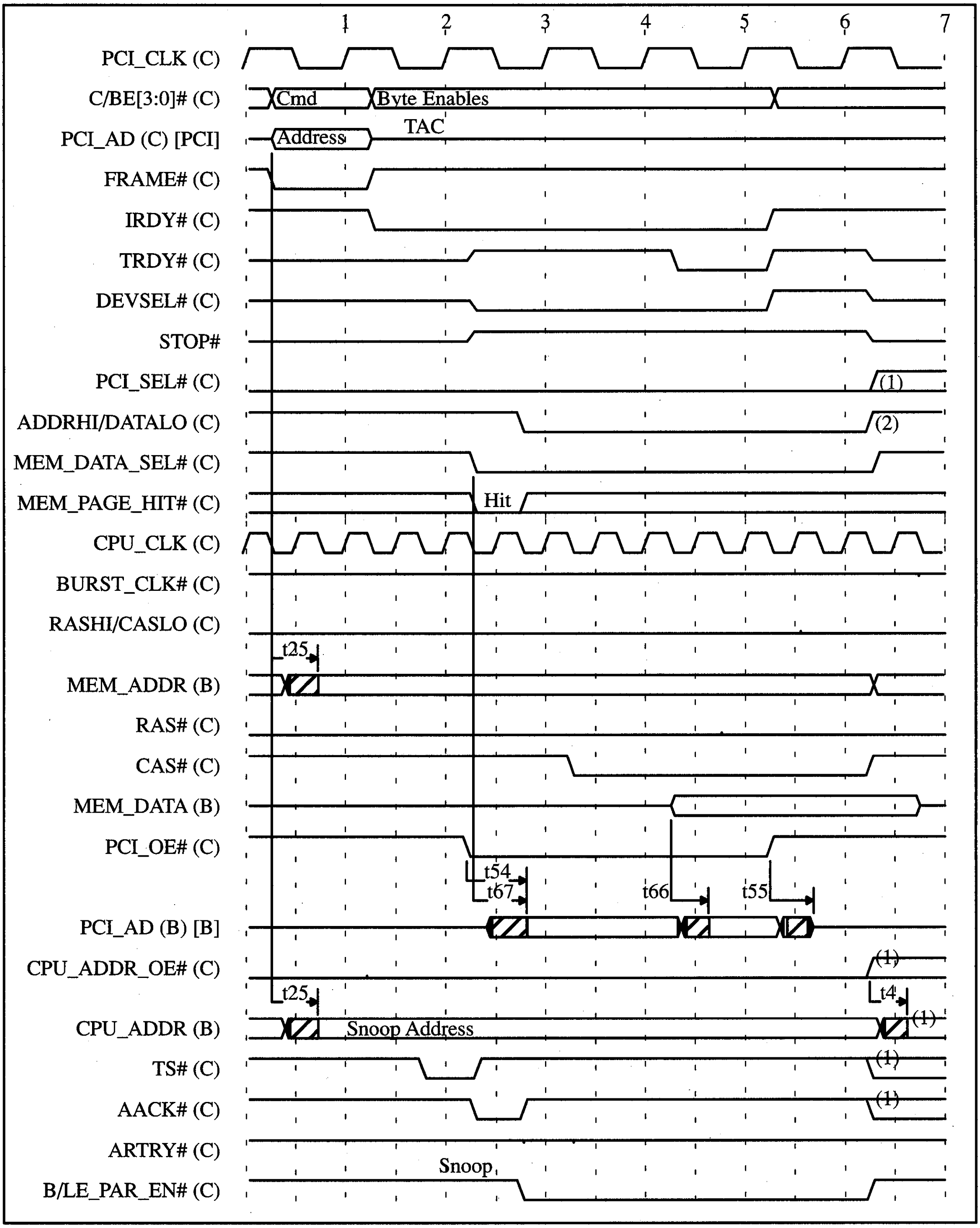


Figure 7–35. PCI To Memory Read – Single, Page Hit

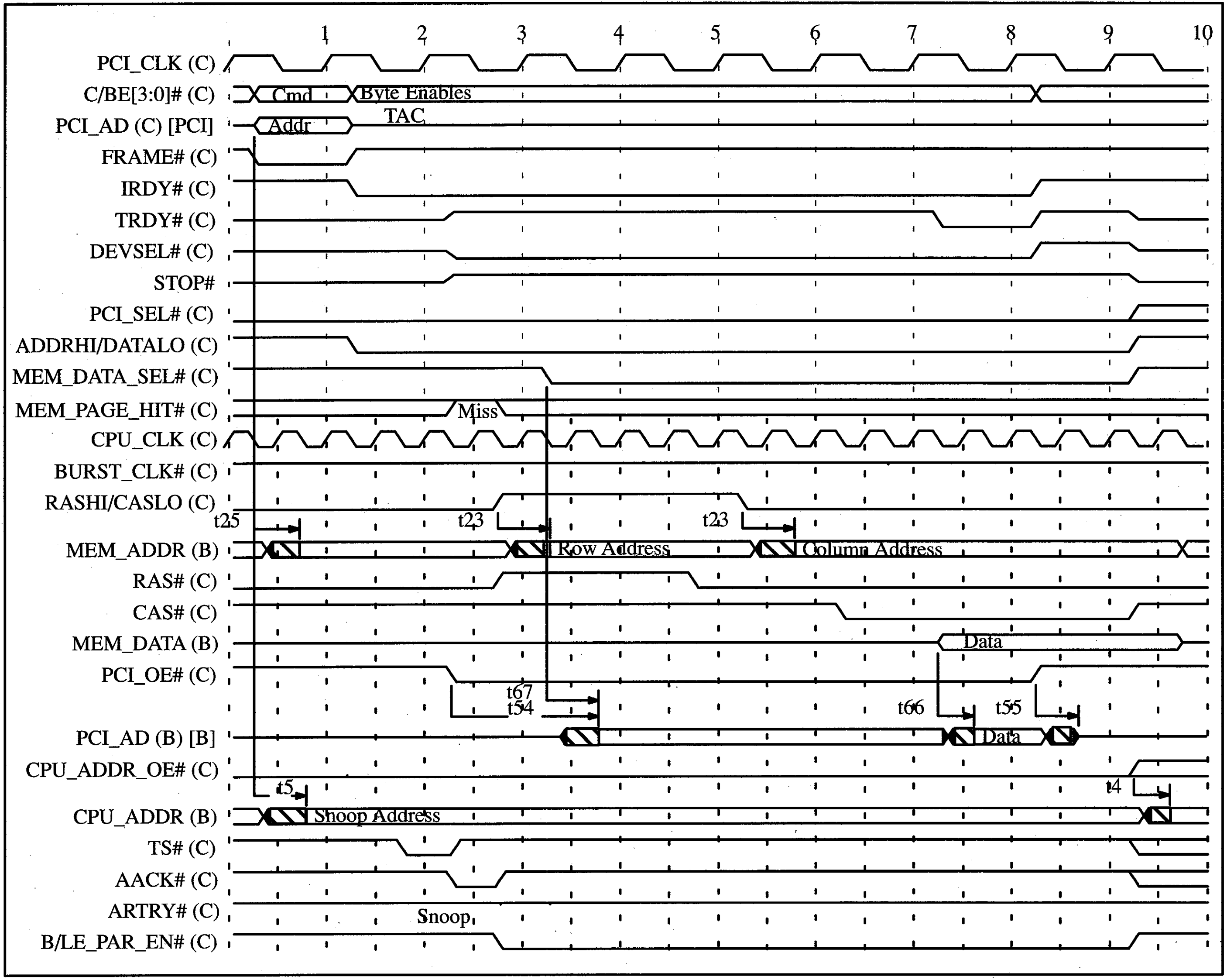


Figure 7–36. PCI To Memory Read – Single, Page Miss

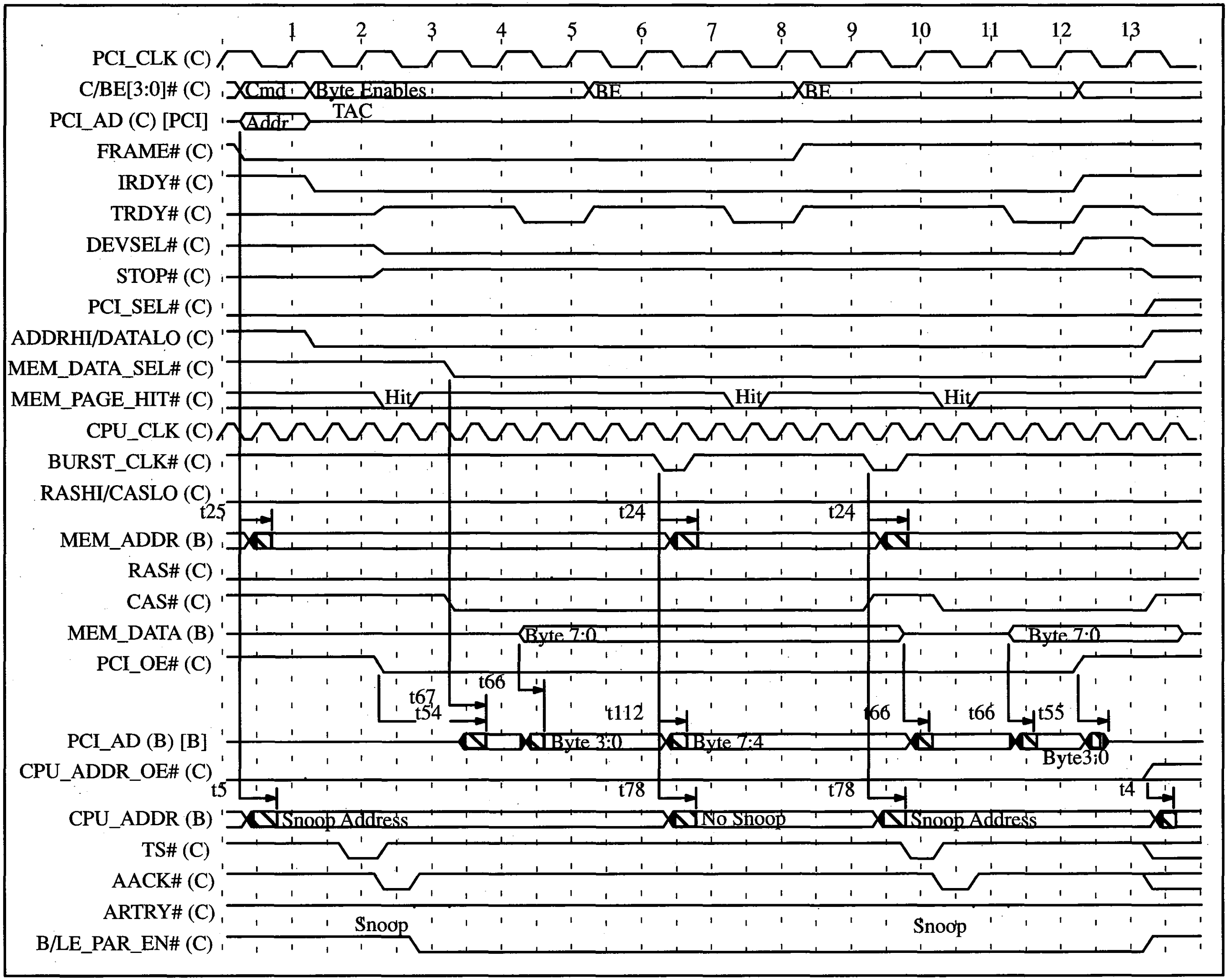


Figure 7–37. PCI To Memory Read – Burst, Page Hit

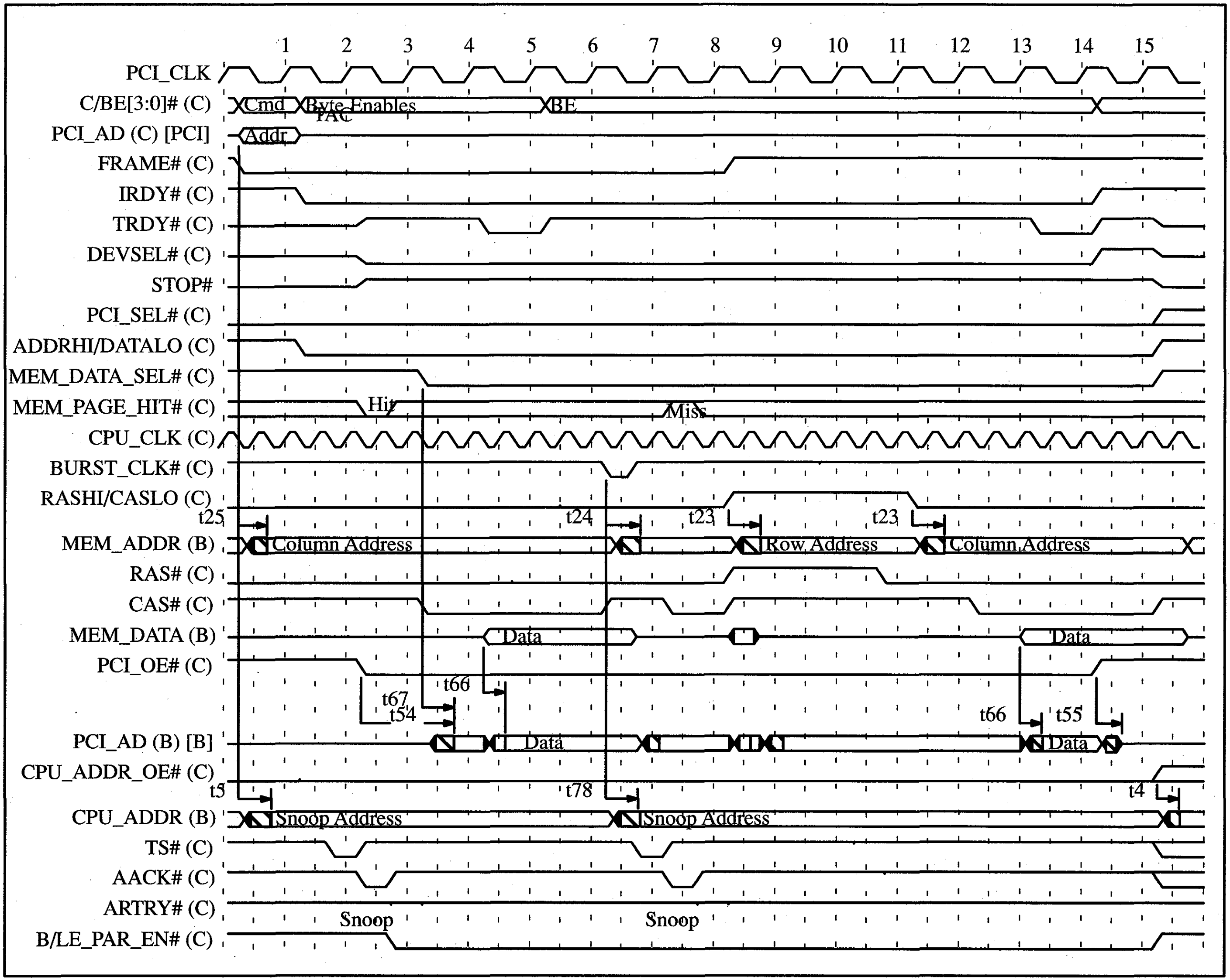
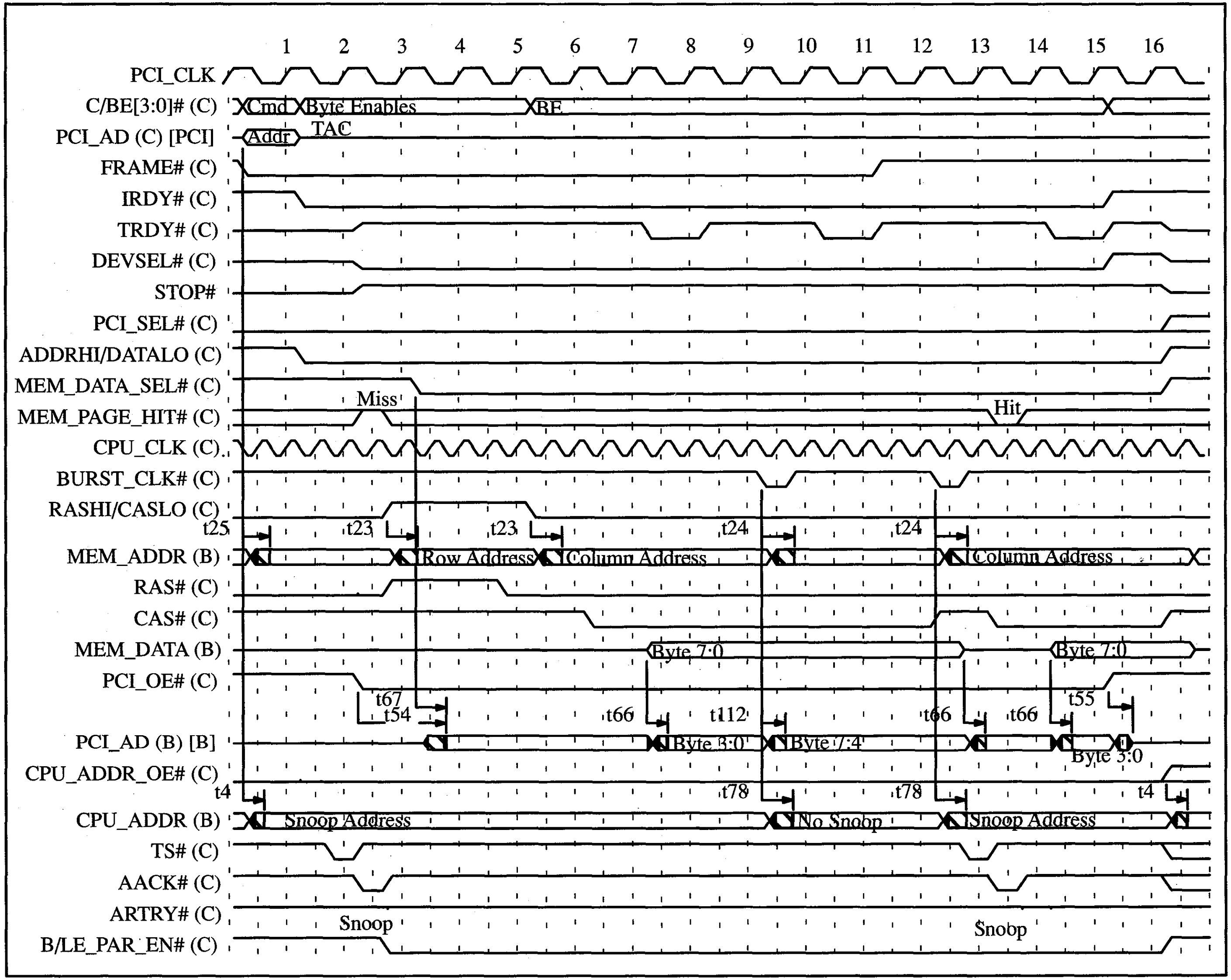


Figure 7–38. PCI To Memory Read – Burst, Page Hit Then Miss

Figure 7–39. PCI To Memory Read – Burst, Page Miss Then Hit



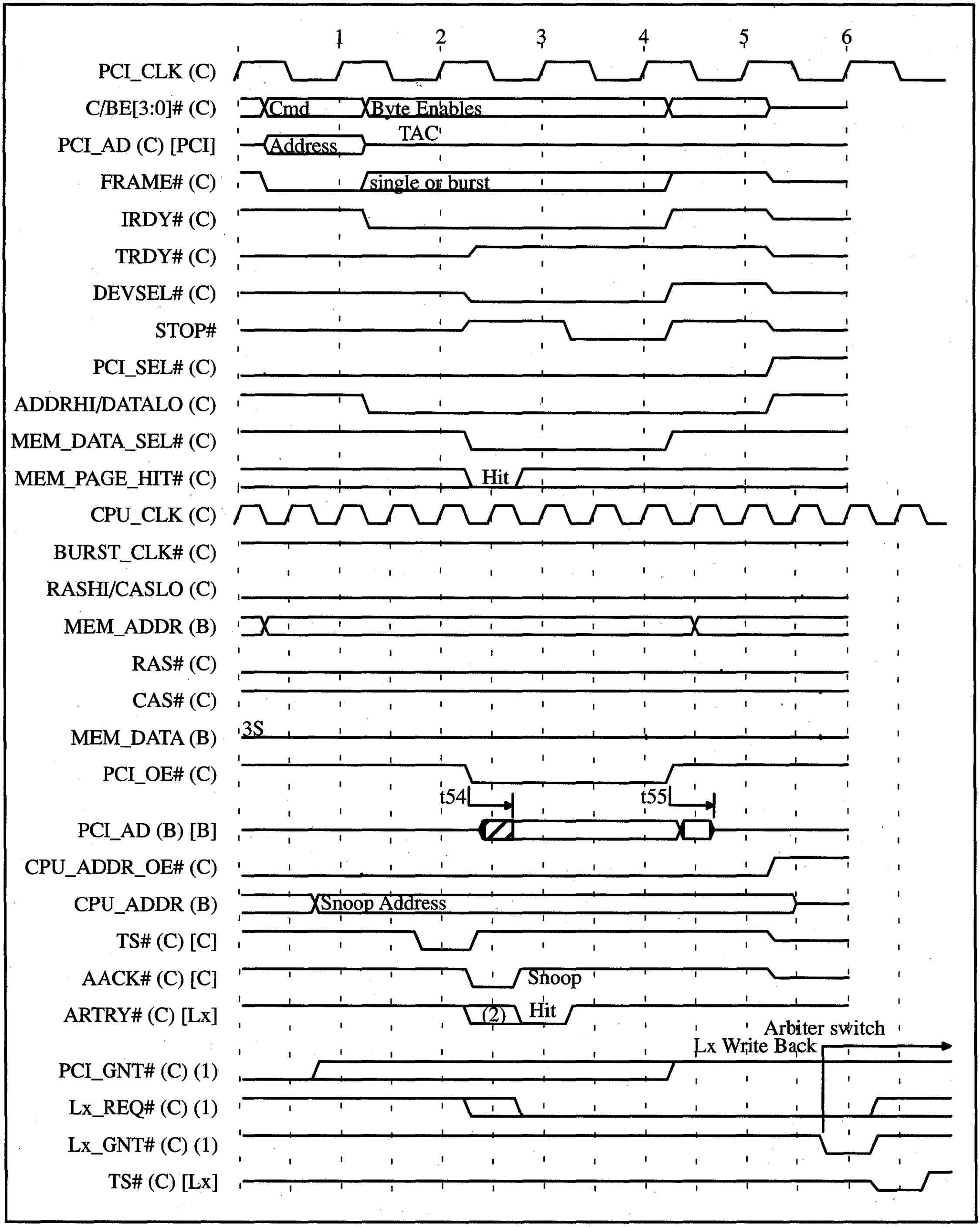


Figure 7–40. PCI To Memory Read – Page Hit, Cache Hit

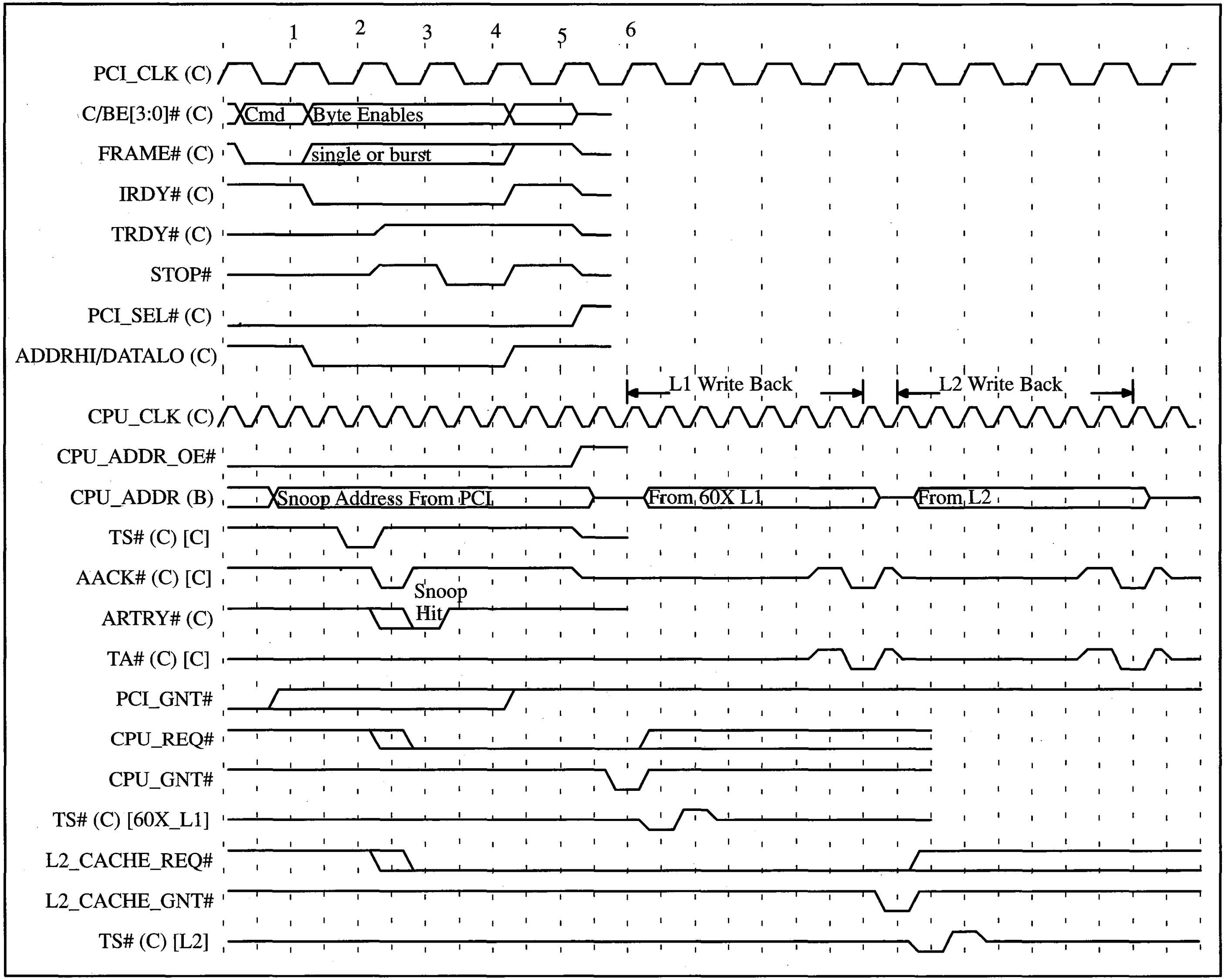


Figure 7–41. PCI To Memory – Cache Hit With Arbiter Switch

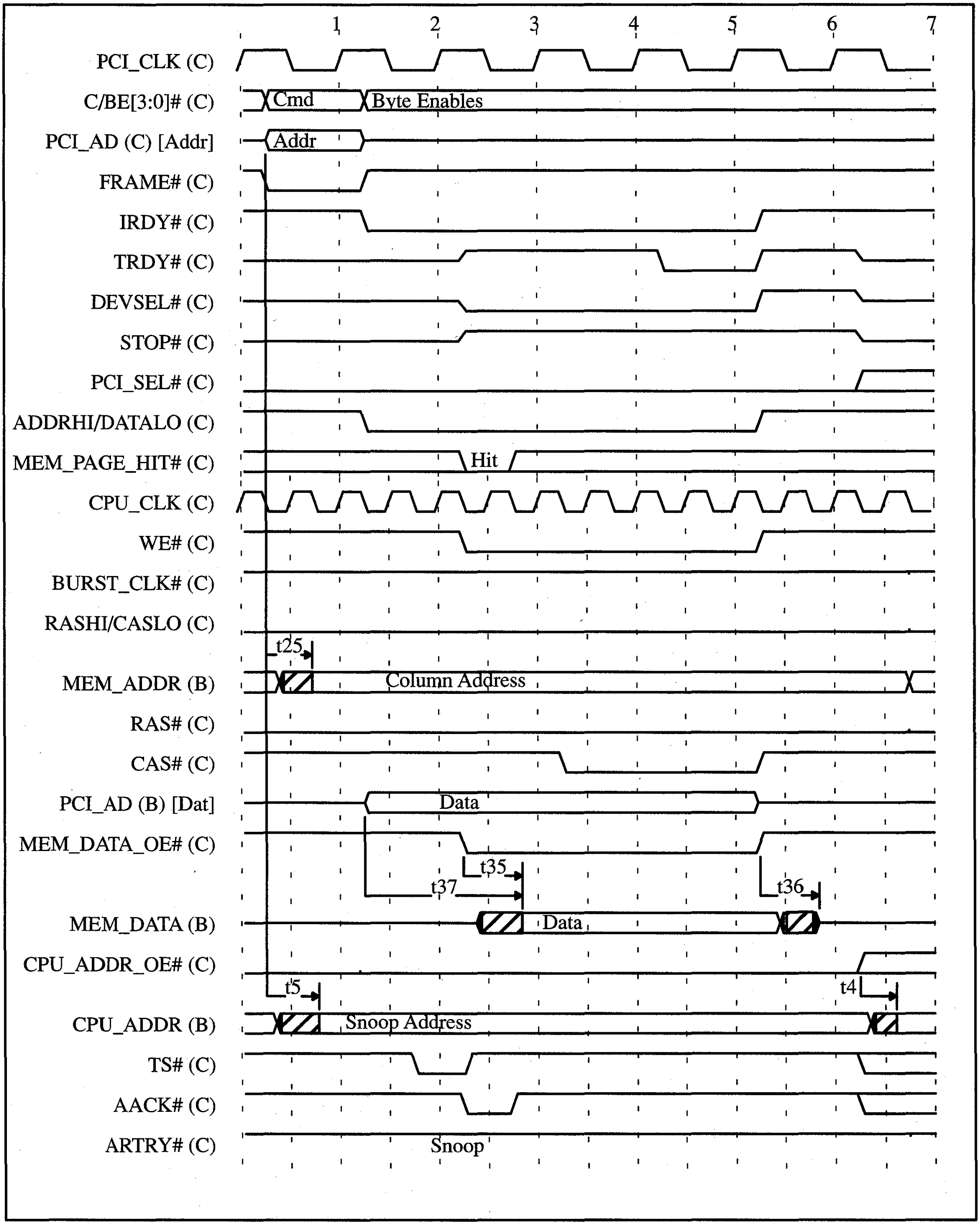


Figure 7–42. PCI To Memory Write – Single, Page Hit

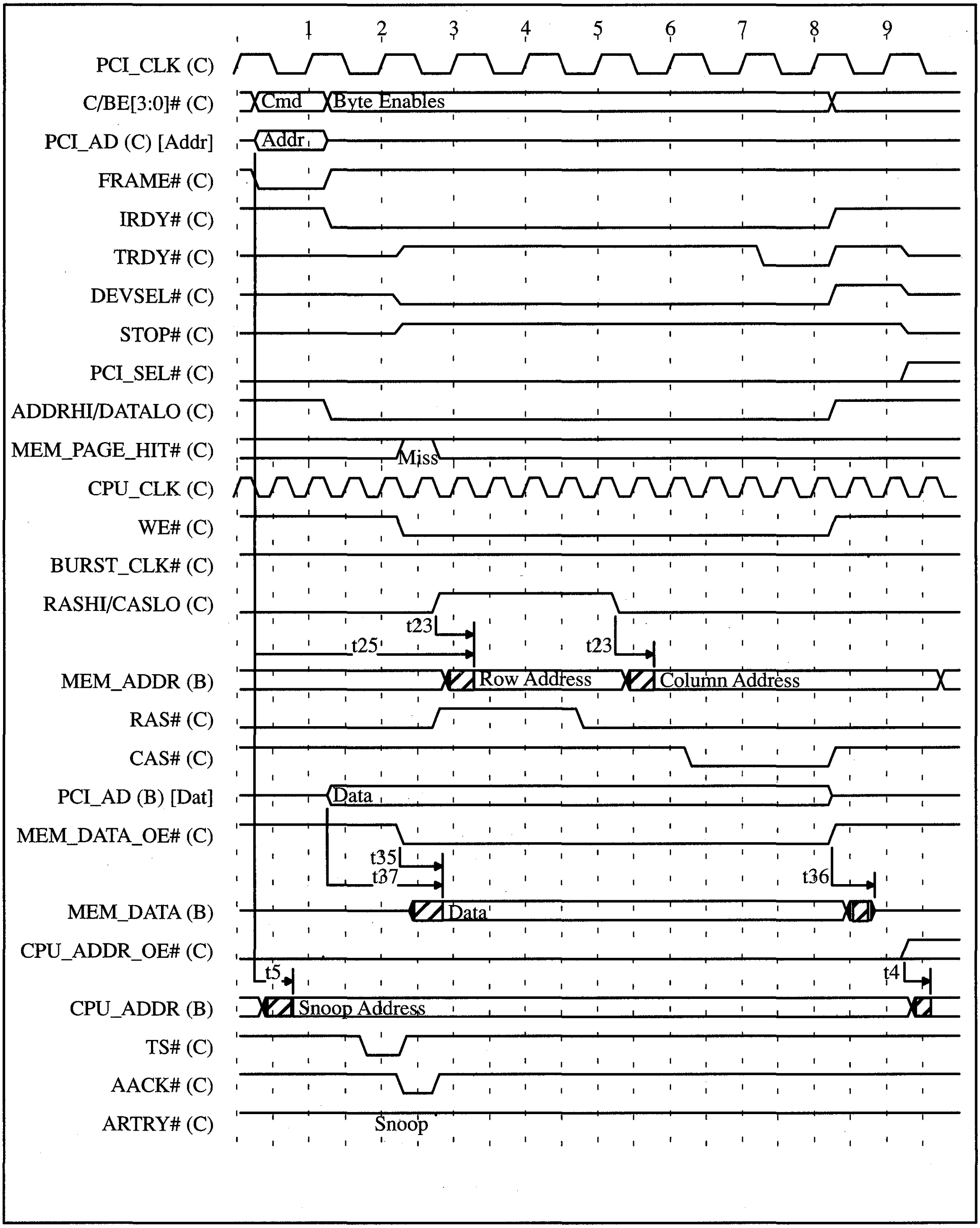


Figure 7–43. PCI To Memory Write – Single, Page Miss

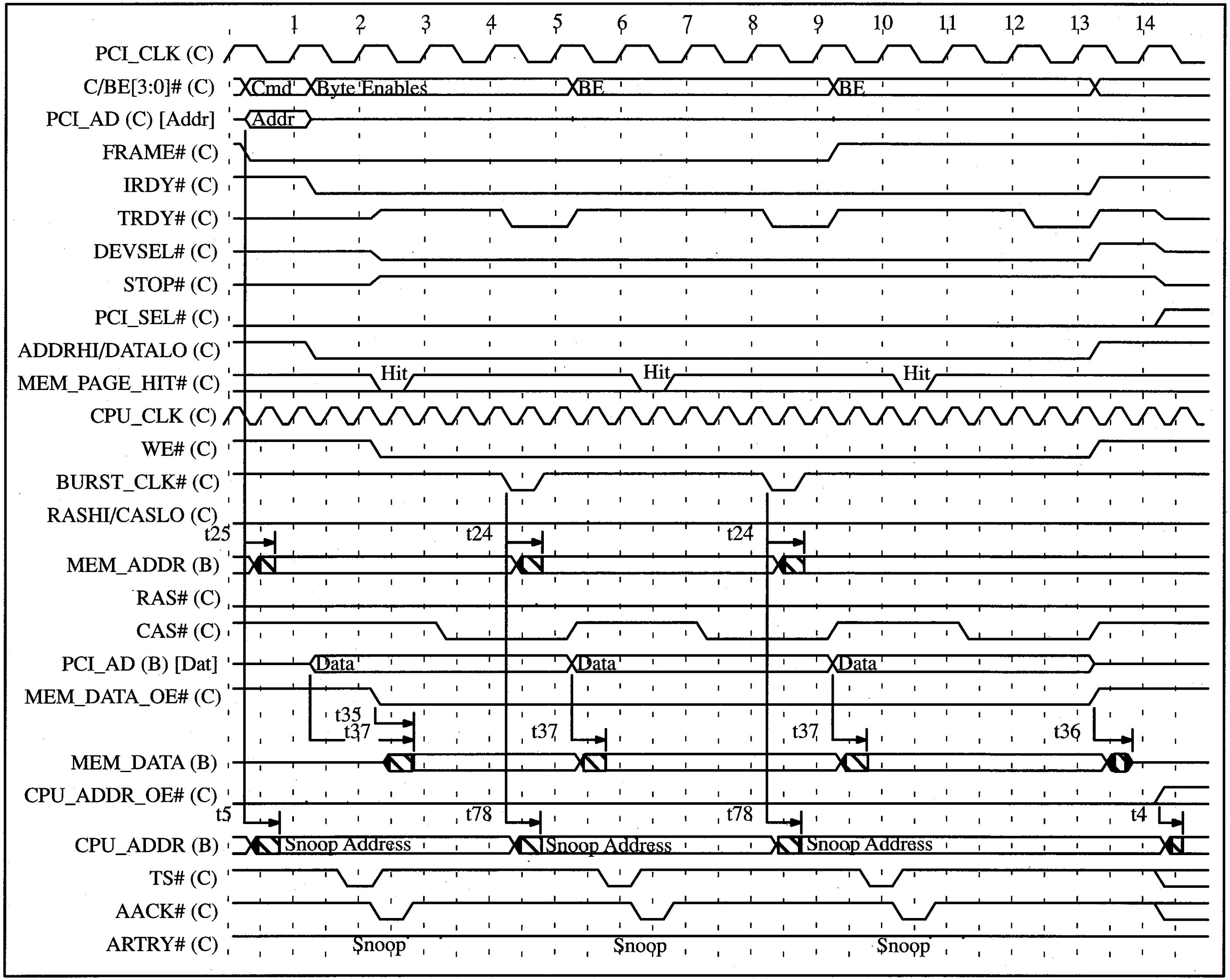


Figure 7–44. PCI To Memory Write – Burst, Page Hit

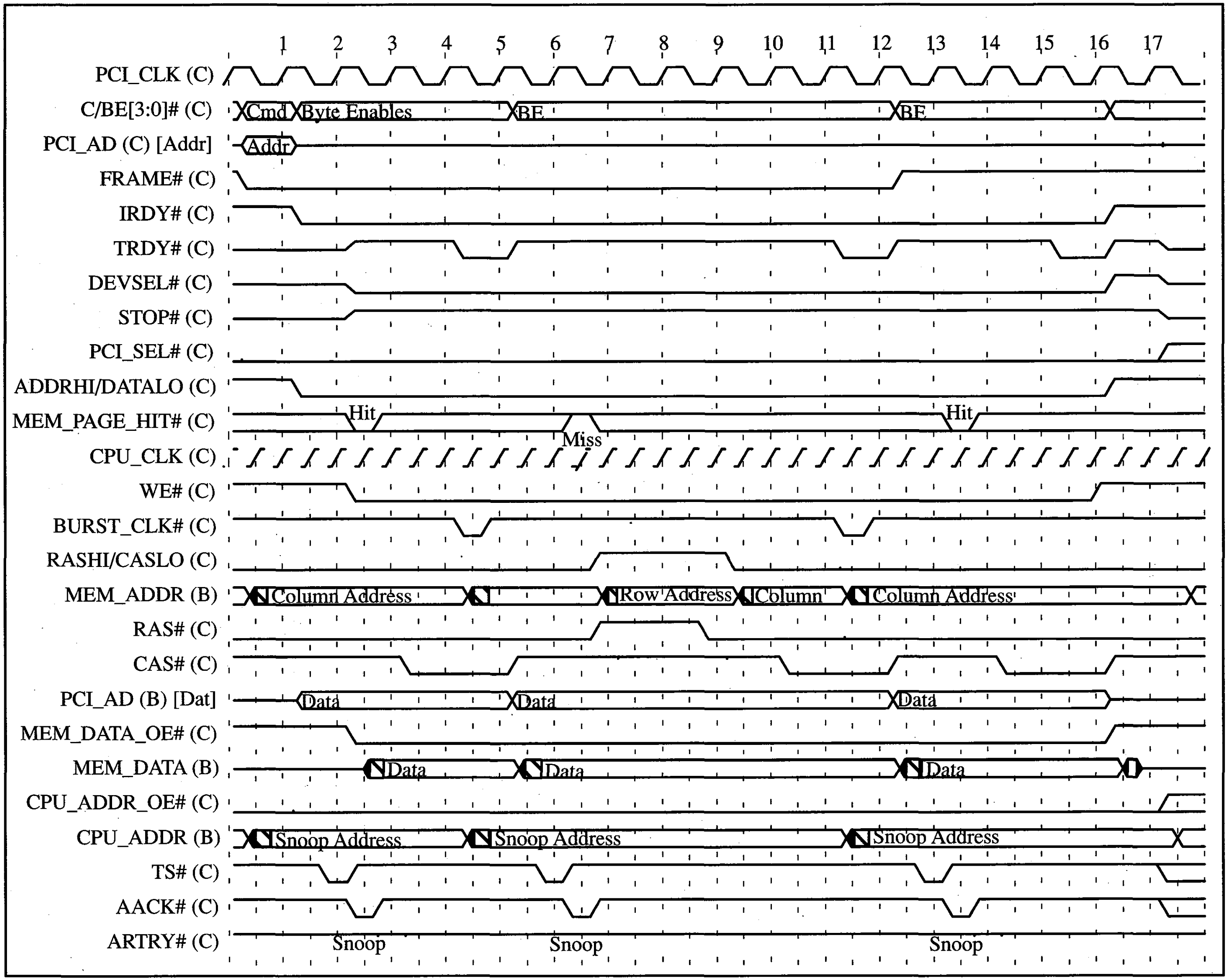


Figure 7–45. PCI To Memory Write – Burst, Page Hit Then Miss

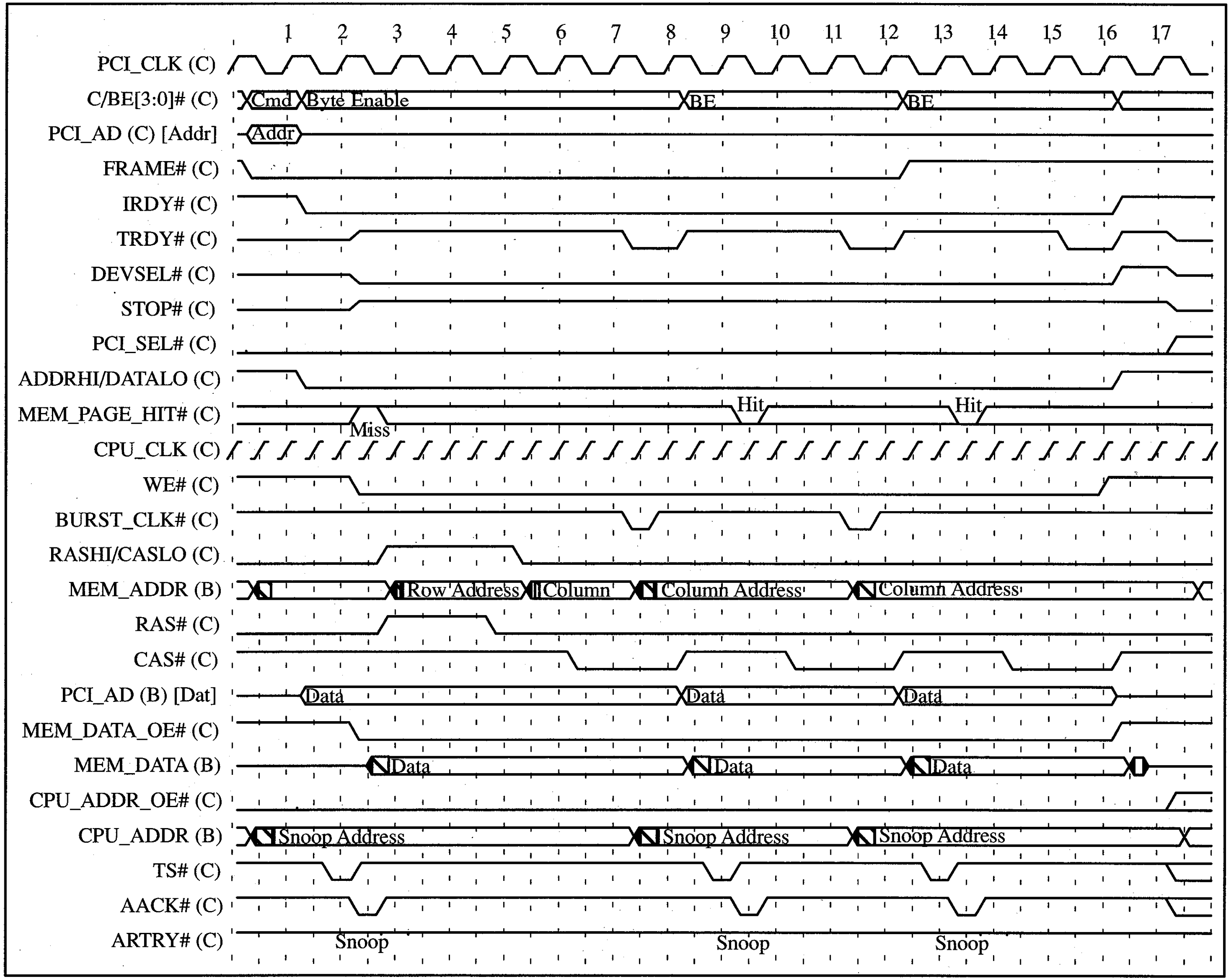


Figure 7–46. PCI To Memory Write – Burst, Page Miss Then Hit

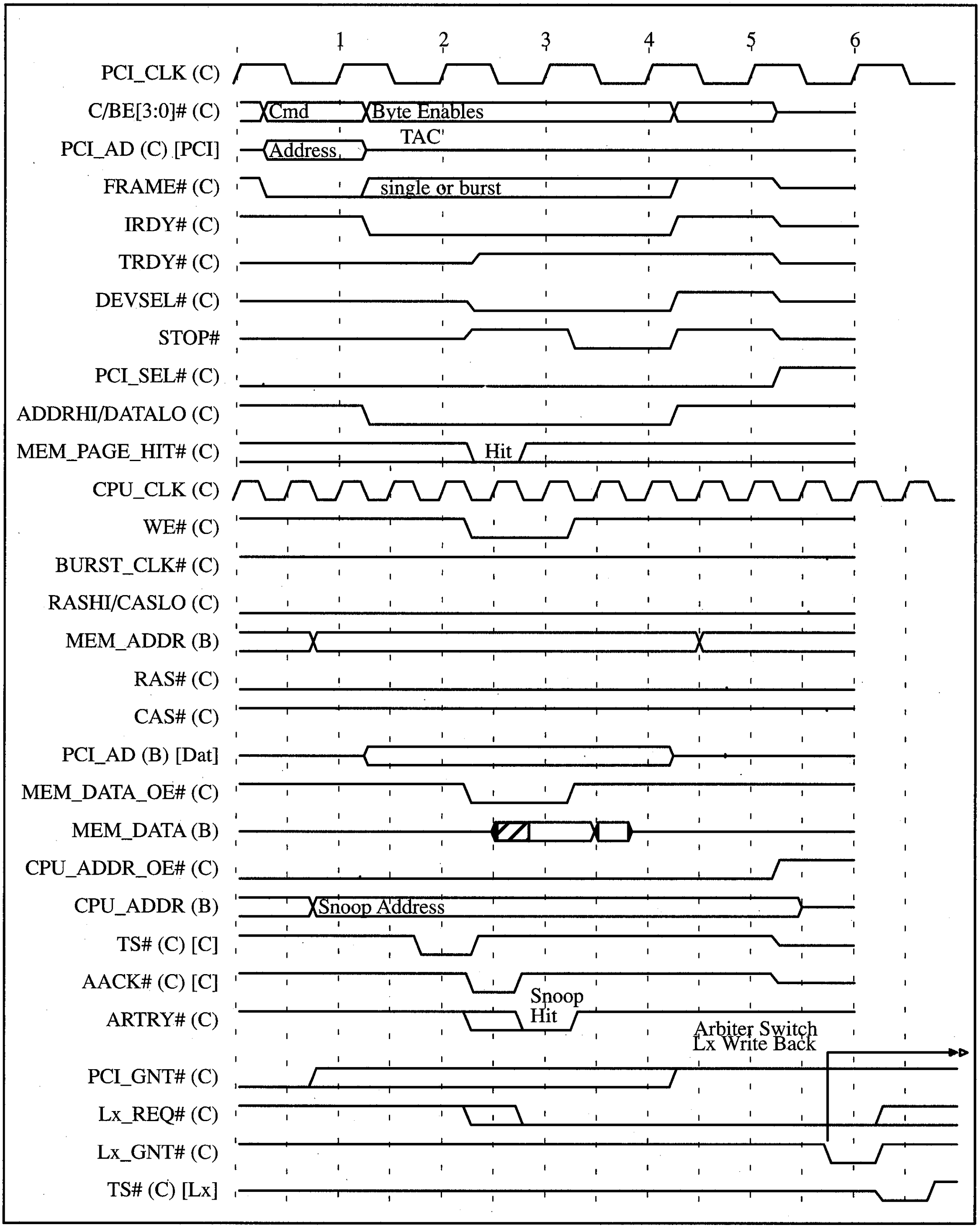


Figure 7–47. PCI To Memory Write – Page Hit, Cache Hit

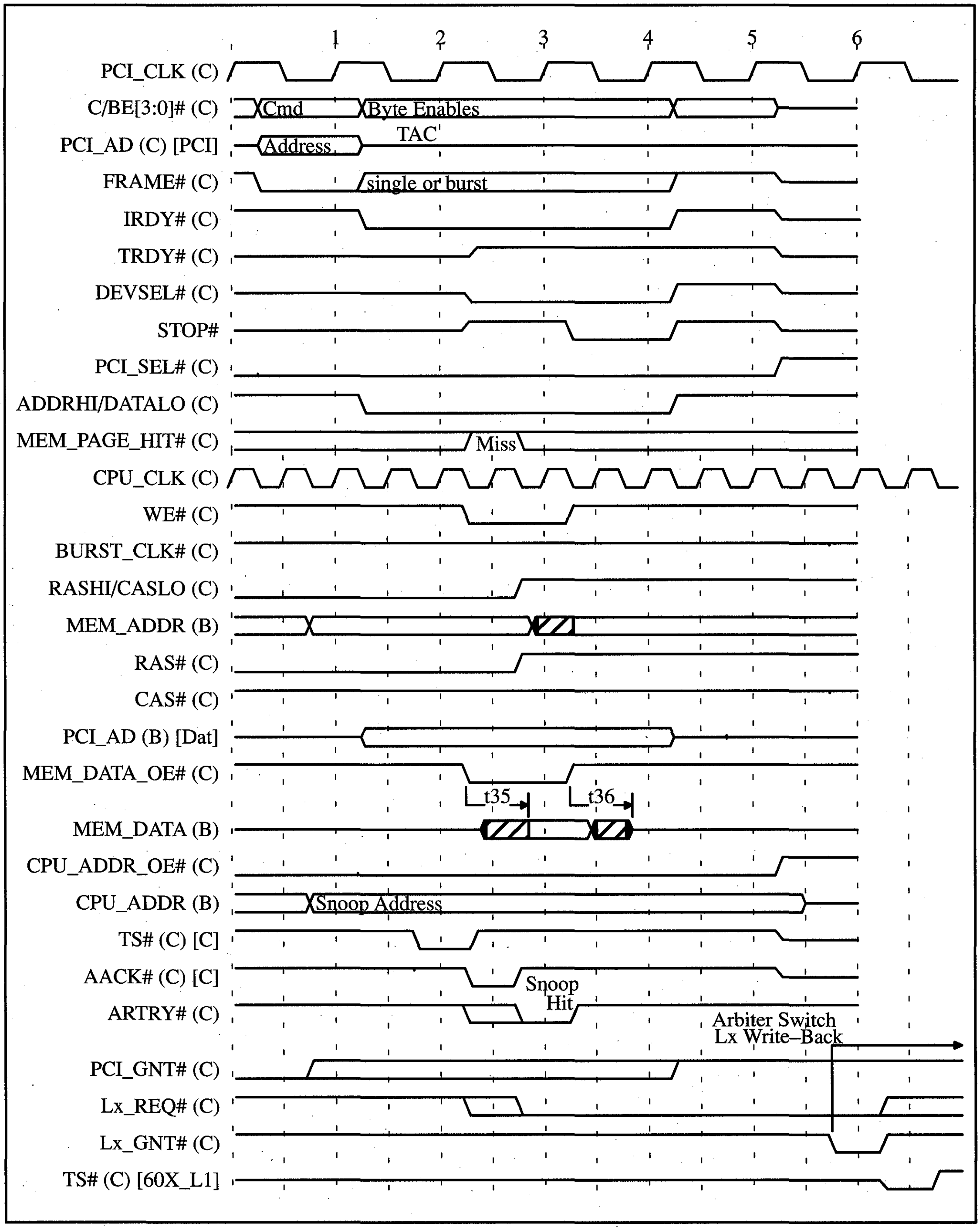


Figure 7–48. PCI To Memory Write – Page Miss, Cache Hit

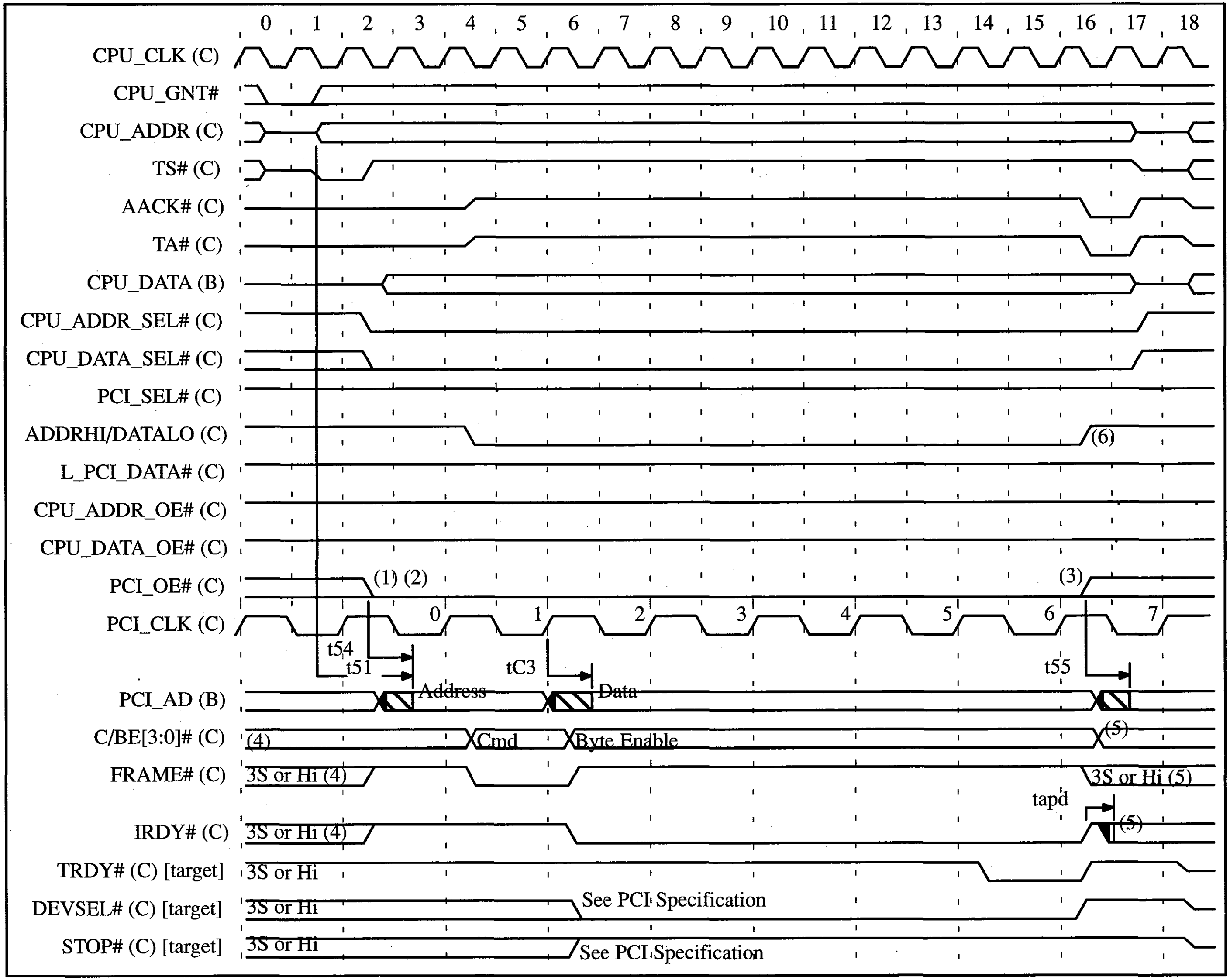


Figure 7–49. CPU To PCI Write – XADIO=0

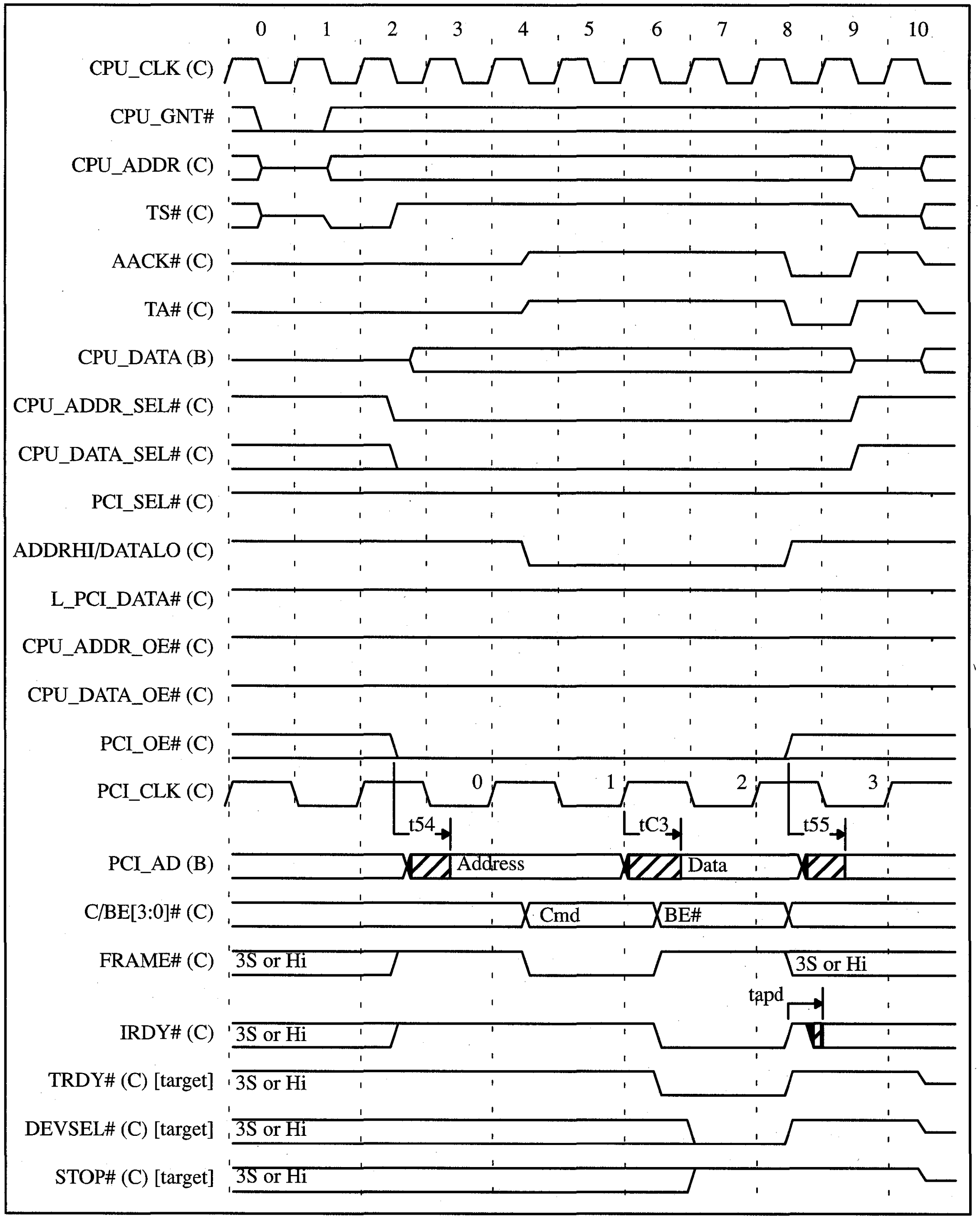


Figure 7–50. CPU To PCI Write – XADIO=0, Fast PCI Target Response

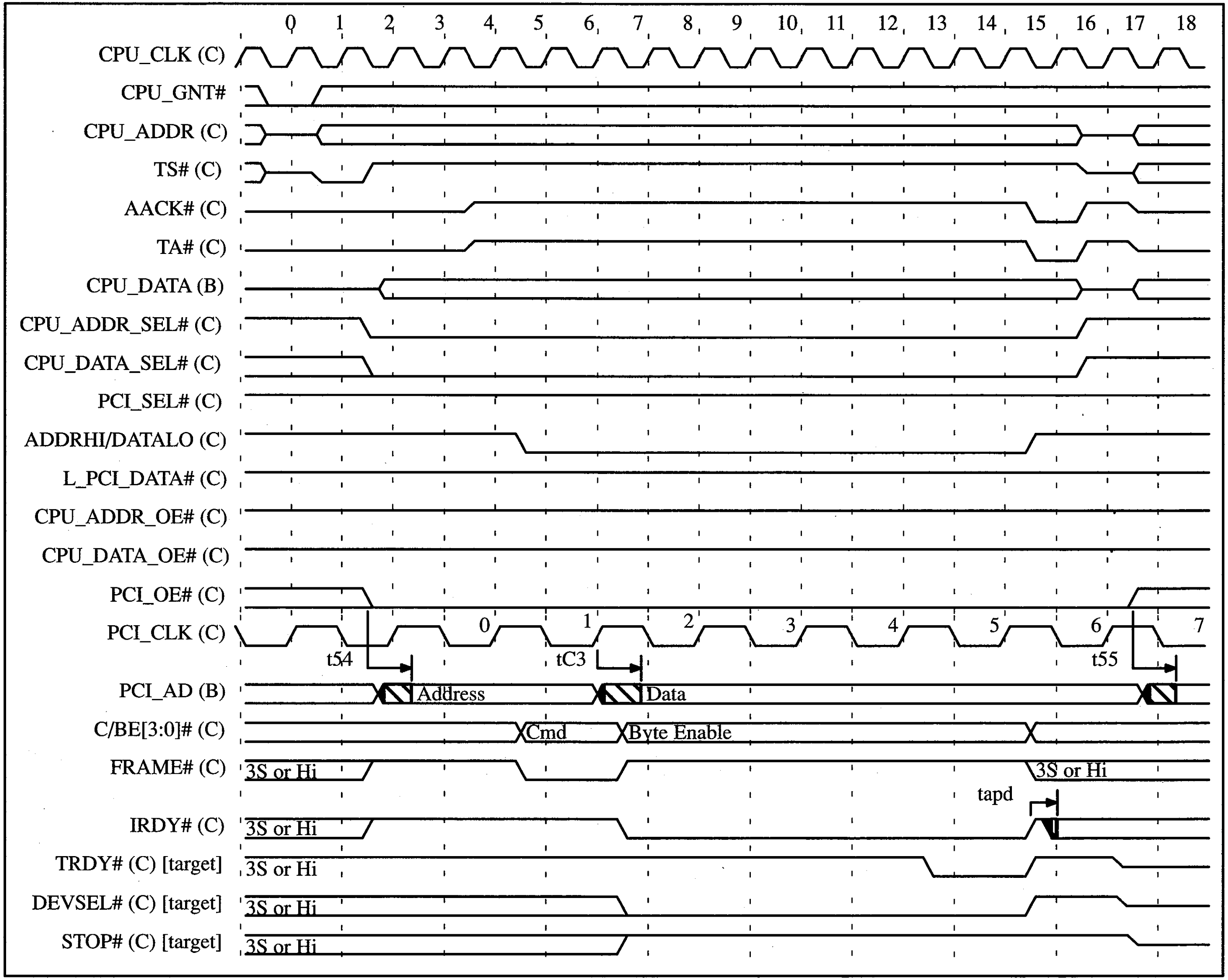


Figure 7–51. CPU To PCI Write – XADIO=0, Offbeat TS#

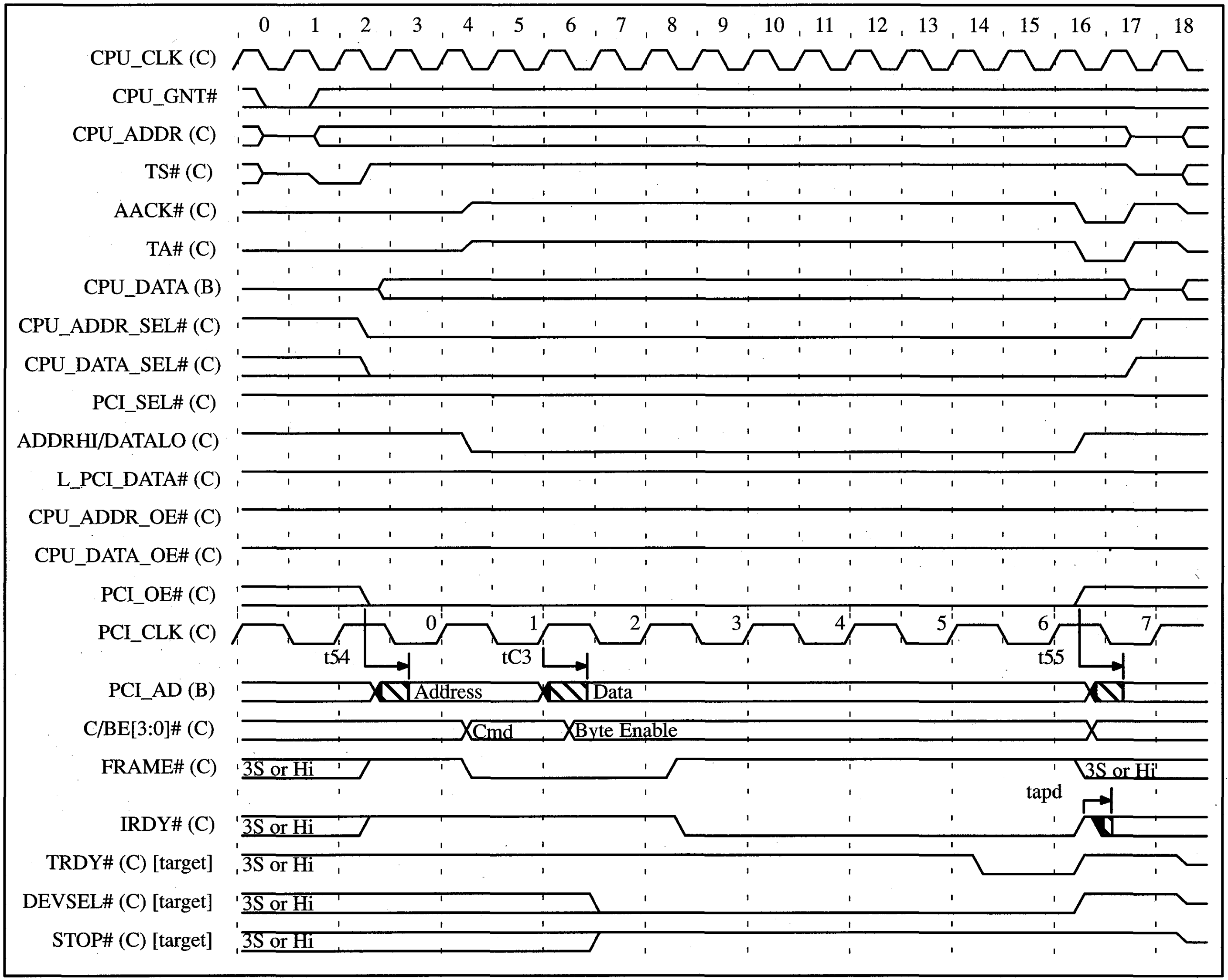


Figure 7–52. CPU To PCI Write – XADIO=1

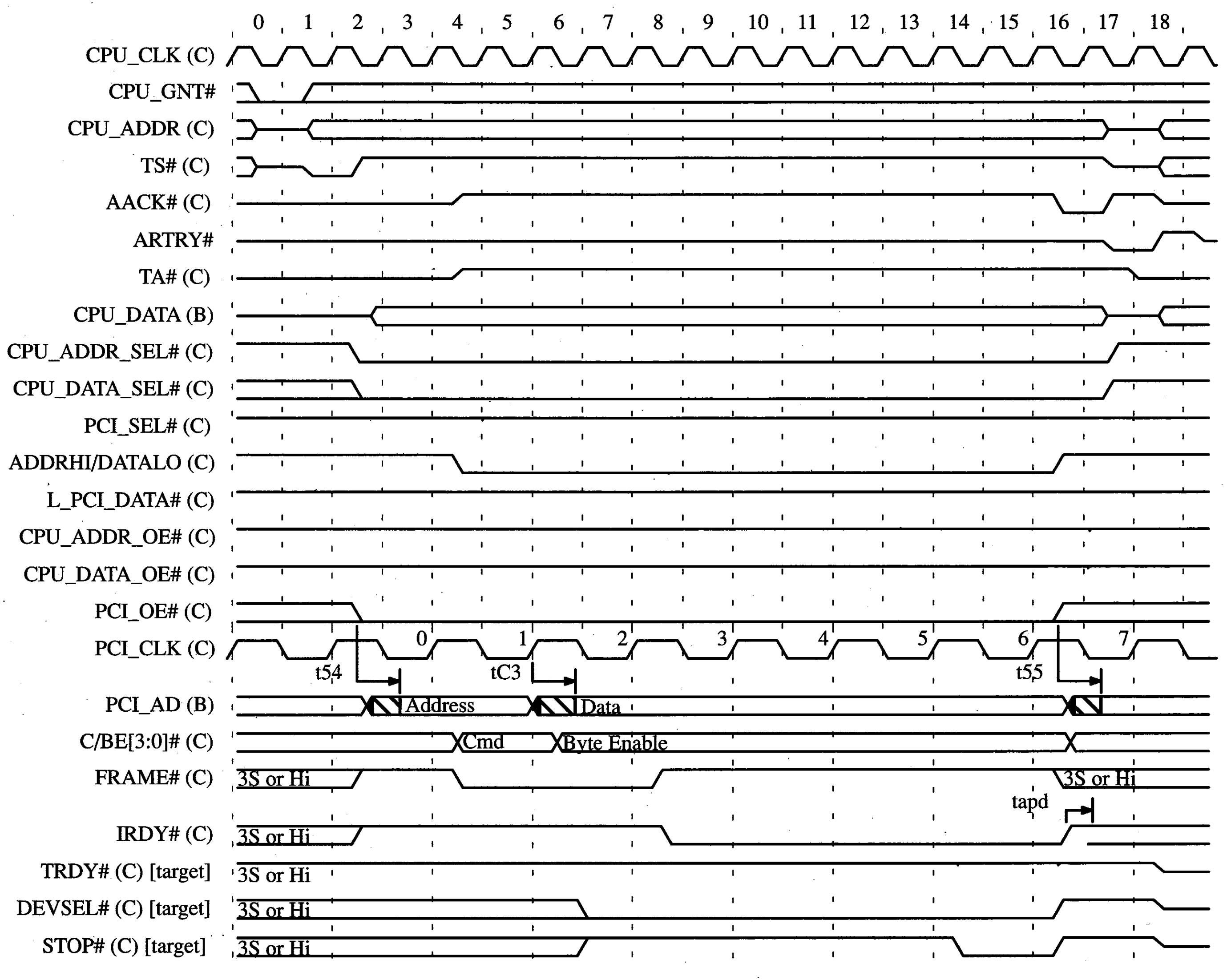


Figure 7–53. CPU To PCI Write – XADIO=1, Target Retry

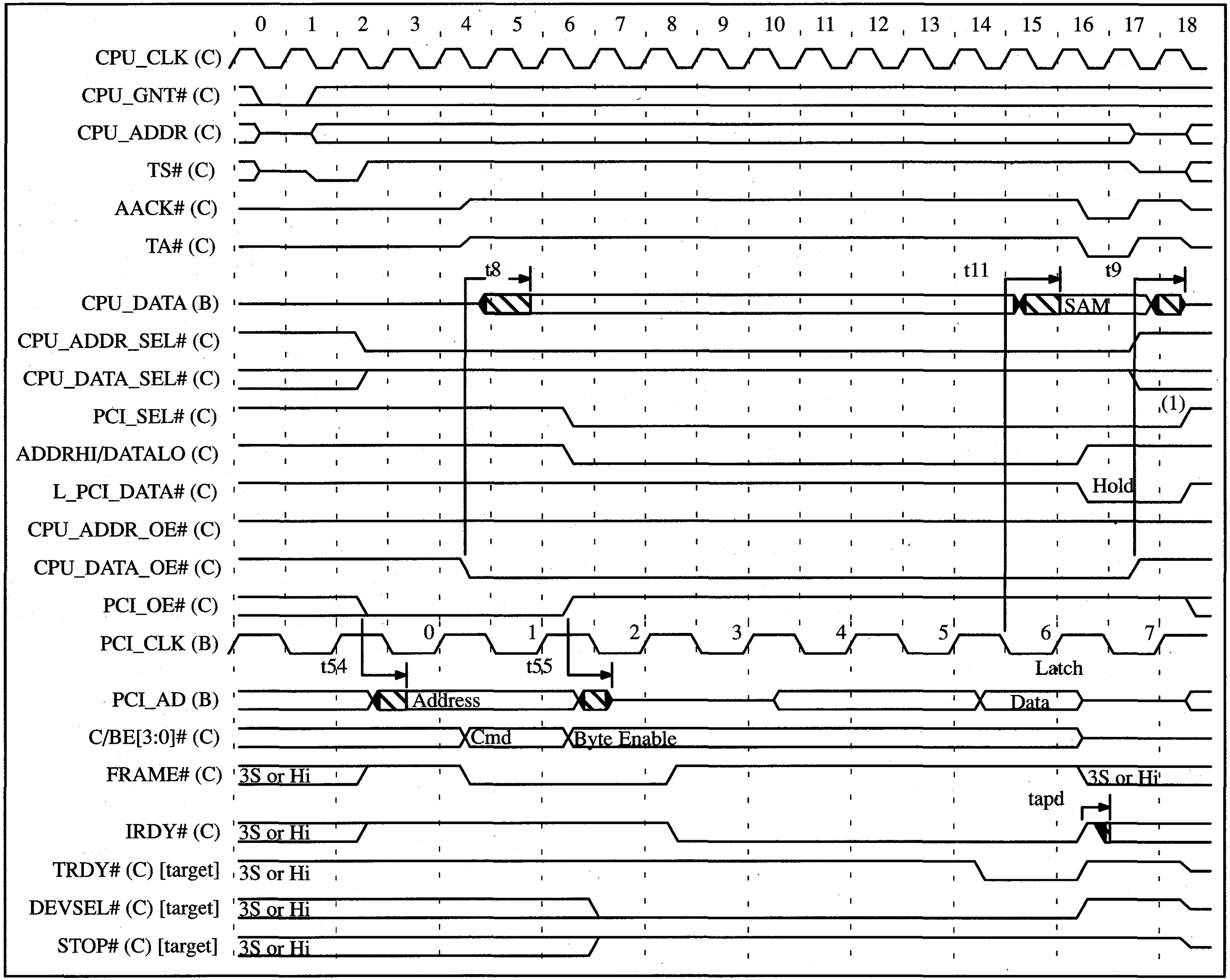


Figure 7–54. CPU To PCI Read

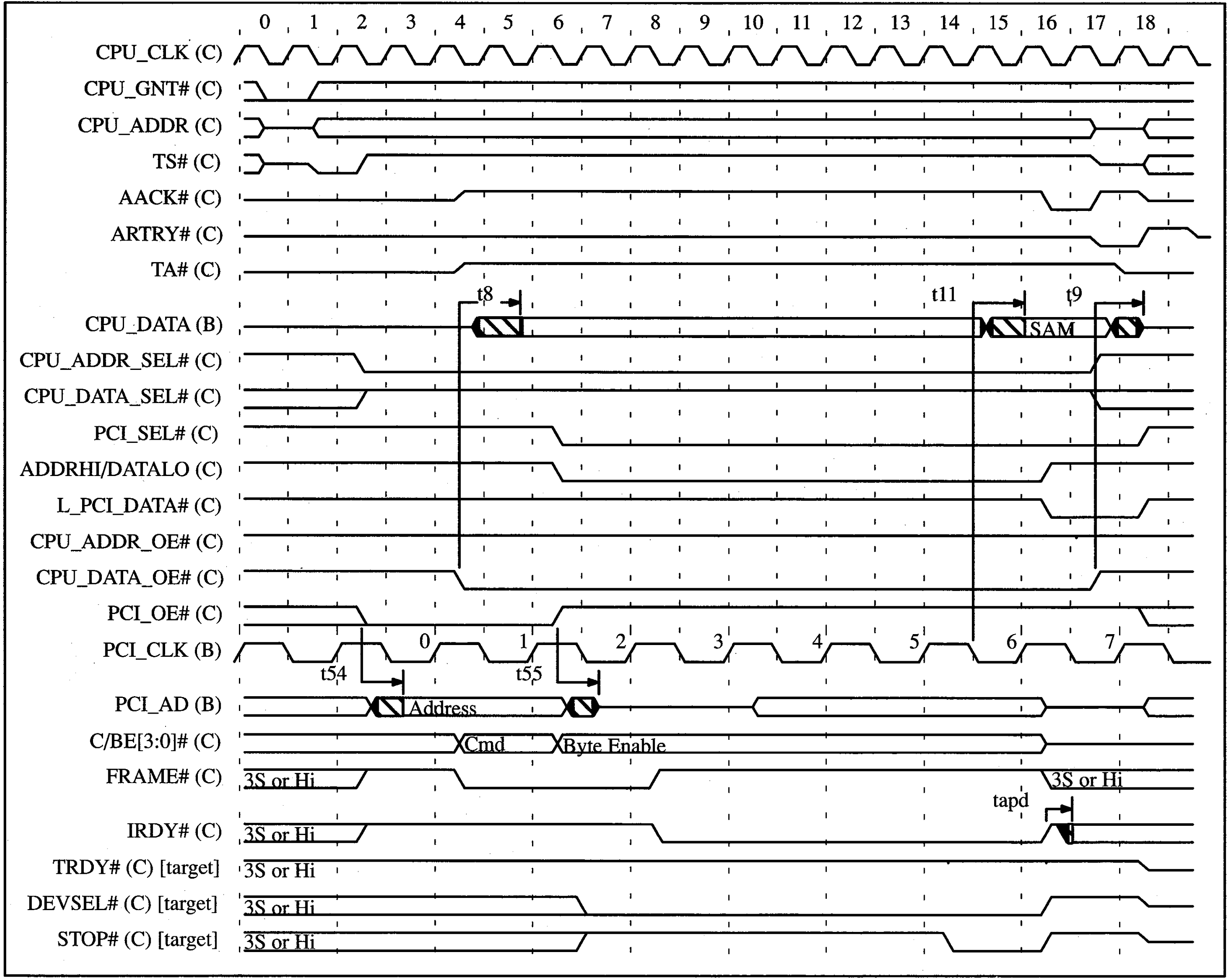


Figure 7–55. CPU To PCI Read – Target Retry

Section 8
The 650 Bridge Pin Lists

This section contains alphabetic and numeric pin lists for the 653 Buffer and the 654 Controller.

8.1 653 Buffer Pin Lists

8.1.1 653 Buffer Numeric Pin List

Table 8–1. 653 Buffer Numeric Pin List

Pin #	Signal Name
1	MEM_PAR (0)
2	MEM_PAR (1)
3	MEM_PAR (2)
4	MEM_PAR (3)
5	MEM_PAR (4)
6	L_ERR_ADDR#
7	MEM_DATA_OE#
8	V_{DD}
9	GND
10	ALL_ONES_SEL#
11	PCI_OE#
12	MEM_PAR (5)
13	MEM_PAR (6)
14	MEM_PAR (7)
15	V_{DD}
16	PCI_AD (0)

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
17	PCI_AD (1)
18	PCI_AD (2)
19	GND
20	PCI_AD (3)
21	PCI_AD (4)
22	PCI_AD (5)
23	PCI_AD (6)
24	V_{DD}
25	GND
26	PCI_AD (7)
27	PCI_AD (8)
28	PCI_AD (9)
29	GND
30	PCI_AD (10)
31	PCI_AD (11)
32	V_{DD}
33	PCI_AD (12)
34	CONTIG_IO
35	PCI_AD (13)
36	PCI_AD (14)
37	PCI_AD (15)
38	V_{DD}
39	GND
40	PCI_AD (16)
41	PCI_AD (17)
42	PCI_AD (18)
43	NO_TRANS
44	PCI_AD (19)
45	V_{DD}

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
46	PCI_AD (20)
47	PCI_AD (21)
48	GND
49	PCI_AD (22)
50	PCI_AD (23)
51	PCI_AD (24)
52	V_{DD}
53	GND
54	PCI_AD (25)
55	PCI_AD (26)
56	PCI_AD (27)
57	PCI_AD (28)
58	GND
59	PCI_AD (29)
60	PCI_AD (30)
61	PCI_AD (31)
62	V_{DD}
63	PCI_AD_PAR
64	MEM_PAGE_HIT#
65	DRAMX9HI/X10LO
66	ROM_SEL#
67	L_PCI_DATA#
68	V_{DD}
69	GND
70	PCI_CLK
71	MEM_PAR_GOOD
72	MEM_DATA_SEL#
73	BURST_CLK#
74	PCI_SEL#

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
75	ADDRHI/DATALO
76	REFRESH_SEL#
77	CPU_DATA (63)
78	CPU_DATA (62)
79	CPU_DATA (61)
80	CPU_DATA (60)
81	CPU_DATA (59)
82	CPU_DATA (58)
83	CPU_DATA (57)
84	CPU_DATA (56)
85	CPU_DATA (55)
86	V_{DD}
87	GND
88	CPU_DATA (54)
89	CPU_DATA (53)
90	CPU_DATA (52)
91	CPU_DATA (51)
92	CPU_DATA (50)
93	CPU_DATA (49)
94	V_{DD}
95	GND
96	CPU_DATA (48)
97	CPU_DATA (47)
98	CPU_DATA (46)
99	CPU_DATA (45)
100	CPU_DATA (44)
101	CPU_DATA (43)
102	V_{DD}
103	GND

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
104	CPU_DATA (42)
105	CPU_DATA (41)
106	CPU_DATA (40)
107	CPU_DATA (39)
108	CPU_DATA (38)
109	GND
110	CPU_DATA (37)
111	CPU_DATA (36)
112	CPU_DATA (35)
113	CPU_DATA (34)
114	V_{DD}
115	GND
116	CPU_DATA (33)
117	CPU_DATA (32)
118	CPU_ADDR (0)
119	CPU_ADDR (1)
120	CPU_ADDR (2)
121	CPU_ADDR (3)
122	CPU_ADDR (4)
123	CPU_ADDR (5)
124	CPU_ADDR (6)
125	CPU_ADDR (7)
126	V_{DD}
127	GND
128	CPU_ADDR (8)
129	CPU_ADDR (9)
130	CPU_ADDR (10)
131	CPU_ADDR (11)
132	CPU_ADDR (12)

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
133	CPU_ADDR (13)
134	GND
135	CPU_ADDR (14)
136	CPU_ADDR (15)
137	CPU_ADDR (16)
138	CPU_ADDR (17)
139	CPU_ADDR (18)
140	CPU_ADDR (19)
141	CPU_ADDR (20)
142	V_{DD}
143	GND
144	CPU_ADDR (21)
145	CPU_ADDR (22)
146	CPU_ADDR (23)
147	CPU_ADDR (24)
148	CPU_ADDR (25)
149	CPU_ADDR (26)
150	CPU_ADDR (27)
151	CPU_ADDR (28)
152	CPU_ADDR (29)
153	CPU_ADDR (30)
154	CPU_ADDR (31)
155	TSIZ (2)
156	TSIZ (1)
157	TSIZ (0)
158	CPU_ADDR_SEL#
159	CPU_ADDR_OE#
160	V_{DD}
161	GND

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
162	CPU_DATA (31)
163	CPU_DATA (30)
164	CPU_DATA (29)
165	CPU_DATA (28)
166	CPU_DATA (27)
167	CPU_DATA (26)
168	CPU_DATA_SEL#
169	GND
170	CPU_DATA (25)
171	CPU_DATA (24)
172	CPU_DATA (23)
173	CPU_DATA (22)
174	CPU_DATA (21)
175	CPU_DATA (20)
176	V_{DD}
177	GND
178	CPU_DATA (19)
179	CPU_DATA (18)
180	CPU_DATA (17)
181	CPU_DATA (16)
182	GND
183	CPU_DATA (15)
184	CPU_DATA (14)
185	CPU_DATA (13)
186	CPU_DATA (12)
187	CPU_DATA (11)
188	CPU_DATA (10)
189	CPU_DATA (9)
190	V_{DD}

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
191	GND
192	CPU_DATA_OE#
193	CPU_DATA (8)
194	CPU_DATA (7)
195	CPU_DATA (6)
196	CPU_DATA (5)
197	CPU_DATA (4)
198	GND
199	CPU_DATA (3)
200	CPU_DATA (2)
201	CPU_DATA (1)
202	CPU_DATA (0)
203	RASHI/CASLO
204	V_{DD}
205	GND
206	MEM_DATA (0)
207	MEM_DATA (1)
208	MEM_DATA (2)
209	MEM_DATA (3)
210	MEM_DATA (4)
211	MEM_DATA (5)
212	V_{DD}
213	GND
214	MEM_DATA (6)
215	MEM_DATA (7)
216	MEM_DATA (8)
217	MEM_DATA (9)
218	MEM_DATA (10)
219	MEM_DATA (11)

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
220	V_{DD}
221	GND
222	TEST#
223	ERR_ADDR_SEL#
224	MEM_DATA (12)
225	MEM_DATA (13)
226	MEM_DATA (14)
227	GND
228	MEM_DATA (15)
229	MEM_DATA (16)
230	MEM_DATA (17)
231	MEM_DATA (18)
232	MEM_DATA (19)
233	MEM_DATA (20)
234	MEM_DATA (21)
235	MEM_DATA (22)
236	MEM_DATA (23)
237	MEM_DATA (24)
238	V_{DD}
239	GND
240	MEM_DATA (25)
241	MEM_DATA (26)
242	MEM_DATA (27)
243	MEM_DATA (28)
244	MEM_DATA (29)
245	MEM_DATA (30)
246	MEM_DATA (31)
247	MEM_ADDR0_B
248	MEM_ADDR (0)

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
249	MEM_ADDR (1)
250	MEM_ADDR (2)
251	MEM_ADDR (3)
252	MEM_ADDR (4)
253	MEM_ADDR (5)
254	V_{DD}
255	GND
256	MEM_ADDR (6)
257	MEM_ADDR (7)
258	MEM_ADDR (8)
259	MEM_ADDR (9)
260	MEM_ADDR (10)
261	MEM_ADDR (11)
262	LE_MODE_SEL#
263	MEM_DATA (32)
264	MEM_DATA (33)
265	MEM_DATA (34)
266	V_{DD}
267	GND
268	MEM_DATA (35)
269	MEM_DATA (36)
270	MEM_DATA (37)
271	MEM_DATA (38)
272	MEM_DATA (39)
273	GND
274	MEM_DATA (40)
275	MEM_DATA (41)
276	MEM_DATA (42)
277	MEM_DATA (43)

Table 8–1. 653 Buffer Numeric Pin List (Continued)

Pin #	Signal Name
278	V_{DD}
279	GND
280	MEM_DATA (44)
281	MEM_DATA (45)
282	MEM_DATA (46)
283	MEM_DATA (47)
284	MEM_DATA (48)
285	MEM_DATA (49)
286	V_{DD}
287	GND
288	MEM_DATA (50)
289	MEM_DATA (51)
290	MEM_DATA (52)
291	MEM_DATA (53)
292	MEM_DATA (54)
293	MEM_DATA (55)
294	V_{DD}
295	GND
296	MEM_DATA (56)
297	MEM_DATA (57)
298	MEM_DATA (58)
299	MEM_DATA (59)
300	MEM_DATA (60)
301	GND
302	MEM_DATA (61)
303	MEM_DATA (62)
304	MEM_DATA (63)

8.1.2 653 Buffer Alphabetic Pin Listing

Table 8–2. 653 Buffer Alphabetic Pin List

Signal Name	Pin #
ADDRHI/DATALO	75
ALL_ONES_SEL#	10
BURST_CLK#	73
CONTIG_IO	34
CPU_ADDR (0)	118
CPU_ADDR (1)	119
CPU_ADDR (2)	120
CPU_ADDR (3)	121
CPU_ADDR (4)	122
CPU_ADDR (5)	123
CPU_ADDR (6)	124
CPU_ADDR (7)	125
CPU_ADDR (8)	128
CPU_ADDR (9)	129
CPU_ADDR (10)	130
CPU_ADDR (11)	131
CPU_ADDR (12)	132
CPU_ADDR (13)	133
CPU_ADDR (14)	135
CPU_ADDR (15)	136
CPU_ADDR (16)	137
CPU_ADDR (17)	138
CPU_ADDR (18)	139
CPU_ADDR (19)	140
CPU_ADDR (20)	141
CPU_ADDR (21)	144
CPU_ADDR (22)	145
CPU_ADDR (23)	146

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
CPU_ADDR (24)	147
CPU_ADDR (25)	148
CPU_ADDR (26)	149
CPU_ADDR (27)	150
CPU_ADDR (28)	151
CPU_ADDR (29)	152
CPU_ADDR (30)	153
CPU_ADDR (31)	154
CPU_ADDR_OE#	159
CPU_ADDR_SEL#	158
CPU_DATA (0)	202
CPU_DATA (1)	201
CPU_DATA (2)	200
CPU_DATA (3)	199
CPU_DATA (4)	197
CPU_DATA (5)	196
CPU_DATA (6)	195
CPU_DATA (7)	194
CPU_DATA (8)	193
CPU_DATA (9)	189
CPU_DATA (10)	188
CPU_DATA (11)	187
CPU_DATA (12)	186
CPU_DATA (13)	185
CPU_DATA (14)	184
CPU_DATA (15)	183
CPU_DATA (16)	181
CPU_DATA (17)	180
CPU_DATA (18)	179

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
CPU_DATA (19)	178
CPU_DATA (20)	175
CPU_DATA (21)	174
CPU_DATA (22)	173
CPU_DATA (23)	172
CPU_DATA (24)	171
CPU_DATA (25)	170
CPU_DATA (26)	167
CPU_DATA (27)	166
CPU_DATA (28)	165
CPU_DATA (29)	164
CPU_DATA (30)	163
CPU_DATA (31)	162
CPU_DATA (32)	117
CPU_DATA (33)	116
CPU_DATA (34)	113
CPU_DATA (35)	112
CPU_DATA (36)	111
CPU_DATA (37)	110
CPU_DATA (38)	108
CPU_DATA (39)	107
CPU_DATA (40)	106
CPU_DATA (41)	105
CPU_DATA (42)	104
CPU_DATA (43)	101
CPU_DATA (44)	100
CPU_DATA (45)	99
CPU_DATA (46)	98
CPU_DATA (47)	97

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
CPU_DATA (48)	96
CPU_DATA (49)	93
CPU_DATA (50)	92
CPU_DATA (51)	91
CPU_DATA (52)	90
CPU_DATA (53)	89
CPU_DATA (54)	88
CPU_DATA (55)	85
CPU_DATA (56)	84
CPU_DATA (57)	83
CPU_DATA (58)	82
CPU_DATA (59)	81
CPU_DATA (60)	80
CPU_DATA (61)	79
CPU_DATA (62)	78
CPU_DATA (63)	77
CPU_DATA_OE#	192
CPU_DATA_SEL#	168
DRAMX9HI/X10LO	65
ERR_ADDR_SEL#	223
GND	9
GND	19
GND	25
GND	29
GND	39
GND	48
GND	53
GND	58
GND	69

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
GND	87
GND	95
GND	103
GND	109
GND	115
GND	127
GND	134
GND	143
GND	161
GND	169
GND	177
GND	182
GND	191
GND	198
GND	205
GND	213
GND	221
GND	227
GND	239
GND	255
GND	267
GND	273
GND	279
GND	287
GND	295
GND	301
L_ERR_ADDR#	6
L_PCI_DATA#	67
LE_MODE_SEL#	262

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
MEM_ADDR (0)	248
MEM_ADDR (1)	249
MEM_ADDR (2)	250
MEM_ADDR (3)	251
MEM_ADDR (4)	252
MEM_ADDR (5)	253
MEM_ADDR (6)	256
MEM_ADDR (7)	257
MEM_ADDR (8)	258
MEM_ADDR (9)	259
MEM_ADDR (10)	260
MEM_ADDR (11)	261
MEM_ADDR0_B	247
MEM_DATA (0)	206
MEM_DATA (1)	207
MEM_DATA (2)	208
MEM_DATA (3)	209
MEM_DATA (4)	210
MEM_DATA (5)	211
MEM_DATA (6)	214
MEM_DATA (7)	215
MEM_DATA (8)	216
MEM_DATA (9)	217
MEM_DATA (10)	218
MEM_DATA (11)	219
MEM_DATA (12)	224
MEM_DATA (13)	225
MEM_DATA (14)	226
MEM_DATA (15)	228

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
MEM_DATA (16)	229
MEM_DATA (17)	230
MEM_DATA (18)	231
MEM_DATA (19)	232
MEM_DATA (20)	233
MEM_DATA (21)	234
MEM_DATA (22)	235
MEM_DATA (23)	236
MEM_DATA (24)	237
MEM_DATA (25)	240
MEM_DATA (26)	241
MEM_DATA (27)	242
MEM_DATA (28)	243
MEM_DATA (29)	244
MEM_DATA (30)	245
MEM_DATA (31)	246
MEM_DATA (32)	263
MEM_DATA (33)	264
MEM_DATA (34)	265
MEM_DATA (35)	268
MEM_DATA (36)	269
MEM_DATA (37)	270
MEM_DATA (38)	271
MEM_DATA (39)	272
MEM_DATA (40)	274
MEM_DATA (41)	275
MEM_DATA (42)	276
MEM_DATA (43)	277
MEM_DATA (44)	280

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
MEM_DATA (45)	281
MEM_DATA (46)	282
MEM_DATA (47)	283
MEM_DATA (48)	284
MEM_DATA (49)	285
MEM_DATA (50)	288
MEM_DATA (51)	289
MEM_DATA (52)	290
MEM_DATA (53)	291
MEM_DATA (54)	292
MEM_DATA (55)	293
MEM_DATA (56)	296
MEM_DATA (57)	297
MEM_DATA (58)	298
MEM_DATA (59)	299
MEM_DATA (60)	300
MEM_DATA (61)	302
MEM_DATA (62)	303
MEM_DATA (63)	304
MEM_DATA_OE#	7
MEM_DATA_SEL#	72
MEM_PAGE_HIT#	64
MEM_PAR (0)	1
MEM_PAR (1)	2
MEM_PAR (2)	3
MEM_PAR (3)	4
MEM_PAR (4)	5
MEM_PAR (5)	12
MEM_PAR (6)	13

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
MEM_PAR (7)	14
MEM_PAR_GOOD	71
NO_TRANS	43
PCI_AD (0)	16
PCI_AD (1)	17
PCI_AD (2)	18
PCI_AD (3)	20
PCI_AD (4)	21
PCI_AD (5)	22
PCI_AD (6)	23
PCI_AD (7)	26
PCI_AD (8)	27
PCI_AD (9)	28
PCI_AD (10)	30
PCI_AD (11)	31
PCI_AD (12)	33
PCI_AD (13)	35
PCI_AD (14)	36
PCI_AD (15)	37
PCI_AD (16)	40
PCI_AD (17)	41
PCI_AD (18)	42
PCI_AD (19)	44
PCI_AD (20)	46
PCI_AD (21)	47
PCI_AD (22)	49
PCI_AD (23)	50
PCI_AD (24)	51
PCI_AD (25)	54

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
PCI_AD (26)	55
PCI_AD (27)	56
PCI_AD (28)	57
PCI_AD (29)	59
PCI_AD (30)	60
PCI_AD (31)	61
PCI_AD_PAR	63
PCI_CLK	70
PCI_OE#	11
PCI_SEL#	74
RASHI/CASLO	203
REFRESH_SEL#	76
ROM_SEL#	66
TEST#	222
TSIZ (0)	157
TSIZ (1)	156
TSIZ (2)	155
V_{DD}	8
V_{DD}	15
V_{DD}	24
V_{DD}	32
V_{DD}	38
V_{DD}	45
V_{DD}	52
V_{DD}	62
V_{DD}	68
V_{DD}	86
V_{DD}	94
V_{DD}	102

Table 8–2. 653 Buffer Alphabetic Pin List (Continued)

Signal Name	Pin #
V_{DD}	114
V_{DD}	126
V_{DD}	142
V_{DD}	160
V_{DD}	176
V_{DD}	190
V_{DD}	204
V_{DD}	212
V_{DD}	220
V_{DD}	238
V_{DD}	254
V_{DD}	266
V_{DD}	278
V_{DD}	286
V_{DD}	294

8.2 654 Controller Pin Lists

8.2.1 654 Controller Numeric Pin List

Table 8–3. 654 Controller Numeric Pin List

Pins	Signal Description
001,010,015,020, 030,041,047,051, 060,070,081,090, 100,110,115,121, 130,140	Voltage: 3.3V
019,099	RESERVED
011,021,031,040, 050,061,071, 080,091,101,111, 120,128,131, 141,143,151,160	Ground
002	ALL_ONES_SEL#
003	PCI_GNT[2]#
004	PCI_GNT[5]#
005	PCI_GNT[4]#
006	PCI_GNT[1]#
007	IO_BRDG_GNT#
008	BURST_CLK#
009	REFRESH_SEL#
012	NMI_IRQ
013	NO_TRANS
014	ROM_SEL#
016	PCI_STOP#
017	PCI_C/BE[1]#
018	PCI_C/BE[0]#
022	CAS[0]#
023	CAS[1]#
024	CAS[2]#
025	CAS[3]#

Table 8–3. 654 Controller Numeric Pin List (Continued)

Pins	Signal Description
026	WE[1]#
027	PCI_TRDY#
028	PCI_DEVSEL#
029	PCI_FRAME#
032	IO_BRDG_IRQ
033	PCI_REQ[5]#
034	PCI_REQ[4]#
035	PCI_REQ[3]#
036	PCI_REQ[2]#
037	SRESET_CPU#
038	PCI_REQ[1]#
039	IO_BRDG_REQ#
042	MEM_PAR_ERR#
043	DPE_ERR#
044	TT_ERR#
045	IO_BRDG_HOLD#
046	MC_SETUP#
048	PCI_C/BE[3]#
049	PCI_C/BE[2]#
052	PCI_IRDY#
053	PCI_PAR
055	WE[0]#
056	CAS[7]#
057	CAS[6]#
058	CAS[5]#
059	CAS[4]#
062	RAS[7]#
063	RAS[6]#

Table 8–3. 654 Controller Numeric Pin List (Continued)

Pins	Signal Description
064	RAS[5]#
065	RAS[4]#
066	RAS[3]#
067	RAS[2]#
068	RAS[1]#
069	RAS[0]#
072	ROM_WE#
073	ROM_OE#
074	ROM_CS#
075	RI#
076	ERR_ADDR_SEL#
077	MASK_TEA#
078	RESERVED
079	ISA_MASTER#
082	L2_PRESENT#
083	INT_CPU#
084	BE_PAR_EN#
085	LE_PAR_EN#
086	RESET#
087	REFRESH_REQ#
088	PCI_GNT[3]#
089	SRESET_REQ#
092	LE_MODE_SEL#
093	CPU_ADDR[29]
094	CPU_ADDR[30]
095	CPU_ADDR[8]
096	CPU_ADDR[7]
097	CPU_ADDR[6]

Table 8–3. 654 Controller Numeric Pin List (Continued)

Pins	Signal Description
098	CPU_ADDR[5]
102	CPU_ADDR[4]
103	CPU_ADDR[3]
104	CPU_ADDR[2]
105	CPU_ADDR[1]
106	CPU_ADDR[0]
107	CPU_ADDR[19]
108	CPU_ADDR[31]
109	L2_CACHE_REQ#
112	ARTRY#
113	AACK#
114	L2_CACHE_GNT#
116	TSIZ[2]
117	DI#
118	TSIZ[1]
119	TSIZ[0]
122	TT[1]
123	TT[3]
124	L2_CLAIM#
125	XATS#
126	CPU_REQ#
127	TBST#
129	PCI_CLK
132	TS#
133	CPU_GNT#
134	TA#
135	TEA#
136	TEST#

Table 8–3. 654 Controller Numeric Pin List (Continued)

Pins	Signal Description
137	TT[0]
138	TT[2]
139	DPE#
142	CPU_CLK
144	CPU_ADDR_OE#
145	CPU_DATA_OE#
146	L_PCI_DATA#
147	ADDRHI/DATALO
148	PCI_OE#
149	MEM_PAGE_HIT#
150	LE_MODE_REQ
152	PCI_AD_PAR
153	MEM_DATA_SEL#
154	MEM_DATA_OE#
155	MEM_PAR_GOOD
156	RASHI/CASLO
157	PCI_SEL#
158	CPU_DATA_SEL#
159	CPU_ADDR_SEL#

8.2.2 654 Controller Alphabetic Pin List

Table 8–4. 654 Controller Alphabetic Pin List

Signal Description	Pins
AACK#	113
ADDRHI/DATALO	147
ALL_ONES_SEL#	002
ARTRY#	112
BE_PAR_EN#	084
BURST_CLK#	008
CAS[0]#	022
CAS[1]#	023
CAS[2]#	024
CAS[3]#	025
CAS[4]#	059
CAS[5]#	058
CAS[6]#	057
CAS[7]#	056
CPU_ADDR[0]	106
CPU_ADDR[1]	105
CPU_ADDR[2]	104
CPU_ADDR[3]	103
CPU_ADDR[4]	102
CPU_ADDR[5]	098
CPU_ADDR[6]	097
CPU_ADDR[7]	096
CPU_ADDR[8]	095
CPU_ADDR[19]	107
CPU_ADDR[29]	093
CPU_ADDR[30]	094
CPU_ADDR[31]	108

Table 8–4. 654 Controller Alphabetic Pin List (Continued)

Signal Description	Pins
CPU_ADDR_OE#	144
CPU_ADDR_SEL#	159
CPU_CLK	142
CPU_DATA_OE#	145
CPU_DATA_SEL#	158
CPU_GNT#	133
CPU_REQ#	126
DI#	117
DPE#	139
DPE_ERR#	043
ERR_ADDR_SEL#	076
Ground	011,021,031,040,050, 054,061,071,080,091, 101,111,120,128,131, 141,143,151,160
INT_CPU#	083
IO_BRDG_GNT#	007
IO_BRDG_HOLD#	045
IO_BRDG_IRQ	032
IO_BRDG_REQ#	039
ISA_MASTER#	079
L_PCI_DATA#	146
L2_CACHE_GNT#	114
L2_CACHE_REQ#	109
L2_CLAIM#	124
L2_PRESENT#	082
LE_MODE_REQ	150
LE_MODE_SEL#	092
LE_PAR_EN#	085

Table 8–4. 654 Controller Alphabetic Pin List (Continued)

Signal Description	Pins
MASK_TEA#	077
MC_SETUP#	046
MEM_DATA_OE#	154
MEM_DATA_SEL#	153
MEM_PAGE_HIT#	149
MEM_PAR_ERR#	042
MEM_PAR_GOOD	155
NMI_IRQ	012
NO_TRANS	013
PCI_AD_PAR	152
PCI_C/BE[0]#	018
PCI_C/BE[1]#	017
PCI_C/BE[2]#	049
PCI_C/BE[3]#	048
PCI_CLK	129
PCI_DEVSEL#	028
PCI_FRAME#	029
PCI_GNT[1]#	006
PCI_GNT[2]#	003
PCI_GNT[3]#	088
PCI_GNT[4]#	005
PCI_GNT[5]#	004
PCI_IRDY#	052
PCI_OE#	148
PCI_PAR	053
PCI_REQ[1]#	038
PCI_REQ[2]#	036
PCI_REQ[3]#	035

Table 8–4. 654 Controller Alphabetic Pin List (Continued)

Signal Description	Pins
PCI_REQ[4]#	034
PCI_REQ[5]#	033
PCI_SEL#	157
PCI_STOP#	016
PCI_TRDY#	027
RAS[0]#	069
RAS[1]#	068
RAS[2]#	067
RAS[3]#	066
RAS[4]#	065
RAS[5]#	064
RAS[6]#	063
RAS[7]#	062
RASHI/CASLO	156
REFRESH_REQ#	087
REFRESH_SEL#	009
RESERVED	019,099
RESERVED	078
RESET#	086
RI#	075
ROM_CS#	074
ROM_OE#	073
ROM_SEL#	014
ROM_WE#	072
SRESET_CPU#	037
SRESET_REQ#	089
TA#	134
TBST#	127

Table 8–4. 654 Controller Alphabetic Pin List (Continued)

Signal Description	Pins
TEA#	135
TEST#	136
TS#	132
TSIZ[0]	119
TSIZ[1]	118
TSIZ[2]	116
TT[0]	137
TT[1]	122
TT[2]	138
TT[3]	123
TT_ERR#	044
Voltage: 3.3V	001,010,015,020,030, 041,047,051,054,060, 070,081,090,100,110, 115,121,130,140
WE[0]#	055
WE[1]#	026
XATS#	125

Section 9
650 Bridge Mechanical Drawings

9.1 653 Buffer Quad Flat Pack Component Detail

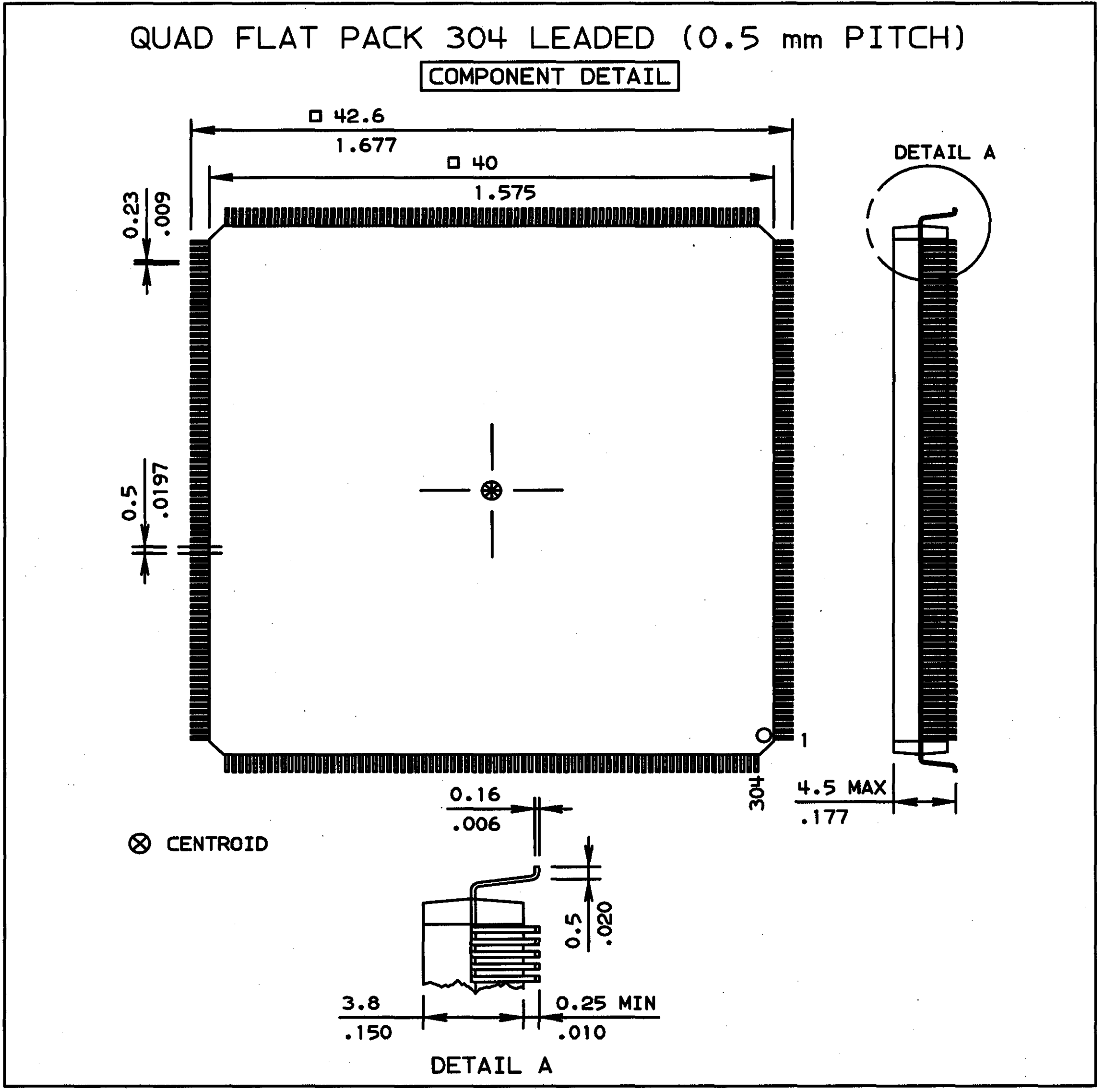


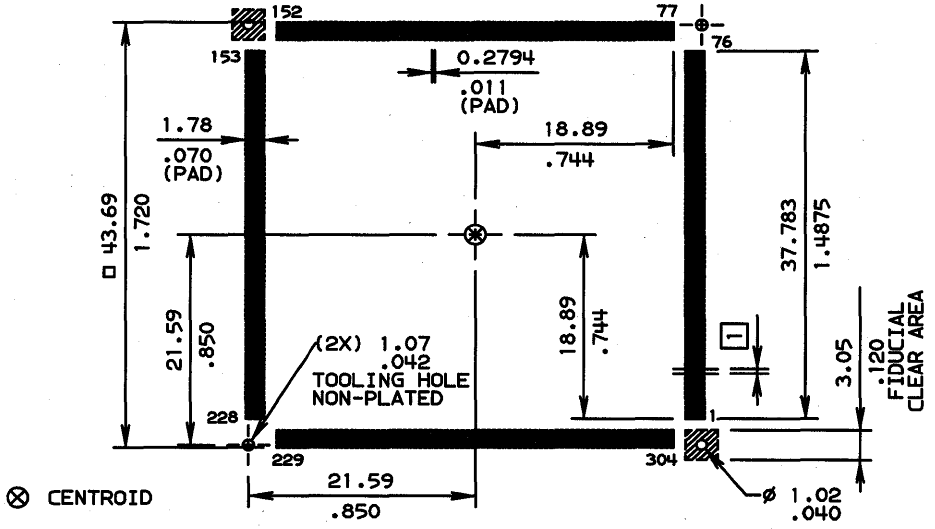
Figure 9–1. 653 Buffer Quad Flat Pack Component Detail

9.2 653 Buffer Quad Flat Pack Component Footprint

QUAD FLAT PACK 304 LEADED (0.5 mm PITCH)

COMPONENT FOOTPRINT

FRONT SIDE



SMT SPACING (PITCH) FROM PAD CENTERLINE

* SPACING IS TO NEAREST 0.013 mm (.0005 in)

PAD	SPACING 1	PAD	SPACING 1	PAD	SPACING 1	PAD	SPACING 1
1	0.000 (.0000)	22	10.503 (.4135)	43	21.006 (.8270)	64	31.496 (1.2400)
2	0.495 (.0195)	23	10.998 (.4330)	44	21.501 (.8465)	65	32.004 (1.2600)
3	1.003 (.0395)	24	11.506 (.4530)	45	21.996 (.8660)	66	32.499 (1.2795)
4	1.498 (.0590)	25	12.002 (.4725)	46	22.504 (.8860)	67	32.995 (1.2990)
5	1.994 (.0785)	26	12.497 (.4920)	47	23.000 (.9055)	68	33.503 (1.3190)
6	2.502 (.0985)	27	13.005 (.5120)	48	23.495 (.9250)	69	33.998 (1.3385)
7	2.997 (.1190)	28	13.500 (.5315)	49	24.003 (.9450)	70	34.506 (1.3585)
8	3.505 (.1380)	29	13.995 (.5510)	50	24.498 (.9645)	71	35.001 (1.3780)
9	4.001 (.1575)	30	14.503 (.5710)	51	25.006 (.9845)	72	35.497 (1.3975)
10	4.496 (.1770)	31	14.999 (.5905)	52	25.502 (1.0040)	73	36.005 (1.4175)
11	5.004 (.1970)	32	15.494 (.6100)	53	25.997 (1.0235)	74	36.500 (1.4370)
12	5.499 (.2165)	33	16.002 (.6300)	54	26.505 (1.0435)	75	36.995 (1.4565)
13	5.994 (.2360)	34	16.497 (.6495)	55	27.000 (1.0630)	76	37.503 (1.4765)
14	6.502 (.2560)	35	17.005 (.6695)	56	27.496 (1.0825)		
15	6.998 (.2755)	36	17.501 (.6890)	57	28.004 (1.1025)		
16	7.506 (.2955)	37	17.996 (.7085)	58	28.499 (1.1220)		
17	8.001 (.3150)	38	18.504 (.7285)	59	28.994 (1.1415)		
18	8.496 (.3345)	39	18.999 (.7480)	60	29.502 (1.1615)		
19	9.004 (.3545)	40	19.495 (.7675)	61	29.997 (1.1810)		
20	9.450 (.3740)	41	20.003 (.7875)	62	30.505 (1.2010)		
21	9.995 (.3935)	42	20.498 (.8070)	63	31.000 (1.2205)		

MAY92
QFP304

Figure 9–2. 653 Buffer Quad Flat Pack Component Footprint

9.3 654 Controller 160-Pin Flat Pack Component Detail

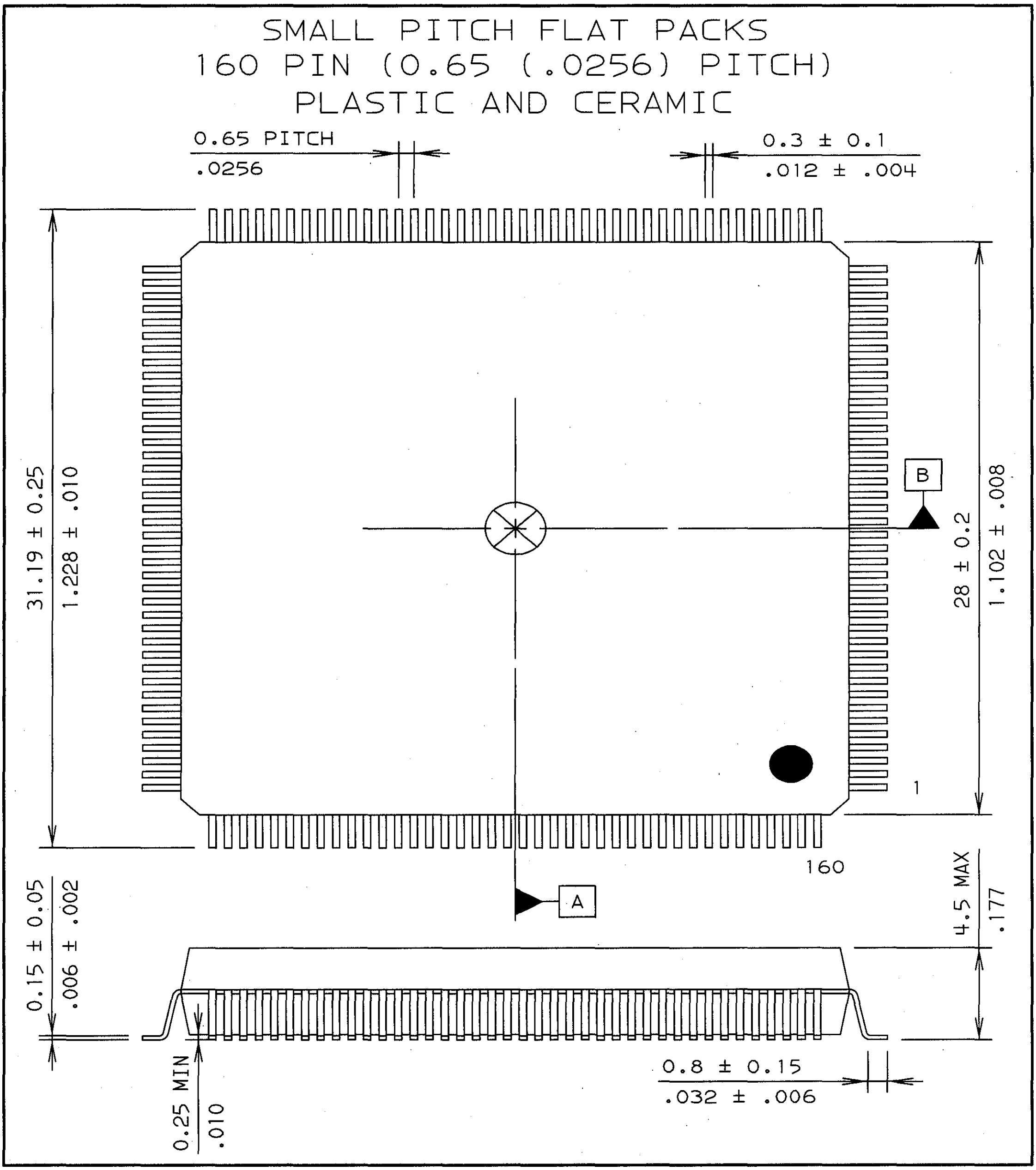
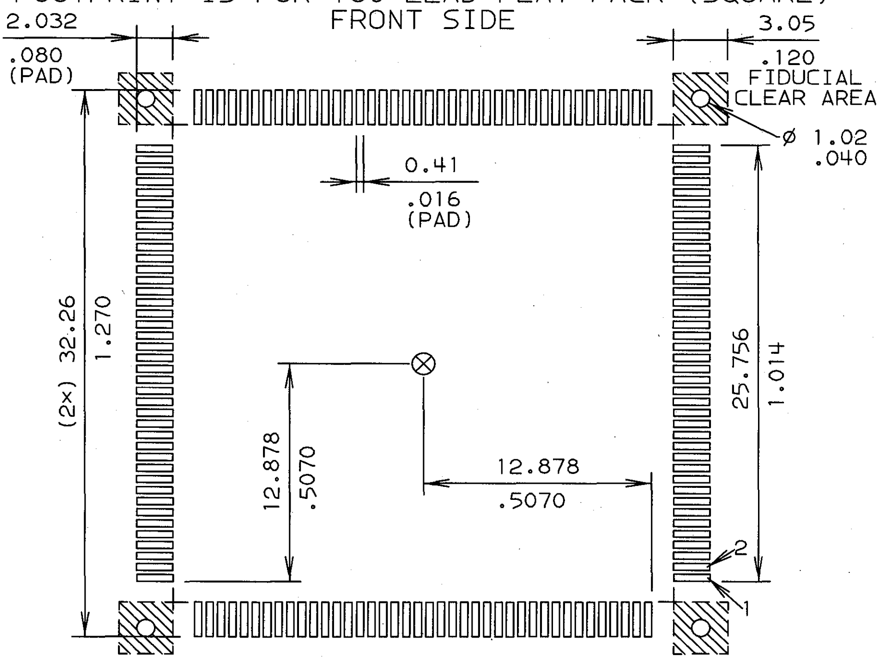


Figure 9–3. 160-Pin Flat Pack

9.4 654 Controller 160-Pin Flat Pack Component Footprint

SMALL PITCH (F.P.) COMPONENT
PAD LOCATIONS (0.65mm PITCH)

FOOTPRINT IS FOR 160 LEAD FLAT PACK (SQUARE)



SMT PAD SPACING (PITCH) FROM PAD CENTER LINE
* SPACING IS TO NEAREST 0.013 mm (.0005 in)

PAD #	SPACING	PAD #	SPACING	PAD #	SPACING
1	0.000 (.0000)	15	9.106 (.3585)	29	18.199 (.7165)
2	0.648 (.0255)	16	9.754 (.3840)	30	18.847 (.7420)
3	1.295 (.0510)	17	10.401 (.4095)	31	19.494 (.7675)
4	1.956 (.0770)	18	11.049 (.4350)	32	20.155 (.7935)
5	2.603 (.1025)	19	11.697 (.4605)	33	20.803 (.8190)
6	3.251 (.1280)	20	12.344 (.4860)	34	21.45 (.8445)
7	3.899 (.1535)	21	13.005 (.5120)	35	22.098 (.8700)
8	4.547 (.1790)	22	13.652 (.5375)	36	22.746 (.8955)
9	5.194 (.2045)	23	14.3 (.5630)	37	23.406 (.9215)
10	5.842 (.2300)	24	14.948 (.5885)	38	24.054 (.9470)
11	6.502 (.2560)	25	15.596 (.6140)	39	24.701 (.9725)
12	7.15 (.2815)	26	16.256 (.6400)	40	25.349 (.9980)
13	7.798 (.3070)	27	16.904 (.6655)		
14	8.445 (.3325)	28	17.551 (.6910)		

REMAINING PADS (3 SIDES) ARE A REPLICATION OF THIS SPACING

OCT90
QFP160A

⊗ CENTROID

Figure 9–4. 160-Pin Flat Pack Pad Locations

Appendix A
Initialization and Setup Requirements

A.1 Processor Initialization

The 601 processor comes up with the cache enabled and bus error checking disabled. The 603 and 604 processors come up with the cache disabled.

A.1.1 Cache Setup

The L1 and L2 caches must be managed in such a way as to purge any lines that are cached during the early part of the boot process so that cast–outs of these lines cannot occur afterward.

All memory pages 2G to 4G must be marked as non-cacheable.

The high priority snoop request function must be enabled (set to 1) by software before running any code that could cause L1 or L2 cache hits on snoops. This bit is HID0–bit 31. The 60X CPU always asserts the HP_SNP_REQ pin, and it depends on the high priority snoop push to function in order to prevent potential livelocks or deadlocks when 60X to PCI cycles are retried by the target.

The L1 cache should be managed in such a way that no cast–outs with the ROM addresses can occur. (Instructions and possibly data are cached from when the 601 first turns on until the L1 is disabled.)

A.1.2 PIO Setup

The segment register T bit, bit 0, defaults to 0 which is the normal storage access mode. It must be left in this state for the hardware to function. Direct store (PIO) segments are not supported.

A.1.3 ARTRY# Precharge

The bit that controls ARTRY# negation, HID0(29), should be set to 0 to enable the precharge of ARTRY# for example system configurations which do not have a device inserted at the upgrade socket or for configurations using the IBM 256/512K write through L2 cache card.

It may be necessary set HID0(29) to 1 to disable the precharge of ARTRY# for example system configurations having a device such as a write-back L2 cache which drives the ARTRY# line. See the specifications for the device inserted at the upgrade socket for details. The bit should be correctly set prior to running any cycles which can be snooped.

A.1.4 Checkstop Enable

HID0 bit 0, Master Checkstop Enable, defaults to 1 which is the enabled state. It should be left in this state so that checkstops can occur.

A.1.5 Bus Error Checks

All error checking is implemented externally using the TEA pin, so the bus error checks should always be left disabled. These error checks are controlled by bits 21, 22, and 23 of register HID0. MSR bit 19, the Machine Check Enable bit, defaults to 1 which is the enabled state. It should be left in this state so that the TEA error checking mechanism can function.

A.2 Initialization of the IBM 82650 Bridge Chip Set

Before DRAM memory operations can begin, software must:

- Read the SIMM presence detect and SIMM type registers.
- Set up and check the registers in the memory controller.

The memory controller SIMM programming register, port 0820 in the example system, is used to access the 654 Controller internal registers to program the starting address of each SIMM and the top of memory. The Memory Controller Timing register, port 0821 in the example system, is used to access the 654 Controller system setup register. Settings for the system setup register are explained in Section 5.2.2.

A.3 I/O Bridge Setup

Program the timer in the Intel SIO register which controls ISA refresh timing. This is counter 1 in the SIO timer section; it should be programmed to operate in Mode 2 with an interval of approximately 15 usec. This timer controls the refresh interval.

Make sure 200 usec has elapsed since starting the timer so that sufficient refresh cycles have occurred to properly start the memory. This will be hidden if approximately 120 ROM accesses occur after the timer is started and before the memory initialization starts.

Initialize all of memory so that all parity bits are properly set. The processor may cache unnecessary data, therefore all of memory must be initialized.

Note: The 650 Bridge does not require reconfiguration when port 4Dh in the SIO chip is utilized to reset the native I/O and the ISA slots.

A.4 PCI Memory Address Assignment

Software should not map any PCI memory at PCI addresses which ISA masters can create—addresses from 0 to 16M. Contention will occur between a device with PCI memory mapped at that address and the ISA master cycles.

A.5 PCI Configuration Scan

The example system allows a software scan to determine the configuration of PCI devices. This is because the system returns 64 one-bits rather than an error when no PCI device responds to initialization cycles. Software can read each possible PCI device ID to determine devices present.

****WARNING****

Using some addresses can cause bus contention on the example system, because multiple PCI slots could be selected. For example, using any 60X address with both CPU_ADDR[19] and CPU_ADDR[20] = 1 causes both the SIO and SCSI to be selected, possibly resulting in damage.

Appendix B
Example Implementation

This section contains schematics for an example system. The schematics illustrate the implementation of a 650 Bridge chip set with a PowerPC 601 microprocessor. The example system design includes provision for an L2 cache or an upgrade microprocessor. Eight SIMM slots allow up to 256M of system memory to be installed, in 8M or 32M sizes and in any configuration in the eight slots.

The example system uses an Intel I/O bridge chip for ISA bus support. The I/O bridge also provides refresh timing and an X-bus interface for various system support functions.

IBM MICROELECTRONICS *

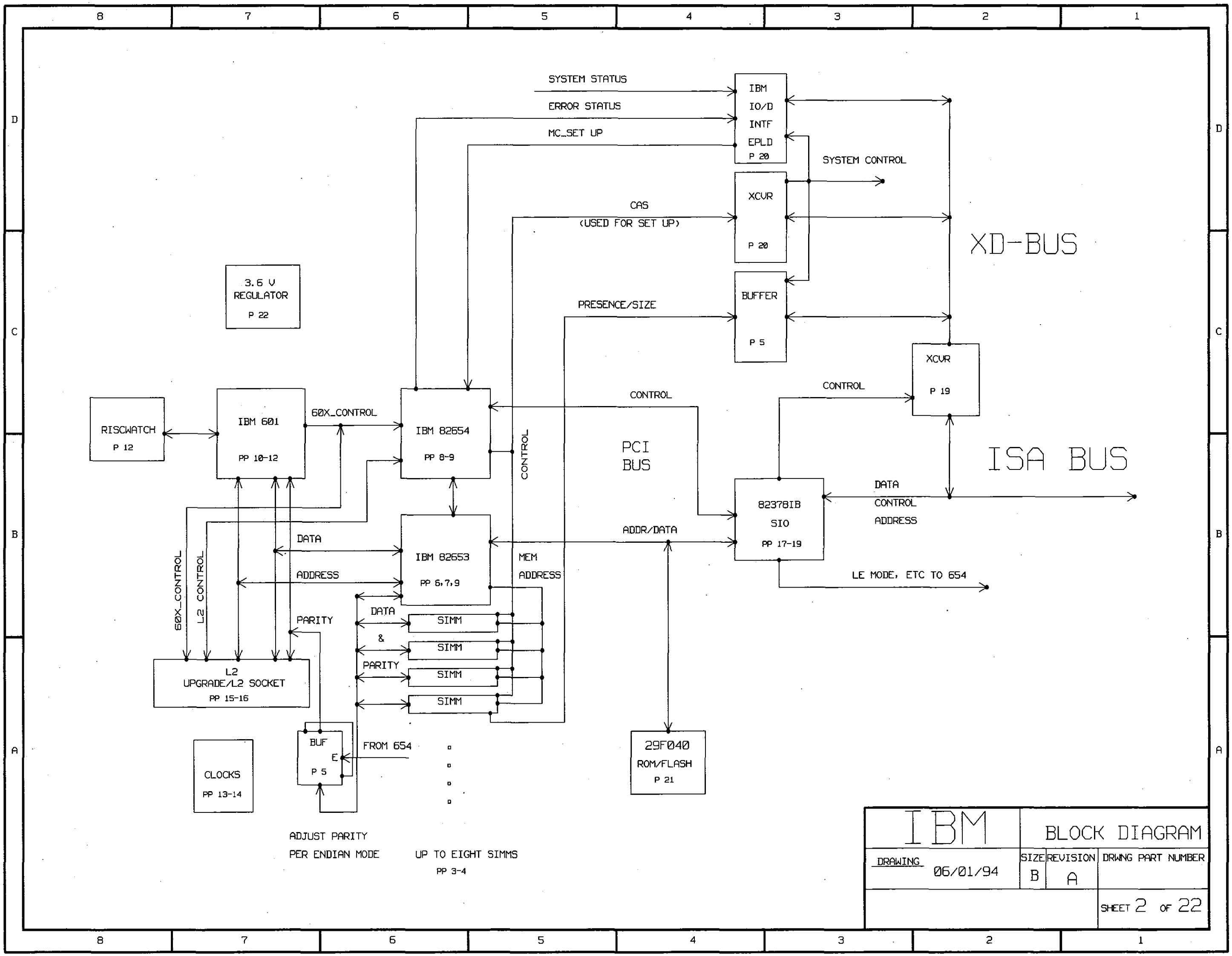
POWER PC 601 *
PROCESSOR COMPLEX

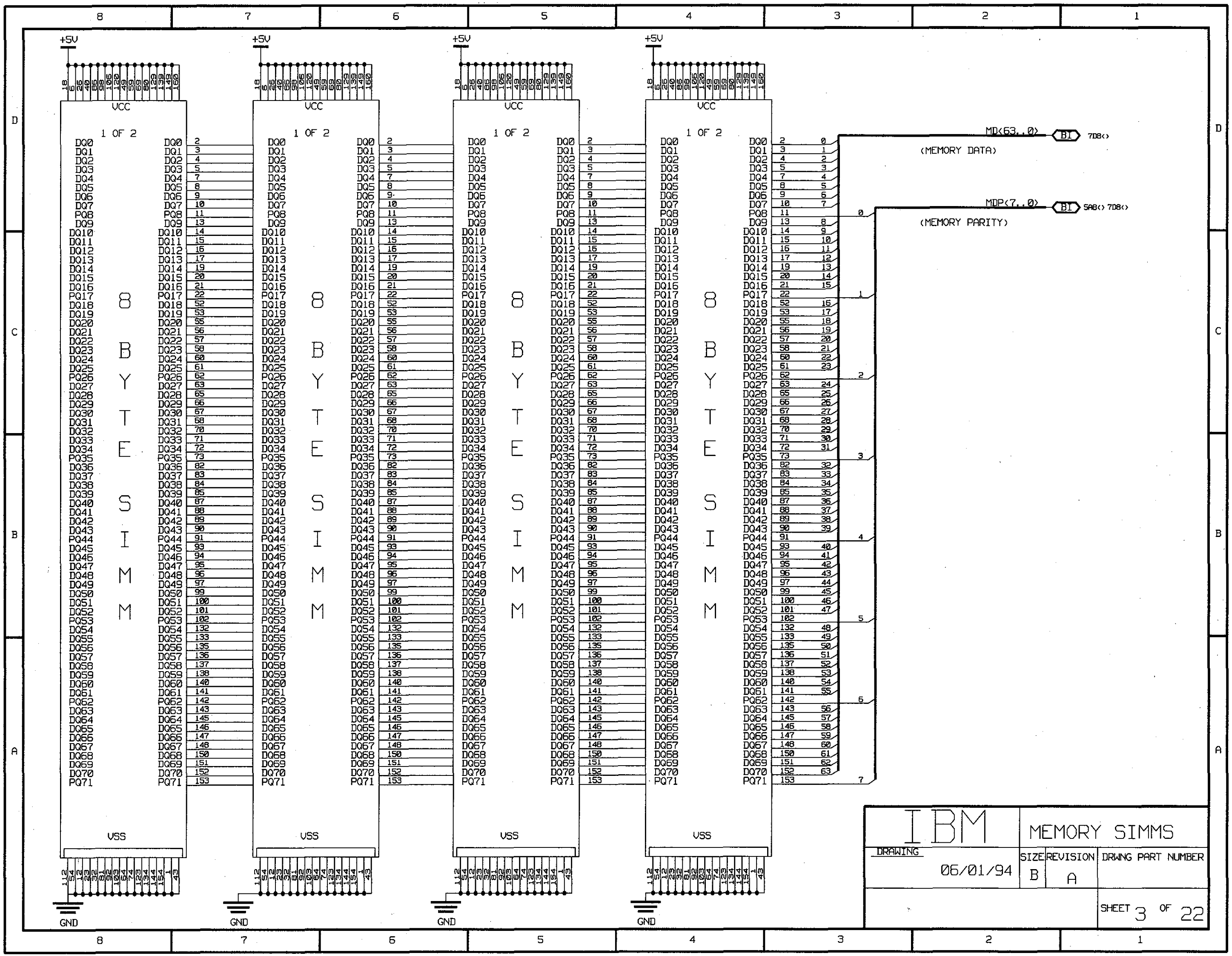
THIS SCHEMATIC IS A FRAGMENT OF A TYPICAL SYSTEM SCHEMATIC. ITS PURPOSE IS TO ILLUSTRATE HOW TO CONNECT THE 650 CHIP SET IN A TYPICAL APPLICATION. THERE ARE A LARGE NUMBER OF WIRES THAT HAVE NO SOURCES OR LOADS WITHIN THESE PAGES BECAUSE THEY CONNECT TO UNSHOWN PARTS OF THE DESIGN. ONLY ENOUGH IS SHOWN TO ILLUSTRATE THE BASIC STRUCTURE OF A SYSTEM.

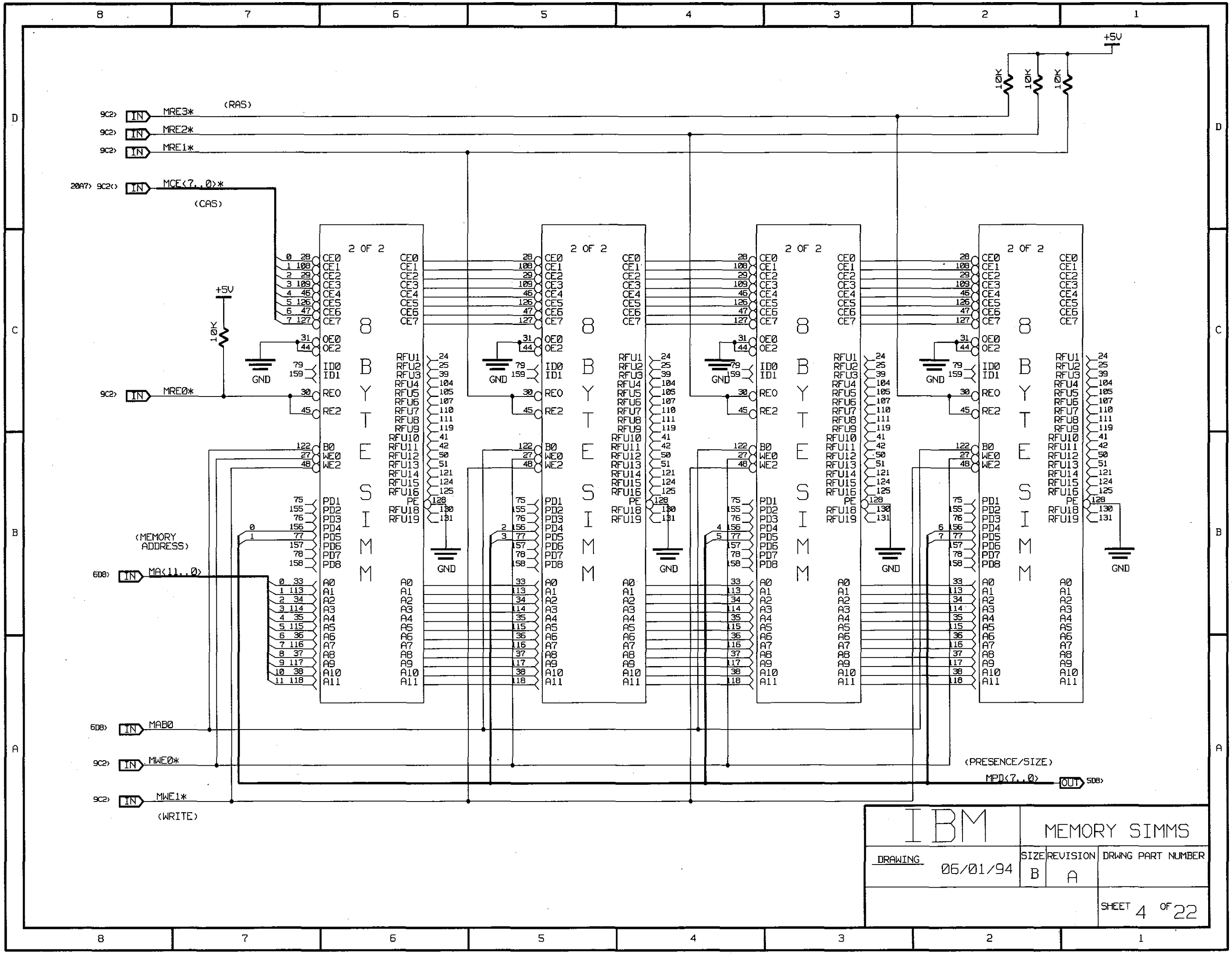
THIS DOCUMENT CONTAINS PRELIMINARY INFORMATION AND IS SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM ASSUMES NO RESPONSIBILITY OF LIABILITY FOR ANY USE OF THE INFORMATION CONTAINED HEREIN. NOTHING IN THIS DOCUMENT SHALL OPERATE AS AN EXPRESS OR IMPLIED LICENSE OR INDEMNITY UNDER THE INTELLECTUAL PROPERTY RIGHTS OF IBM OR THIRD PARTIES. NO WARRANTY OR GUARANTEE IS GIVEN CONCERNING ANY INFORMATION CONTAINED IN THIS DOCUMENT.

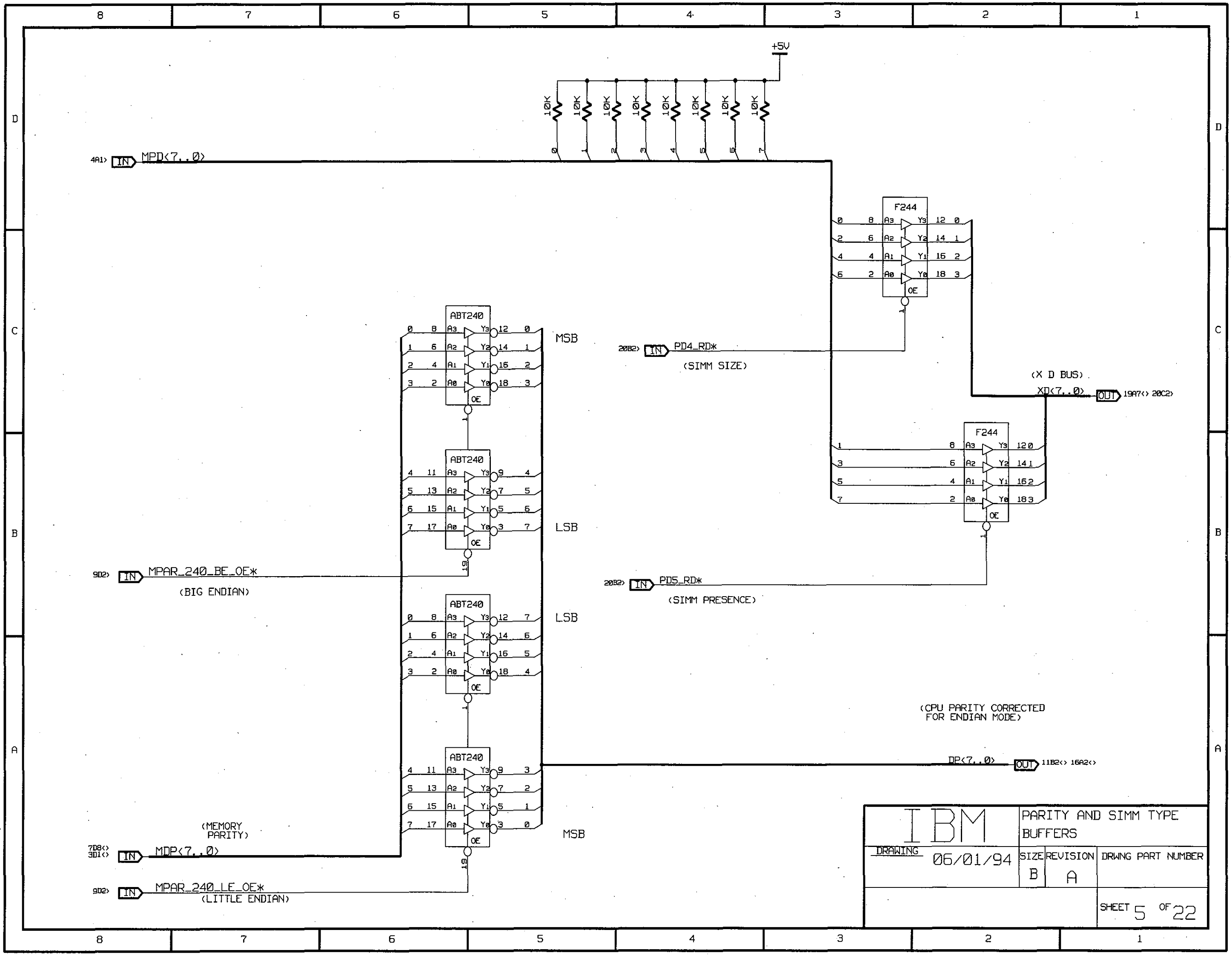
* TRADE MARKS OF IBM CORP.

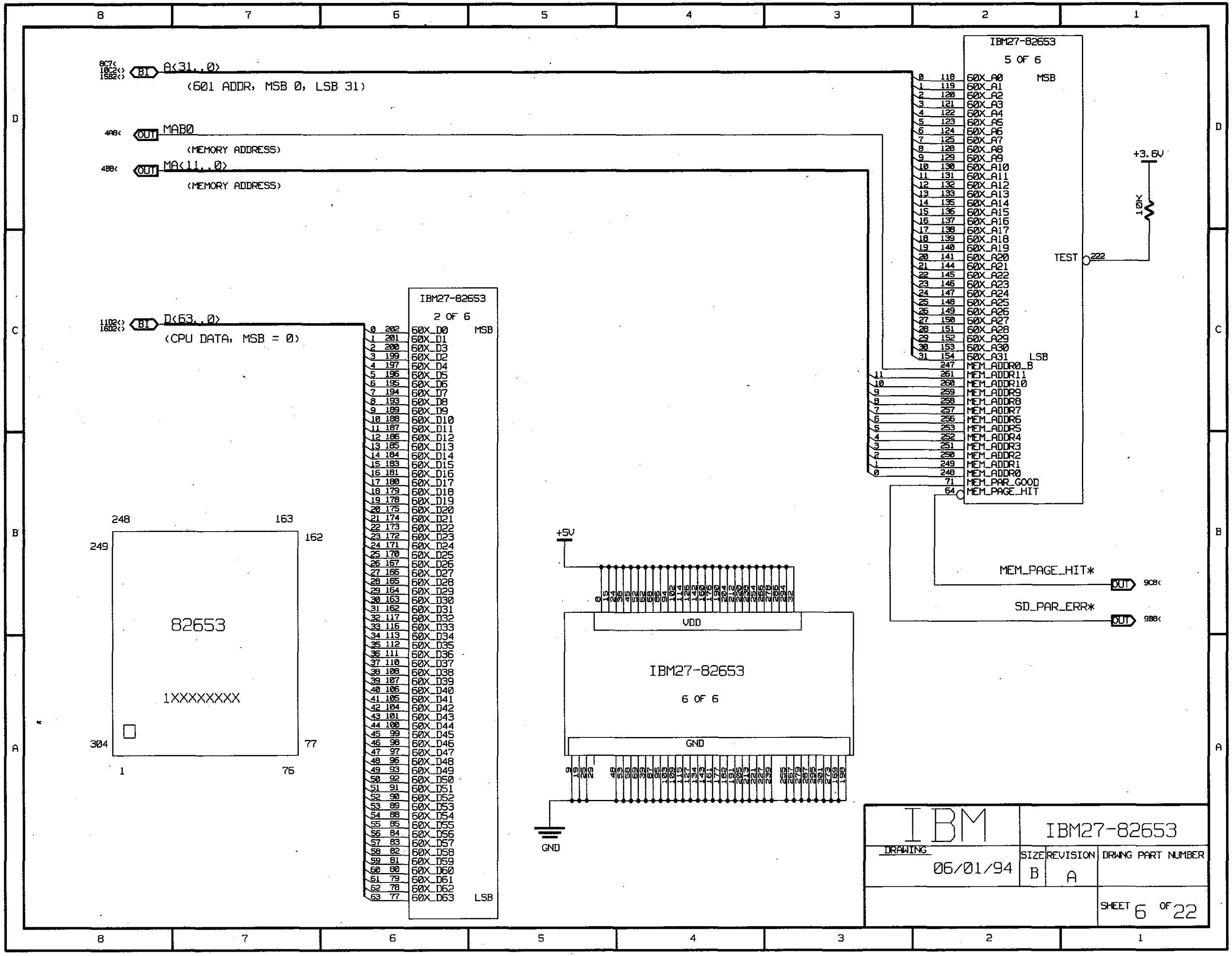
IBM			
DRAWING 06/1/94	SIZE B	REVISION A	DRWNG PART NUMBER
			SHEET 1 OF 22

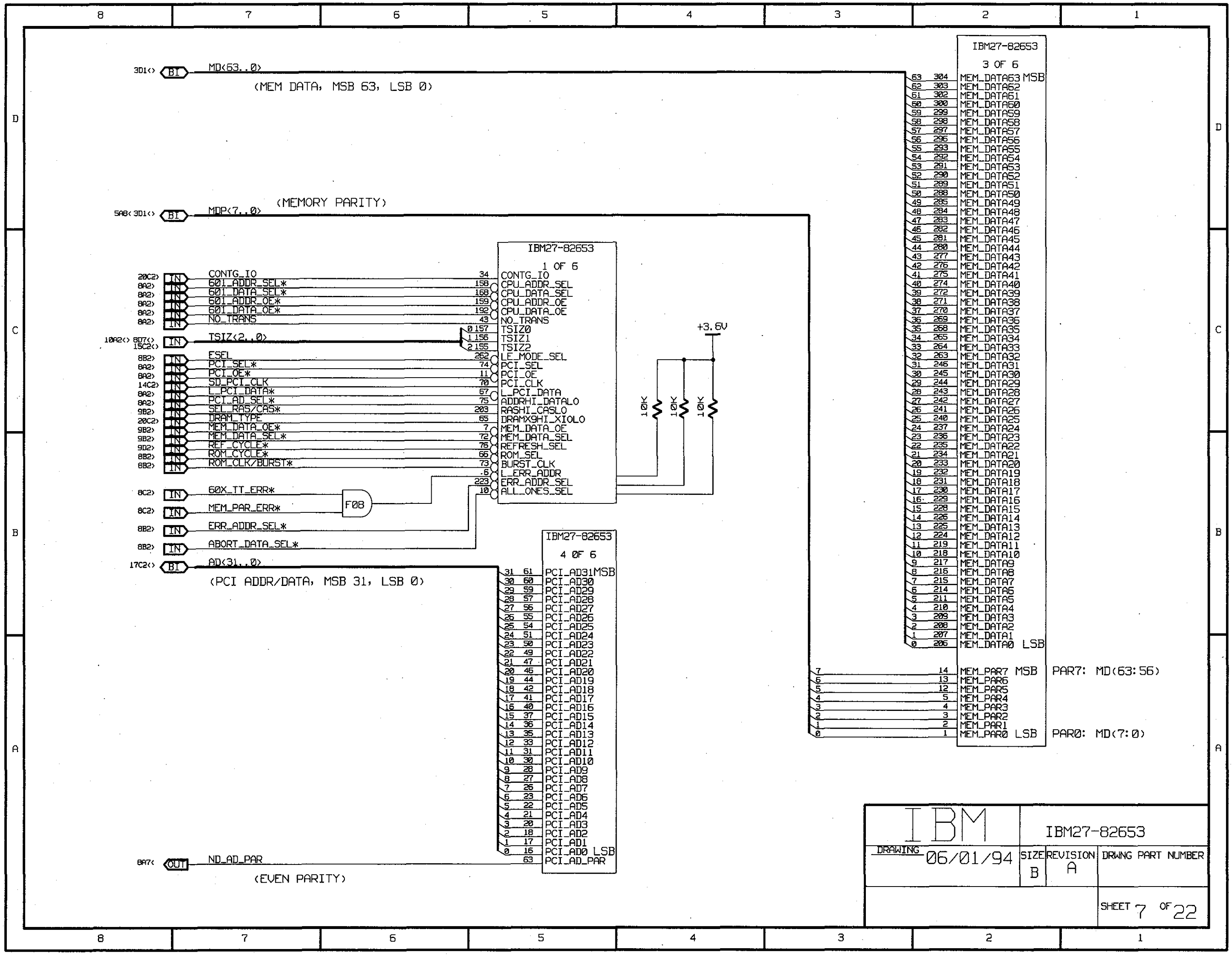

SYSTEM STATUS
ERROR STATUS
MC_SET UP
IBM IO/D INTF EPLD P 20
SYSTEM CONTROL
CAS
(USED FOR SET UP)
XCVR P 20
XD-BUS
BUFFER P 5
PRESENCE/SIZE
3.6 V REGULATOR P 22
XCVR P 19
CONTROL
CONTROL
RISCWATCH P 12
IBM 601 PP 10-12
60X_CONTROL
IBM 82654 PP 8-9
CONTROL
PCI BUS
ISA BUS
DATA
CONTROL
ADDRESS
82378IB SIO PP 17-19
60X_CONTROL
L2 CONTROL
DATA
ADDRESS
IBM 82653 PP 6,7,9
ADDR/DATA
MEM ADDRESS
LE MODE, ETC TO 654
PARITY
DATA
&
PARITY
SIMM
SIMM
SIMM
SIMM
L2 UPGRADE/L2 SOCKET PP 15-16
CLOCKS PP 13-14
BUF E P 5
FROM 654
29F040 ROM/FLASH P 21
ADJUST PARITY PER ENDIAN MODE
UP TO EIGHT SIMMS PP 3-4
IBM
BLOCK DIAGRAM
DRAWING 06/01/94
SIZE B
REVISION A
DRWNG PART NUMBER
SHEET 2 OF 22
8 7 6 5 4 3 2 1
D C B A


+5V
VCC
1 OF 2
8 BYTE SIMM
DQ0 … DQ71
PQ8 PQ17 PQ26 PQ35 PQ44 PQ53 PQ62 PQ71
VSS
GND
MD<63..0>
BI
7D8<>
(MEMORY DATA)
MDP<7..0>
5A8<> 7D8<>
(MEMORY PARITY)
IBM
MEMORY SIMMS
DRAWING
06/01/94
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 3 OF 22
8 7 6 5 4 3 2 1
D C B A

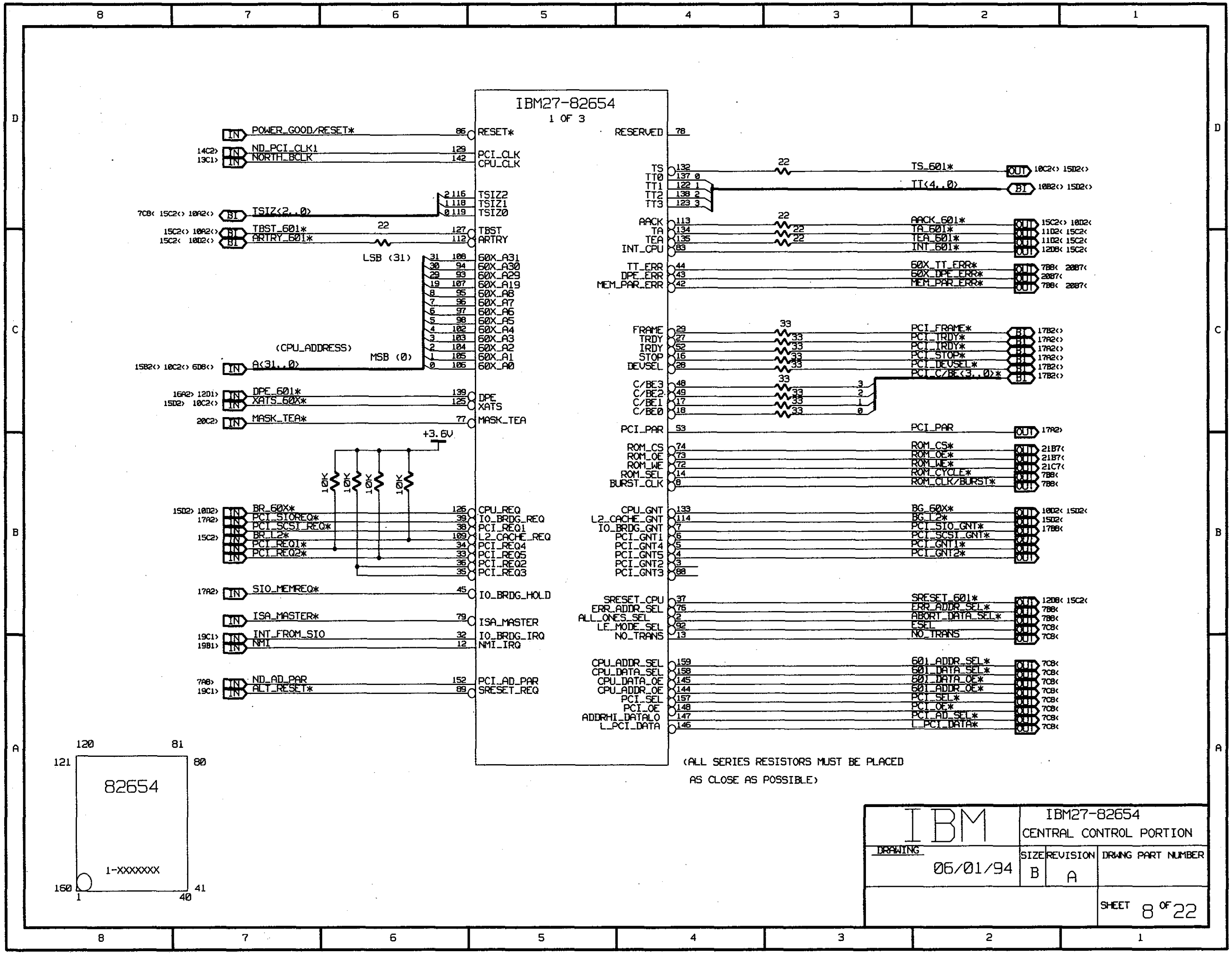

+5V
10K
10K
10K
(RAS)
9C2> IN MRE3*
9C2> IN MRE2*
9C2> IN MRE1*
20A7> 9C2<> IN MCE<7..0>*
(CAS)
+5V
10K
9C2> IN MRE0*
2 OF 2
CE0 CE1 CE2 CE3 CE4 CE5 CE6 CE7
OE0 OE2
ID0 ID1
REO RE2
B0 WE0 WE2
PD1 PD2 PD3 PD4 PD5 PD6 PD7 PD8
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11
RFU1 RFU2 RFU3 RFU4 RFU5 RFU6 RFU7 RFU8 RFU9 RFU10 RFU11 RFU12 RFU13 RFU14 RFU15 RFU16
PE
RFU18 RFU19
8 BYTE SIMM
GND
(MEMORY ADDRESS)
6D8> IN MA<11..0>
6D8> IN MAB0
9C2> IN MWE0*
9C2> IN MWE1*
(WRITE)
(PRESENCE/SIZE)
MPD<7..0> OUT 5D8>
IBM
MEMORY SIMMS
DRAWING 06/01/94
SIZE B
REVISION A
DRWNG PART NUMBER
SHEET 4 OF 22

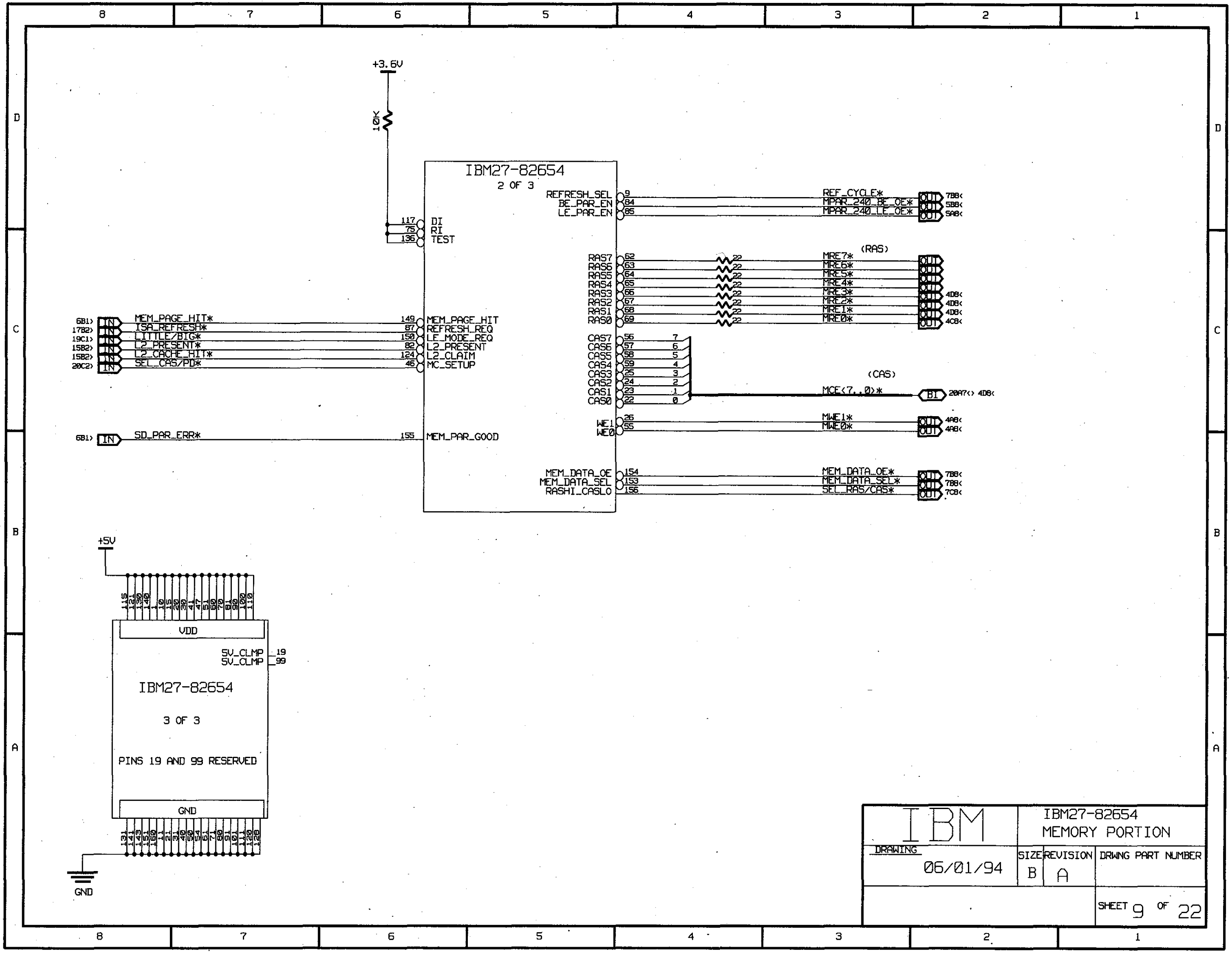

+5V
10K
4A1> IN MPD<7..0>
F244
PD4_RD*
(SIMM SIZE)
(X D BUS)
XD<7..0>
OUT 19A7<> 20C2>
F244
PD5_RD*
(SIMM PRESENCE)
ABT240
MSB
LSB
LSB
MSB
9D2> IN MPAR_240_BE_OE*
(BIG ENDIAN)
(CPU PARITY CORRECTED FOR ENDIAN MODE)
DP<7..0>
OUT 11B2<> 16A2<>
(MEMORY PARITY)
7D8<> 3D1<> IN MDP<7..0>
9D2> IN MPAR_240_LE_OE*
(LITTLE ENDIAN)
IBM
PARITY AND SIMM TYPE BUFFERS
DRAWING 06/01/94
SIZE B
REVISION A
DRWNG PART NUMBER
SHEET 5 OF 22

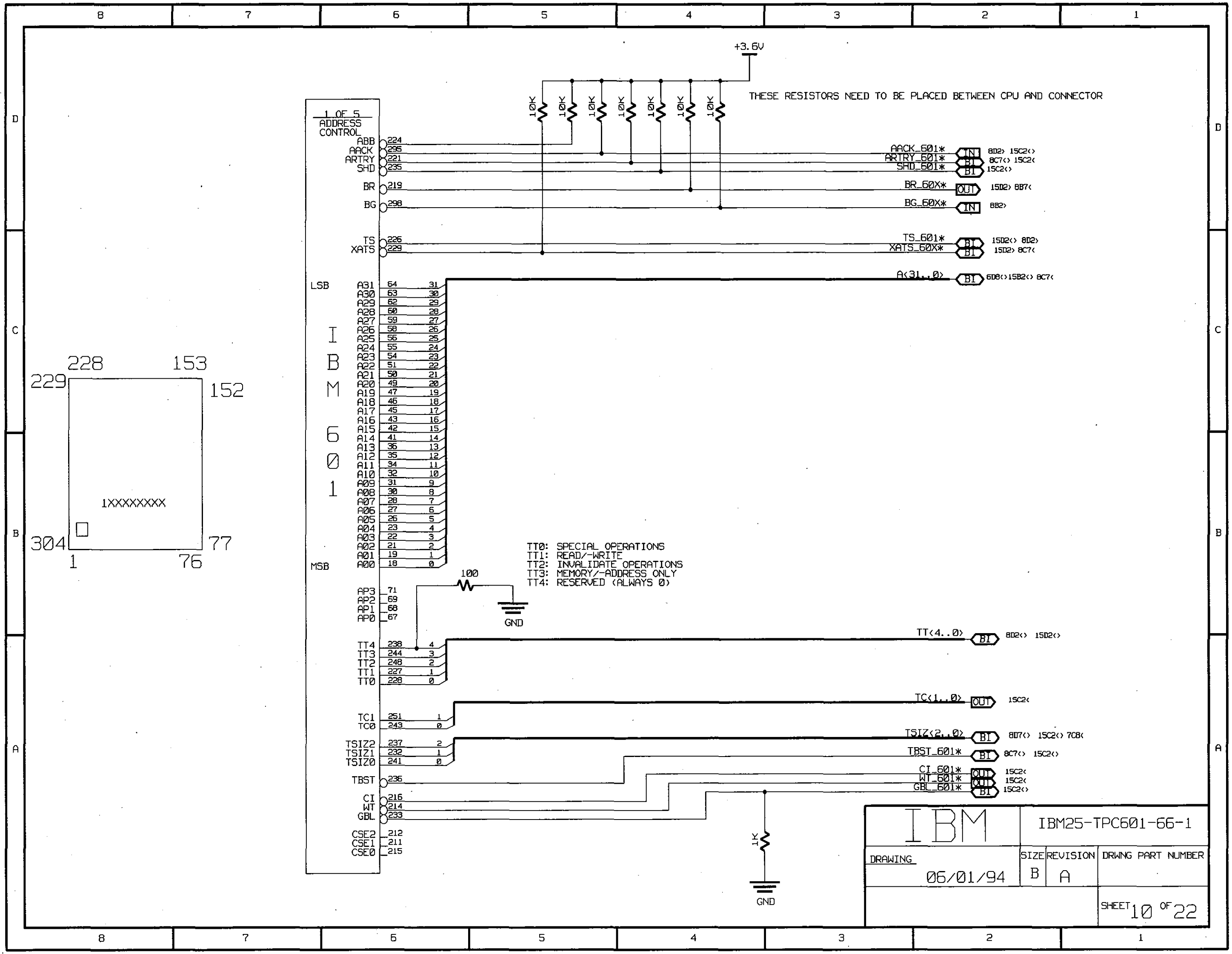

8C7< 10C2<> 15B2<> BI A<31..0>
(601 ADDR, MSB 0, LSB 31)
4A8< OUT MAB0
(MEMORY ADDRESS)
4B8< OUT MA<11..0>
(MEMORY ADDRESS)
11D2<> 16D2<> BI D<63..0>
(CPU DATA, MSB = 0)
IBM27-82653
5 OF 6
MSB
0 118 60X_A0
1 119 60X_A1
2 120 60X_A2
3 121 60X_A3
4 122 60X_A4
5 123 60X_A5
6 124 60X_A6
7 125 60X_A7
8 128 60X_A8
9 129 60X_A9
10 130 60X_A10
11 131 60X_A11
12 132 60X_A12
13 133 60X_A13
14 135 60X_A14
15 136 60X_A15
16 137 60X_A16
17 138 60X_A17
18 139 60X_A18
19 140 60X_A19
20 141 60X_A20
21 144 60X_A21
22 145 60X_A22
23 146 60X_A23
24 147 60X_A24
25 148 60X_A25
26 149 60X_A26
27 150 60X_A27
28 151 60X_A28
29 152 60X_A29
30 153 60X_A30
31 154 60X_A31
LSB
247 MEM_ADDR0_B
11 261 MEM_ADDR11
10 260 MEM_ADDR10
9 259 MEM_ADDR9
8 258 MEM_ADDR8
7 257 MEM_ADDR7
6 256 MEM_ADDR6
5 253 MEM_ADDR5
4 252 MEM_ADDR4
3 251 MEM_ADDR3
2 250 MEM_ADDR2
1 249 MEM_ADDR1
0 248 MEM_ADDR0
71 MEM_PAR_GOOD
64 MEM_PAGE_HIT
TEST 222
+3.6V
10K
MEM_PAGE_HIT*
OUT 9C8<
SD_PAR_ERR*
OUT 9B8<
IBM27-82653
2 OF 6
MSB
0 202 60X_D0
1 201 60X_D1
2 200 60X_D3
3 199 60X_D2
4 197 60X_D4
5 196 60X_D5
6 195 60X_D6
7 194 60X_D7
8 193 60X_D8
9 189 60X_D9
10 188 60X_D10
11 187 60X_D11
12 186 60X_D12
13 185 60X_D13
14 184 60X_D14
15 183 60X_D15
16 181 60X_D16
17 180 60X_D17
18 179 60X_D18
19 178 60X_D19
20 175 60X_D20
21 174 60X_D21
22 173 60X_D22
23 172 60X_D23
24 171 60X_D24
25 170 60X_D25
26 167 60X_D26
27 166 60X_D27
28 165 60X_D28
29 164 60X_D29
30 163 60X_D30
31 162 60X_D31
32 117 60X_D32
33 116 60X_D33
34 113 60X_D34
35 112 60X_D35
36 111 60X_D36
37 110 60X_D37
38 108 60X_D38
39 107 60X_D39
40 106 60X_D40
41 105 60X_D41
42 104 60X_D42
43 101 60X_D43
44 100 60X_D44
45 99 60X_D45
46 98 60X_D46
47 97 60X_D47
48 96 60X_D48
49 93 60X_D49
50 92 60X_D50
51 91 60X_D51
52 90 60X_D52
53 89 60X_D53
54 88 60X_D54
55 85 60X_D55
56 84 60X_D56
57 83 60X_D57
58 82 60X_D58
59 81 60X_D59
60 80 60X_D60
61 79 60X_D61
62 78 60X_D62
63 77 60X_D63
LSB
248
163
249
162
82653
1XXXXXXXX
304
77
1
76
+5V
VDD
IBM27-82653
6 OF 6
GND
GND
IBM
DRAWING
06/01/94
IBM27-82653
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 6 OF 22
8
7
6
5
4
3
2
1
D
C
B
A

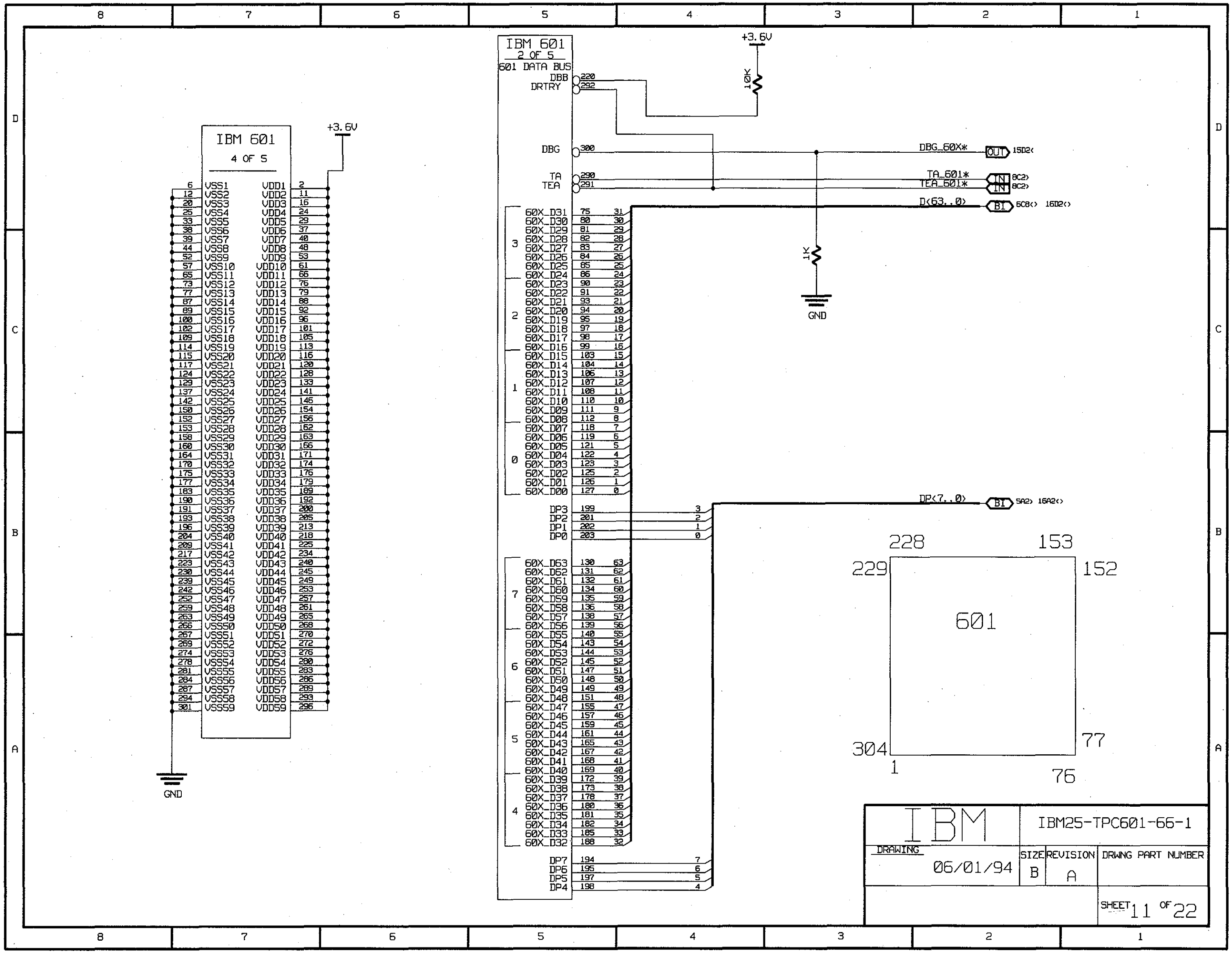


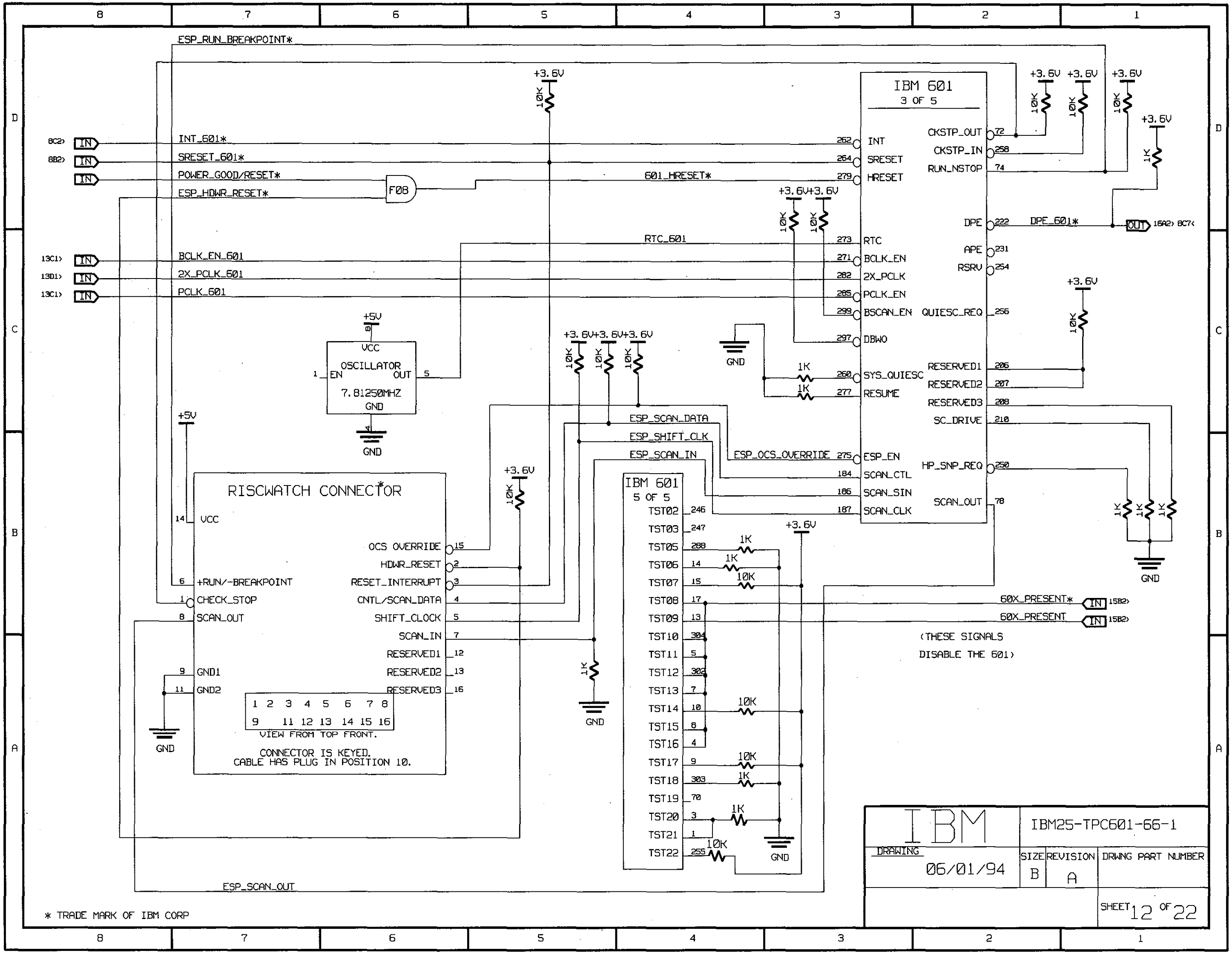
8
7
6
5
4
3
2
1
D
C
B
A
3D1<> BI MD<63..0>
(MEM DATA, MSB 63, LSB 0)
5A8< 3D1<> BI MDP<7..0>
(MEMORY PARITY)
20C2> IN CONTG_IO
8A2> IN 601_ADDR_SEL*
8A2> IN 601_DATA_SEL*
8A2> IN 601_ADDR_OE*
8A2> IN 601_DATA_OE*
8A2> IN NO_TRANS
10A2<> 8D7<> 15C2<> IN TSIZ<2..0>
8B2> IN ESEL
8A2> IN PCI_SEL*
8A2> IN PCI_OE*
14C2> IN SD_PCI_CLK
8A2> IN L_PCI_DATA*
8A2> IN PCI_AD_SEL*
9B2> IN SEL_RAS/CAS*
20C2> IN DRAM_TYPE
9B2> IN MEM_DATA_OE*
9B2> IN MEM_DATA_SEL*
9D2> IN REF_CYCLE*
8B2> IN ROM_CYCLE*
8B2> IN ROM_CLK/BURST*
8C2> IN 60X_TT_ERR*
8C2> IN MEM_PAR_ERR*
F08
8B2> IN ERR_ADDR_SEL*
8B2> IN ABORT_DATA_SEL*
17C2< BI AD<31..0>
(PCI ADDR/DATA, MSB 31, LSB 0)
8A7< OUT ND_AD_PAR
(EVEN PARITY)
IBM27-82653
1 OF 6
34 CONTG_IO
158 CPU_ADDR_SEL
168 CPU_DATA_SEL
159 CPU_ADDR_OE
192 CPU_DATA_OE
43 NO_TRANS
0 157 TSIZ0
1 156 TSIZ1
2 155 TSIZ2
262 LE_MODE_SEL
74 PCI_SEL
11 PCI_OE
70 PCI_CLK
67 L_PCI_DATA
75 ADDRHI_DATALO
203 RASHI_CASLO
65 DRAMX9HI_X10LO
7 MEM_DATA_OE
72 MEM_DATA_SEL
76 REFRESH_SEL
66 ROM_SEL
73 BURST_CLK
6 L_ERR_ADDR
223 ERR_ADDR_SEL
10 ALL_ONES_SEL
+3.6V
10K
10K
10K
IBM27-82653
4 OF 6
31 61 PCI_AD31 MSB
30 60 PCI_AD30
29 59 PCI_AD29
28 57 PCI_AD28
27 56 PCI_AD27
26 55 PCI_AD26
25 54 PCI_AD25
24 51 PCI_AD24
23 50 PCI_AD23
22 49 PCI_AD22
21 47 PCI_AD21
20 45 PCI_AD20
19 44 PCI_AD19
18 42 PCI_AD18
17 41 PCI_AD17
16 40 PCI_AD16
15 37 PCI_AD15
14 36 PCI_AD14
13 35 PCI_AD13
12 33 PCI_AD12
11 31 PCI_AD11
10 30 PCI_AD10
9 28 PCI_AD9
8 27 PCI_AD8
7 26 PCI_AD7
6 23 PCI_AD6
5 22 PCI_AD5
4 21 PCI_AD4
3 20 PCI_AD3
2 18 PCI_AD2
1 17 PCI_AD1
0 16 PCI_AD0 LSB
63 PCI_AD_PAR
IBM27-82653
3 OF 6
63 304 MEM_DATA63 MSB
62 303 MEM_DATA62
61 302 MEM_DATA61
60 300 MEM_DATA60
59 299 MEM_DATA59
58 298 MEM_DATA58
57 297 MEM_DATA57
56 296 MEM_DATA56
55 293 MEM_DATA55
54 292 MEM_DATA54
53 291 MEM_DATA53
52 290 MEM_DATA52
51 289 MEM_DATA51
50 288 MEM_DATA50
49 285 MEM_DATA49
48 284 MEM_DATA48
47 283 MEM_DATA47
46 282 MEM_DATA46
45 281 MEM_DATA45
44 280 MEM_DATA44
43 277 MEM_DATA43
42 276 MEM_DATA42
41 275 MEM_DATA41
40 274 MEM_DATA40
39 272 MEM_DATA39
38 271 MEM_DATA38
37 270 MEM_DATA37
36 269 MEM_DATA36
35 268 MEM_DATA35
34 265 MEM_DATA34
33 264 MEM_DATA33
32 263 MEM_DATA32
31 246 MEM_DATA31
30 245 MEM_DATA30
29 244 MEM_DATA29
28 243 MEM_DATA28
27 242 MEM_DATA27
26 241 MEM_DATA26
25 240 MEM_DATA25
24 237 MEM_DATA24
23 236 MEM_DATA23
22 235 MEM_DATA22
21 234 MEM_DATA21
20 233 MEM_DATA20
19 232 MEM_DATA19
18 231 MEM_DATA18
17 230 MEM_DATA17
16 229 MEM_DATA16
15 228 MEM_DATA15
14 226 MEM_DATA14
13 225 MEM_DATA13
12 224 MEM_DATA12
11 219 MEM_DATA11
10 218 MEM_DATA10
9 217 MEM_DATA9
8 216 MEM_DATA8
7 215 MEM_DATA7
6 214 MEM_DATA6
5 211 MEM_DATA5
4 210 MEM_DATA4
3 209 MEM_DATA3
2 208 MEM_DATA2
1 207 MEM_DATA1
0 206 MEM_DATA0 LSB
7 14 MEM_PAR7 MSB
6 13 MEM_PAR6
5 12 MEM_PAR5
4 5 MEM_PAR4
3 4 MEM_PAR3
2 3 MEM_PAR2
1 2 MEM_PAR1
0 1 MEM_PAR0 LSB
PAR7: MD(63:56)
PAR0: MD(7:0)
IBM
IBM27-82653
DRAWING 06/01/94
SIZE B
REVISION A
DRWNG PART NUMBER
SHEET 7 OF 22

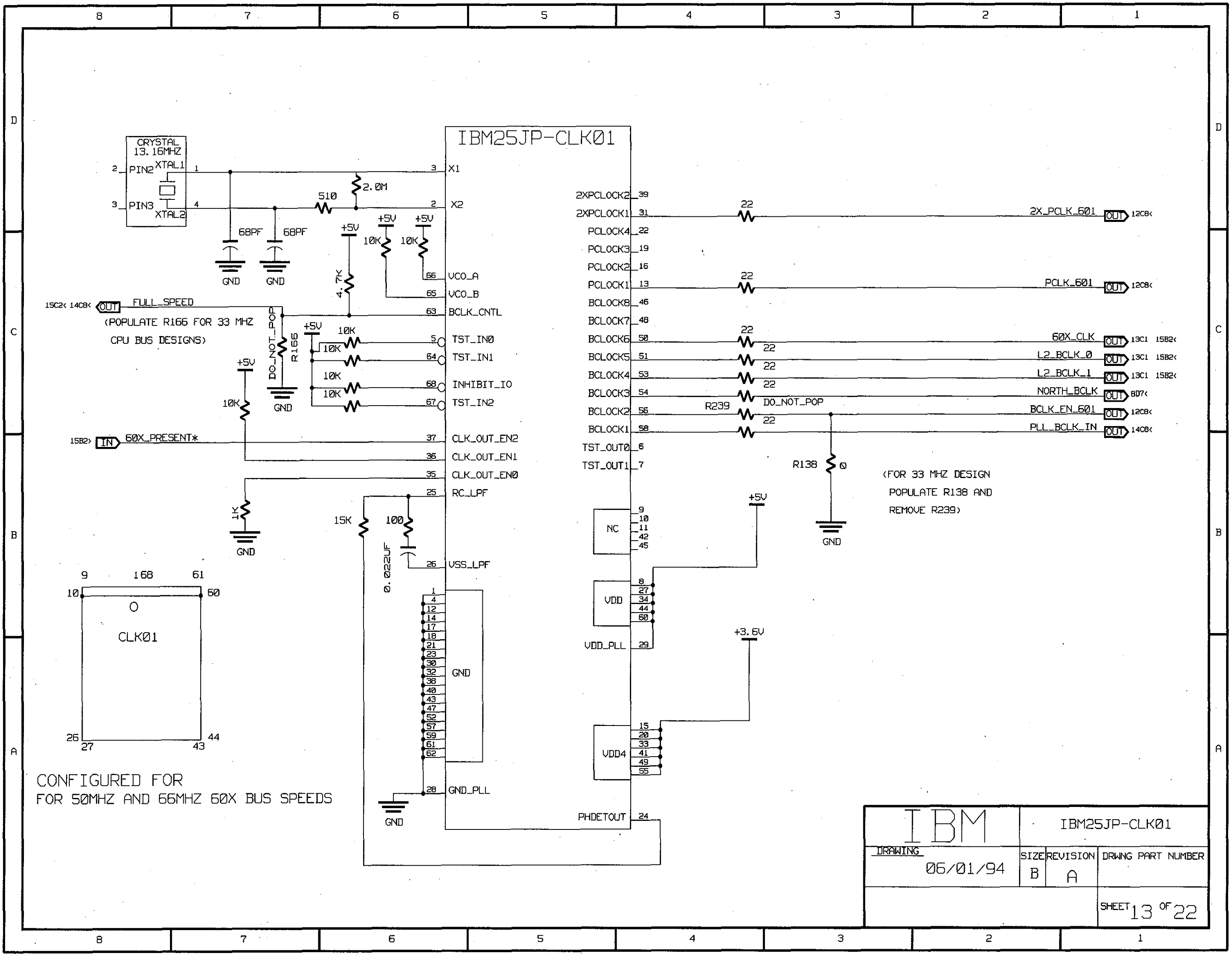

IBM27-82654
1 OF 3
POWER_GOOD/RESET*
RESET*
ND_PCI_CLK1
NORTH_BCLK
PCI_CLK
CPU_CLK
TSIZ<2..0>
TSIZ2
TSIZ1
TSIZ0
TBST_601*
ARTRY_601*
TBST
ARTRY
LSB (31)
(CPU_ADDRESS)
MSB (0)
A<31..0>
60X_A31
60X_A30
60X_A29
60X_A19
60X_A8
60X_A7
60X_A6
60X_A5
60X_A4
60X_A3
60X_A2
60X_A1
60X_A0
DPE_601*
XATS_60X*
DPE
XATS
MASK_TEA*
MASK_TEA
+3.6V
10K
BR_60X*
PCI_SIOREQ*
PCI_SCSI_REQ*
BR_L2*
PCI_REQ1*
PCI_REQ2*
CPU_REQ
IO_BRDG_REQ
PCI_REQ1
L2_CACHE_REQ
PCI_REQ4
PCI_REQ5
PCI_REQ2
PCI_REQ3
SIO_MEMREQ*
IO_BRDG_HOLD
ISA_MASTER*
ISA_MASTER
INT_FROM_SIO
NMI
IO_BRDG_IRQ
NMI_IRQ
ND_AD_PAR
ALT_RESET*
PCI_AD_PAR
SRESET_REQ
RESERVED
TS
TT0
TT1
TT2
TT3
TS_601*
TT<4..0>
AACK
TA
TEA
INT_CPU
AACK_601*
TA_601*
TEA_601*
INT_601*
TT_ERR
DPE_ERR
MEM_PAR_ERR
60X_TT_ERR*
60X_DPE_ERR*
MEM_PAR_ERR*
FRAME
TRDY
IRDY
STOP
DEVSEL
PCI_FRAME*
PCI_TRDY*
PCI_IRDY*
PCI_STOP*
PCI_DEVSEL*
C/BE3
C/BE2
C/BE1
C/BE0
PCI_C/BE<3..0>*
PCI_PAR
ROM_CS
ROM_OE
ROM_WE
ROM_SEL
BURST_CLK
ROM_CS*
ROM_OE*
ROM_WE*
ROM_CYCLE*
ROM_CLK/BURST*
CPU_GNT
L2_CACHE_GNT
IO_BRDG_GNT
PCI_GNT1
PCI_GNT4
PCI_GNT5
PCI_GNT2
PCI_GNT3
BG_60X*
BG_L2*
PCI_SIO_GNT*
PCI_SCSI_GNT*
PCI_GNT1*
PCI_GNT2*
SRESET_CPU
ERR_ADDR_SEL
ALL_ONES_SEL
LE_MODE_SEL
NO_TRANS
SRESET_601*
ERR_ADDR_SEL*
ABORT_DATA_SEL*
ESEL
NO_TRANS
CPU_ADDR_SEL
CPU_DATA_SEL
CPU_DATA_OE
CPU_ADDR_OE
PCI_SEL
PCI_OE
ADDRHI_DATALO
L_PCI_DATA
601_ADDR_SEL*
601_DATA_SEL*
601_DATA_OE*
601_ADDR_OE*
PCI_SEL*
PCI_OE*
PCI_AD_SEL*
L_PCI_DATA*
IN
BI
OUT
22
33
(ALL SERIES RESISTORS MUST BE PLACED
AS CLOSE AS POSSIBLE)
120
81
121
80
82654
1-XXXXXXX
160
1
40
41
IBM
IBM27-82654
CENTRAL CONTROL PORTION
DRAWING
06/01/94
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 8 OF 22
8 7 6 5 4 3 2 1
D C B A

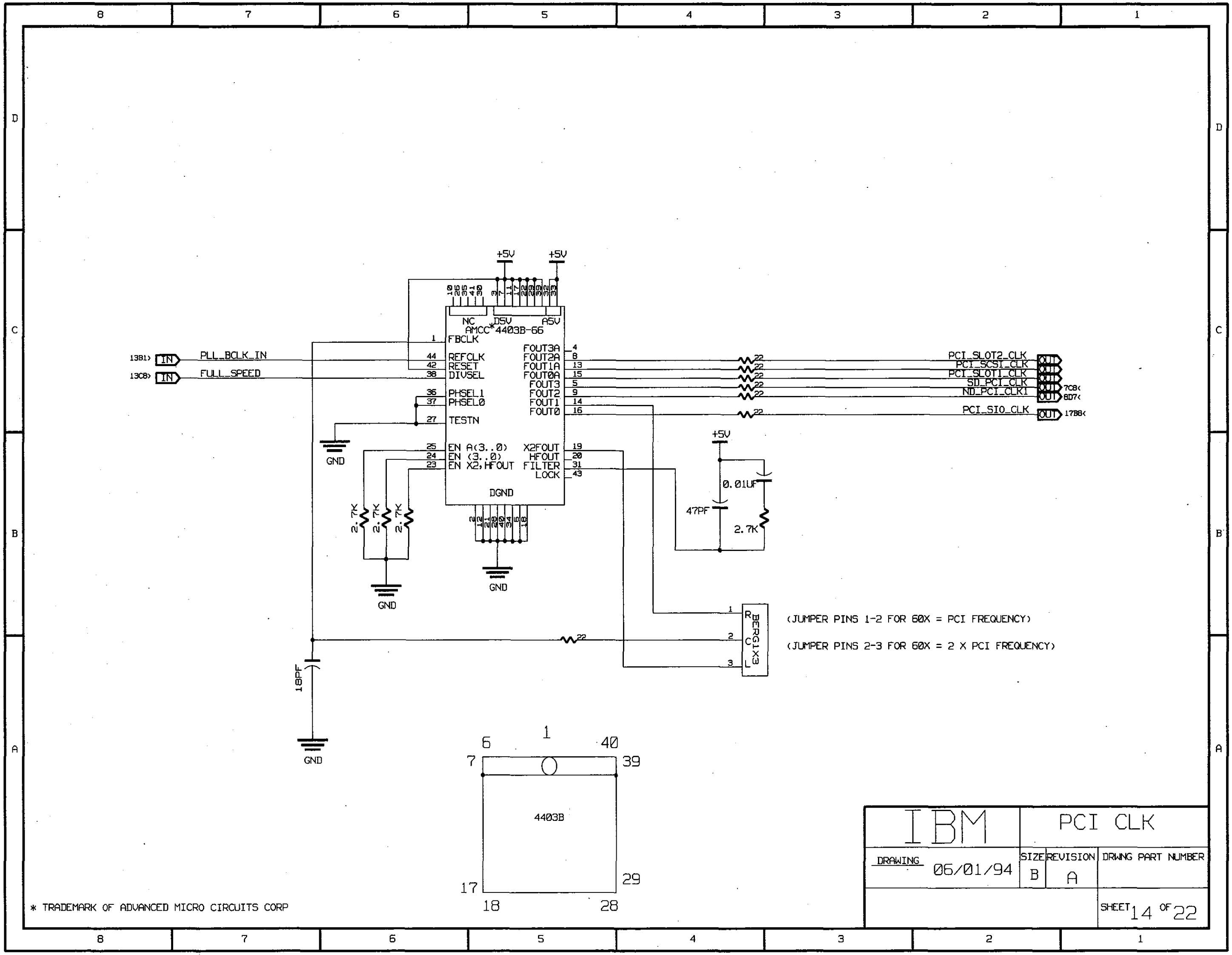

8
7
6
5
4
3
2
1
D
C
B
A
+3.6V
10K
IBM27-82654
2 OF 3
117
DI
75
RI
136
TEST
6B1>
IN
MEM_PAGE_HIT*
149
MEM_PAGE_HIT
17B2>
IN
ISA_REFRESH*
87
REFRESH_REQ
19C1>
IN
LITTLE/BIG*
150
LE_MODE_REQ
15B2>
IN
L2_PRESENT*
82
L2_PRESENT
15B2>
IN
L2_CACHE_HIT*
124
L2_CLAIM
20C2>
IN
SEL_CAS/PD*
46
MC_SETUP
6B1>
IN
SD_PAR_ERR*
155
MEM_PAR_GOOD
REFRESH_SEL
9
REF_CYCLE*
OUT
7B8<
BE_PAR_EN
84
MPAR_240_BE_OE*
OUT
5B8<
LE_PAR_EN
85
MPAR_240_LE_OE*
OUT
5A8<
(RAS)
RAS7
62
22
MRE7*
OUT
RAS6
63
22
MRE6*
OUT
RAS5
64
22
MRE5*
OUT
RAS4
65
22
MRE4*
OUT
RAS3
66
22
MRE3*
OUT
4D8<
RAS2
67
22
MRE2*
OUT
4D8<
RAS1
68
22
MRE1*
OUT
4D8<
RAS0
69
22
MRE0*
OUT
4C8<
CAS7
56
7
CAS6
57
6
CAS5
58
5
CAS4
59
4
CAS3
25
3
CAS2
24
2
CAS1
23
1
CAS0
22
0
(CAS)
MCE<7..0>*
BI
20A7<> 4D8<
WE1
26
MWE1*
OUT
4A8<
WE0
55
MWE0*
OUT
4A8<
MEM_DATA_OE
154
MEM_DATA_OE*
OUT
7B8<
MEM_DATA_SEL
153
MEM_DATA_SEL*
OUT
7B8<
RASHI_CASLO
156
SEL_RAS/CAS*
OUT
7C8<
+5V
115
121
130
140
1
10
15
20
30
41
47
51
60
70
81
90
100
110
VDD
5V_CLMP
19
5V_CLMP
99
IBM27-82654
3 OF 3
PINS 19 AND 99 RESERVED
GND
131
141
143
151
160
11
21
31
40
50
54
61
71
80
91
101
111
120
128
GND
IBM
IBM27-82654
MEMORY PORTION
DRAWING
06/01/94
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET
9
OF
22


+3.6V
10K
THESE RESISTORS NEED TO BE PLACED BETWEEN CPU AND CONNECTOR
1 OF 5
ADDRESS CONTROL
ABB 224
AACK 295
ARTRY 221
SHD 235
BR 219
BG 298
TS 226
XATS 229
AACK_601*
ARTRY_601*
SHD_601*
BR_60X*
BG_60X*
TS_601*
XATS_60X*
A<31..0>
IN
BI
OUT
LSB
MSB
IBM 601
A31 ... A00
TT0: SPECIAL OPERATIONS
TT1: READ/-WRITE
TT2: INVALIDATE OPERATIONS
TT3: MEMORY/-ADDRESS ONLY
TT4: RESERVED (ALWAYS 0)
100
GND
AP3 71
AP2 69
AP1 68
AP0 67
TT4 238
TT3 244
TT2 248
TT1 227
TT0 228
TT<4..0>
TC1 251
TC0 243
TC<1..0>
TSIZ2 237
TSIZ1 232
TSIZ0 241
TSIZ<2..0>
TBST 236
TBST_601*
CI 216
WT 214
GBL 233
CI_601*
WT_601*
GBL_601*
CSE2 212
CSE1 211
CSE0 215
1K
228
153
229
152
1XXXXXXXX
304
77
1
76
IBM
IBM25-TPC601-66-1
DRAWING
06/01/94
SIZE B
REVISION A
DRWNG PART NUMBER
SHEET 10 OF 22


IBM 601
4 OF 5
+3.6V
VSS1 6
VSS2 12
VSS3 20
VSS4 25
VSS5 33
VSS6 38
VSS7 39
VSS8 44
VSS9 52
VSS10 57
VSS11 65
VSS12 73
VSS13 77
VSS14 87
VSS15 89
VSS16 100
VSS17 102
VSS18 109
VSS19 114
VSS20 115
VSS21 117
VSS22 124
VSS23 129
VSS24 137
VSS25 142
VSS26 150
VSS27 152
VSS28 153
VSS29 158
VSS30 160
VSS31 164
VSS32 170
VSS33 175
VSS34 177
VSS35 183
VSS36 190
VSS37 191
VSS38 193
VSS39 196
VSS40 204
VSS41 209
VSS42 217
VSS43 223
VSS44 230
VSS45 239
VSS46 242
VSS47 252
VSS48 259
VSS49 263
VSS50 266
VSS51 267
VSS52 269
VSS53 274
VSS54 278
VSS55 281
VSS56 284
VSS57 287
VSS58 294
VSS59 301
VDD1 2
VDD2 11
VDD3 16
VDD4 24
VDD5 29
VDD6 37
VDD7 40
VDD8 48
VDD9 53
VDD10 61
VDD11 66
VDD12 76
VDD13 79
VDD14 88
VDD15 92
VDD16 96
VDD17 101
VDD18 105
VDD19 113
VDD20 116
VDD21 120
VDD22 128
VDD23 133
VDD24 141
VDD25 146
VDD26 154
VDD27 156
VDD28 162
VDD29 163
VDD30 166
VDD31 171
VDD32 174
VDD33 176
VDD34 179
VDD35 189
VDD36 192
VDD37 200
VDD38 205
VDD39 213
VDD40 218
VDD41 225
VDD42 234
VDD43 240
VDD44 245
VDD45 249
VDD46 253
VDD47 257
VDD48 261
VDD49 265
VDD50 268
VDD51 270
VDD52 272
VDD53 276
VDD54 280
VDD55 283
VDD56 286
VDD57 289
VDD58 293
VDD59 296
GND
IBM 601
2 OF 5
601 DATA BUS
DBB 220
DRTRY 292
+3.6V
10K
DBG 300
DBG_60X*
OUT
15D2<
TA 290
TEA 291
TA_601*
TEA_601*
IN
8C2>
IN
8C2>
D<63..0>
BI
5C8<> 16D2<>
1K
GND
3
60X_D31 75 31
60X_D30 80 30
60X_D29 81 29
60X_D28 82 28
60X_D27 83 27
60X_D26 84 26
60X_D25 85 25
60X_D24 86 24
2
60X_D23 90 23
60X_D22 91 22
60X_D21 93 21
60X_D20 94 20
60X_D19 95 19
60X_D18 97 18
60X_D17 98 17
60X_D16 99 16
1
60X_D15 103 15
60X_D14 104 14
60X_D13 106 13
60X_D12 107 12
60X_D11 108 11
60X_D10 110 10
60X_D09 111 9
60X_D08 112 8
0
60X_D07 118 7
60X_D06 119 6
60X_D05 121 5
60X_D04 122 4
60X_D03 123 3
60X_D02 125 2
60X_D01 126 1
60X_D00 127 0
DP<7..0>
BI
5A2> 16A2<>
DP3 199 3
DP2 201 2
DP1 202 1
DP0 203 0
7
60X_D63 130 63
60X_D62 131 62
60X_D61 132 61
60X_D60 134 60
60X_D59 135 59
60X_D58 136 58
60X_D57 138 57
60X_D56 139 56
6
60X_D55 140 55
60X_D54 143 54
60X_D53 144 53
60X_D52 145 52
60X_D51 147 51
60X_D50 148 50
60X_D49 149 49
60X_D48 151 48
5
60X_D47 155 47
60X_D46 157 46
60X_D45 159 45
60X_D44 161 44
60X_D43 165 43
60X_D42 167 42
60X_D41 168 41
60X_D40 169 40
4
60X_D39 172 39
60X_D38 173 38
60X_D37 178 37
60X_D36 180 36
60X_D35 181 35
60X_D34 182 34
60X_D33 185 33
60X_D32 188 32
DP7 194 7
DP6 195 6
DP5 197 5
DP4 198 4
228
153
229
152
601
304
77
1
76
IBM
IBM25-TPC601-66-1
DRAWING
06/01/94
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 11 OF 22
8
7
6
5
4
3
2
1
D
C
B
A

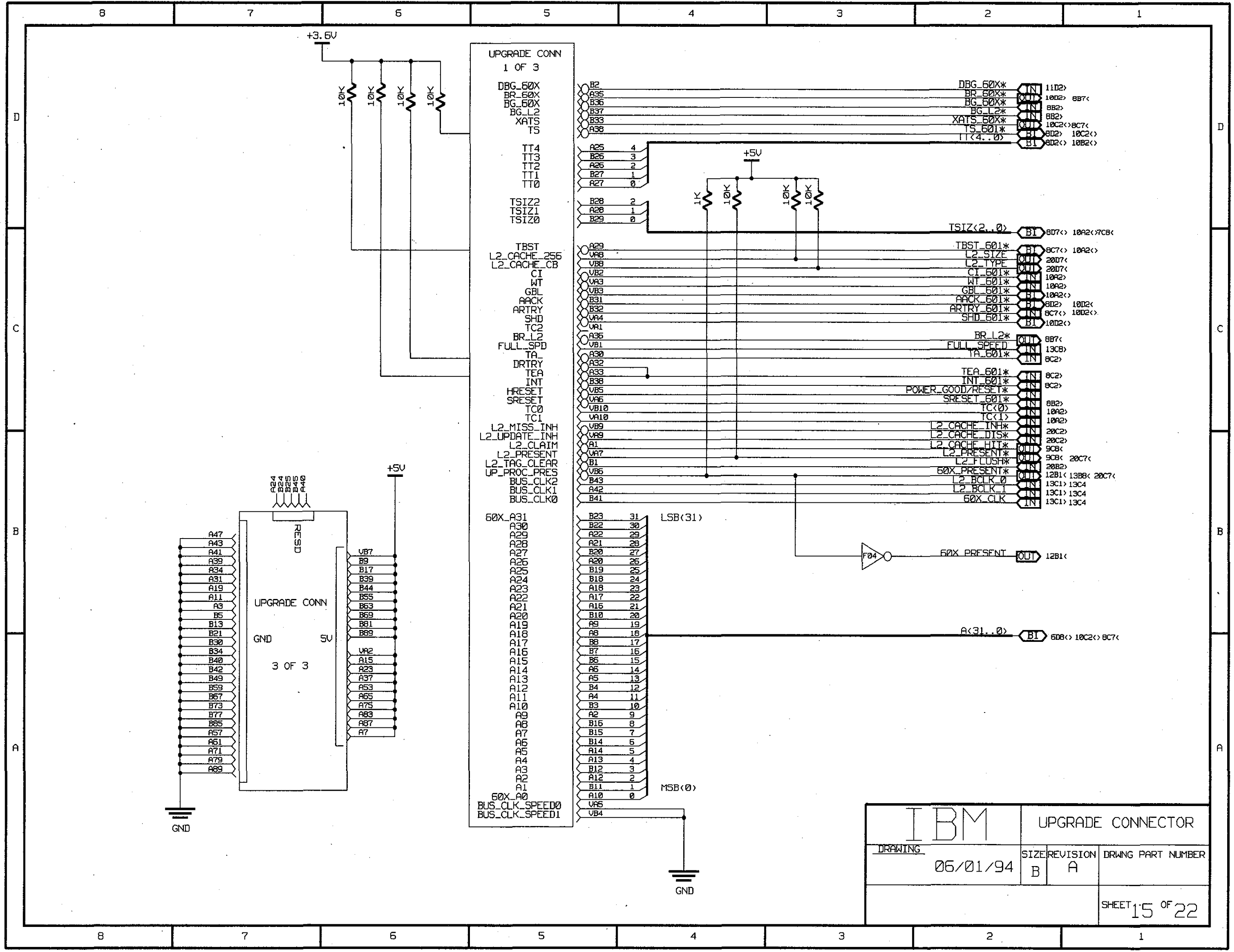

ESP_RUN_BREAKPOINT*
INT_601*
SRESET_601*
POWER_GOOD/RESET*
ESP_HDWR_RESET*
F08
601_HRESET*
RTC_601
BCLK_EN_601
2X_PCLK_601
PCLK_601
IBM 601
3 OF 5
INT
SRESET
HRESET
RTC
BCLK_EN
2X_PCLK
PCLK_EN
BSCAN_EN
DBWO
SYS_QUIESC
RESUME
ESP_EN
SCAN_CTL
SCAN_SIN
SCAN_CLK
CKSTP_OUT
CKSTP_IN
RUN_NSTOP
DPE
APE
RSRV
QUIESC_REQ
RESERVED1
RESERVED2
RESERVED3
SC_DRIVE
HP_SNP_REQ
SCAN_OUT
DPE_601*
OUT
+3.6V
10K
1K
GND
+5V
VCC
OSCILLATOR
EN
OUT
7.81250MHZ
ESP_SCAN_DATA
ESP_SHIFT_CLK
ESP_SCAN_IN
ESP_OCS_OVERRIDE
RISCWATCH CONNECTOR*
VCC
OCS OVERRIDE
HDWR_RESET
RESET_INTERRUPT
+RUN/-BREAKPOINT
CHECK_STOP
CNTL/SCAN_DATA
SCAN_OUT
SHIFT_CLOCK
SCAN_IN
RESERVED1
RESERVED2
RESERVED3
GND1
GND2
1 2 3 4 5 6 7 8
9 11 12 13 14 15 16
VIEW FROM TOP FRONT.
CONNECTOR IS KEYED.
CABLE HAS PLUG IN POSITION 10.
IBM 601
5 OF 5
TST02
TST03
TST05
TST06
TST07
TST08
TST09
TST10
TST11
TST12
TST13
TST14
TST15
TST16
TST17
TST18
TST19
TST20
TST21
TST22
60X_PRESENT*
60X_PRESENT
IN
(THESE SIGNALS
DISABLE THE 601)
ESP_SCAN_OUT
* TRADE MARK OF IBM CORP
IBM
DRAWING
06/01/94
IBM25-TPC601-66-1
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 12 OF 22

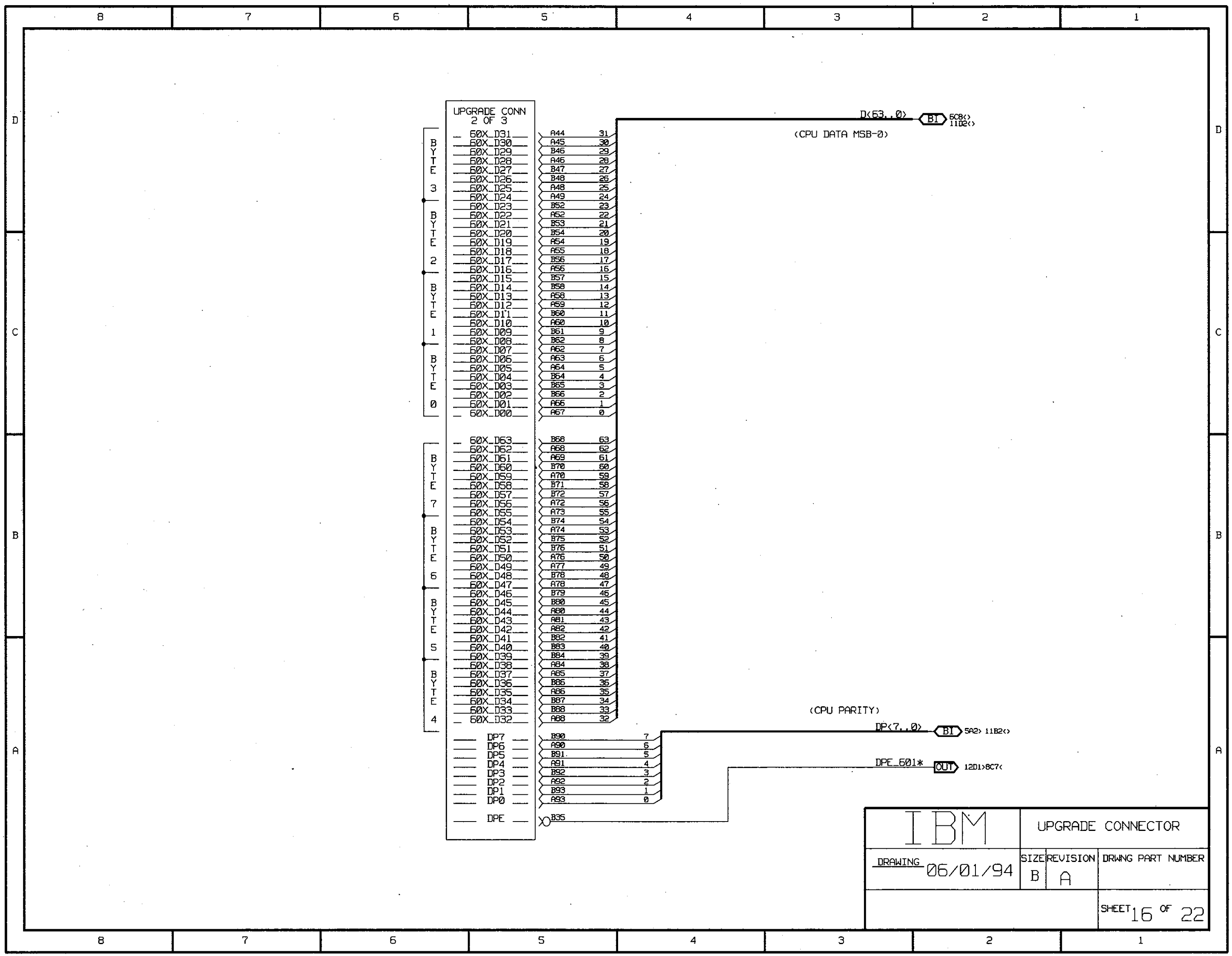

IBM25JP-CLK01
CRYSTAL 13.16MHZ
PIN2 XTAL1
PIN3 XTAL2
2.0M
510
68PF
68PF
GND
+5V
4.7K
10K
X1
X2
VCO_A
VCO_B
BCLK_CNTL
TST_IN0
TST_IN1
INHIBIT_IO
TST_IN2
CLK_OUT_EN2
CLK_OUT_EN1
CLK_OUT_EN0
RC_LPF
VSS_LPF
GND_PLL
15C2< 14C8< OUT FULL_SPEED
(POPULATE R166 FOR 33 MHZ CPU BUS DESIGNS)
DO_NOT_POP R166
15B2> IN 60X_PRESENT*
1K
15K
100
0.022UF
2XPCLOCK2
2XPCLOCK1
PCLOCK4
PCLOCK3
PCLOCK2
PCLOCK1
BCLOCK8
BCLOCK7
BCLOCK6
BCLOCK5
BCLOCK4
BCLOCK3
BCLOCK2
BCLOCK1
TST_OUT0
TST_OUT1
NC
VDD
VDD_PLL
VDD4
PHDETOUT
22
R239
DO_NOT_POP
2X_PCLK_601 OUT 12C8<
PCLK_601 OUT 12C8<
60X_CLK OUT 13C1 15B2<
L2_BCLK_0 OUT 13C1 15B2<
L2_BCLK_1 OUT 13C1 15B2<
NORTH_BCLK OUT 8D7<
BCLK_EN_601 OUT 12C8<
PLL_BCLK_IN OUT 14C8<
R138 0
(FOR 33 MHZ DESIGN POPULATE R138 AND REMOVE R239)
+3.6V
CLK01
CONFIGURED FOR
FOR 50MHZ AND 66MHZ 60X BUS SPEEDS
IBM
IBM25JP-CLK01
DRAWING 06/01/94
SIZE B
REVISION A
DRWNG PART NUMBER
SHEET 13 OF 22

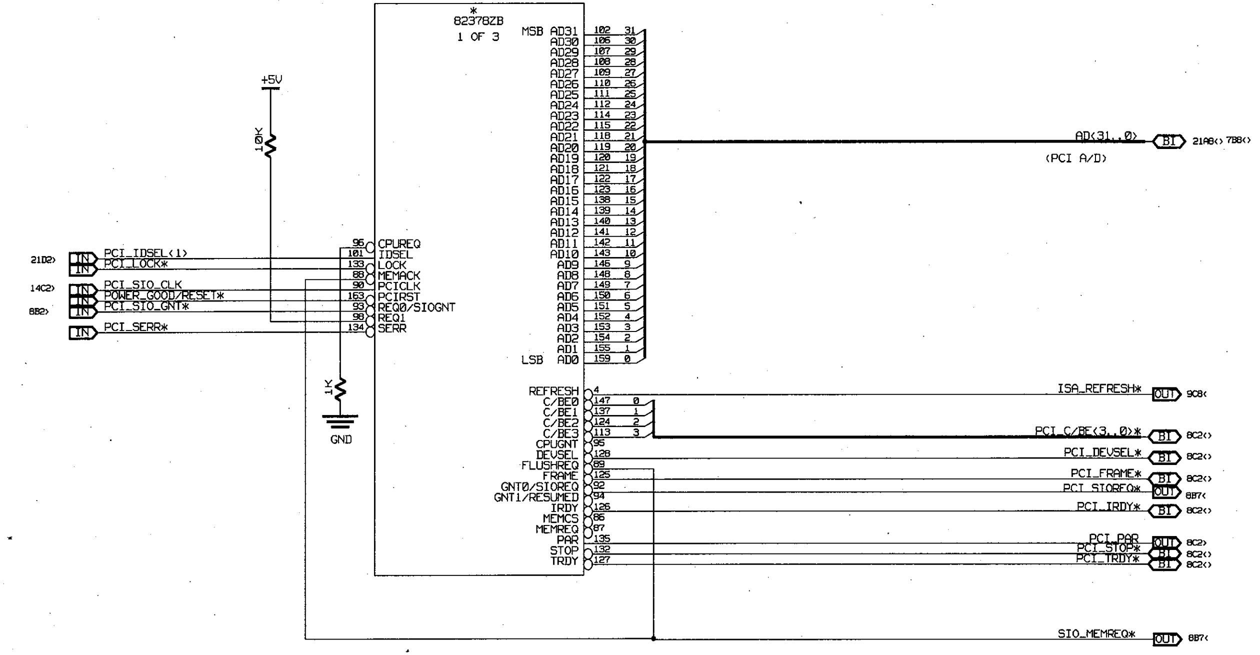


* TRADEMARK OF ADVANCED MICRO CIRCUITS CORP

IBM	PCI CLK		
DRAWING 06/01/94	SIZE B	REVISION A	DRWNG PART NUMBER
			SHEET 14 OF 22


+3.6V
10K
10K
10K
10K
UPGRADE CONN
1 OF 3
DBG_60X
BR_60X
BG_60X
BG_L2
XATS
TS
TT4
TT3
TT2
TT1
TT0
TSIZ2
TSIZ1
TSIZ0
TBST
L2_CACHE_256
L2_CACHE_CB
CI
WT
GBL
AACK
ARTRY
SHD
TC2
BR_L2
FULL_SPD
TA_
DRTRY
TEA
INT
HRESET
SRESET
TC0
TC1
L2_MISS_INH
L2_UPDATE_INH
L2_CLAIM
L2_PRESENT
L2_TAG_CLEAR
UP_PROC_PRES
BUS_CLK2
BUS_CLK1
BUS_CLK0
60X_A31
A30 … A1
60X_A0
BUS_CLK_SPEED0
BUS_CLK_SPEED1
+5V
1K
10K
10K
10K
DBG_60X*
BR_60X*
BG_60X*
BG_L2*
XATS_60X*
TS_601*
TT<4..0>
TSIZ<2..0>
TBST_601*
L2_SIZE
L2_TYPE
CI_601*
WT_601*
GBL_601*
AACK_601*
ARTRY_601*
SHD_601*
BR_L2*
FULL_SPEED
TA_601*
TEA_601*
INT_601*
POWER_GOOD/RESET*
SRESET_601*
TC<0>
TC<1>
L2_CACHE_INH*
L2_CACHE_DIS*
L2_CACHE_HIT*
L2_PRESENT*
L2_FLUSH*
60X_PRESENT*
L2_BCLK_0
L2_BCLK_1
60X_CLK
IN
OUT
BI
11D2>
10D2> 8B7<
8B2>
8B2>
10C2<>8C7<
8D2> 10C2<>
8D2<> 10B2<>
8D7<> 10A2<>7C8<
8C7<> 10A2<>
20D7<
20D7<
10A2>
10A2>
10A2<>
8D2> 10D2<
8C7<> 10D2<>
10D2<>
8B7<
13C8>
8C2>
8C2>
8C2>
8B2>
10A2>
10A2>
20C2>
20C2>
9C8<
9C8< 20C7<
20B2>
12B1< 13B8< 20C7<
13C1> 13C4
13C1> 13C4
13C1> 13C4
LSB(31)
MSB(0)
F04
60X_PRESENT
OUT 12B1<
A<31..0>
BI 6D8<> 10C2<> 8C7<
A24 B24 B25 B45 A40
RESD
UPGRADE CONN
GND
5V
3 OF 3
GND
GND
IBM
UPGRADE CONNECTOR
DRAWING
06/01/94
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 15 OF 22
8 7 6 5 4 3 2 1
D C B A

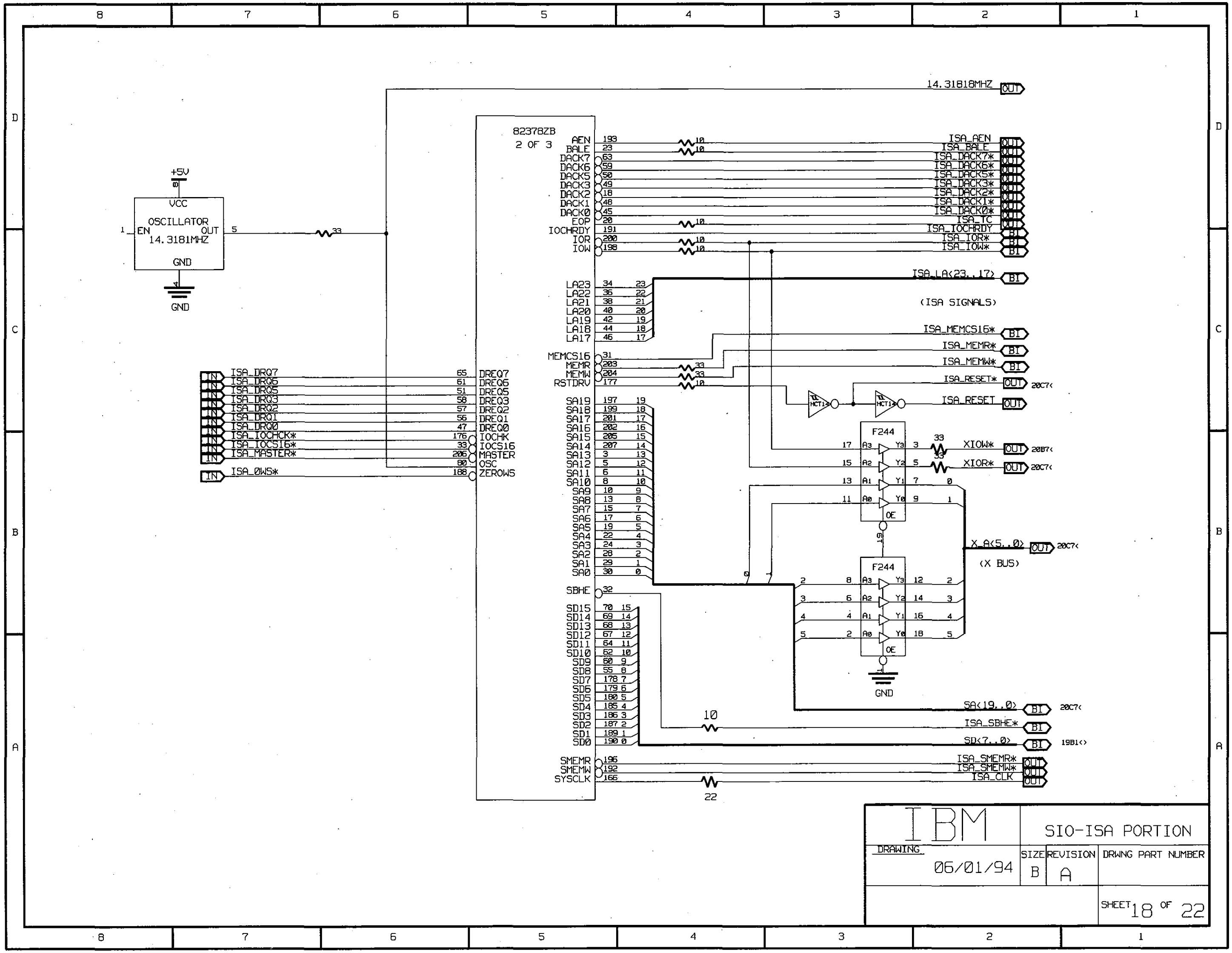

8
7
6
5
4
3
2
1
D
C
B
A
UPGRADE CONN
2 OF 3
BYTE 3
BYTE 2
BYTE 1
BYTE 0
BYTE 7
BYTE 6
BYTE 5
BYTE 4
60X_D31 A44 31
60X_D30 A45 30
60X_D29 B46 29
60X_D28 A46 28
60X_D27 B47 27
60X_D26 B48 26
60X_D25 A48 25
60X_D24 A49 24
60X_D23 B52 23
60X_D22 A52 22
60X_D21 B53 21
60X_D20 B54 20
60X_D19 A54 19
60X_D18 A55 18
60X_D17 B56 17
60X_D16 A56 16
60X_D15 B57 15
60X_D14 B58 14
60X_D13 A58 13
60X_D12 A59 12
60X_D11 B60 11
60X_D10 A60 10
60X_D09 B61 9
60X_D08 B62 8
60X_D07 A62 7
60X_D06 A63 6
60X_D05 A64 5
60X_D04 B64 4
60X_D03 B65 3
60X_D02 B66 2
60X_D01 A66 1
60X_D00 A67 0
60X_D63 B68 63
60X_D62 A68 62
60X_D61 A69 61
60X_D60 B70 60
60X_D59 A70 59
60X_D58 B71 58
60X_D57 B72 57
60X_D56 A72 56
60X_D55 A73 55
60X_D54 B74 54
60X_D53 A74 53
60X_D52 B75 52
60X_D51 B76 51
60X_D50 A76 50
60X_D49 A77 49
60X_D48 B78 48
60X_D47 A78 47
60X_D46 B79 46
60X_D45 B80 45
60X_D44 A80 44
60X_D43 A81 43
60X_D42 A82 42
60X_D41 B82 41
60X_D40 B83 40
60X_D39 B84 39
60X_D38 A84 38
60X_D37 A85 37
60X_D36 B86 36
60X_D35 A86 35
60X_D34 B87 34
60X_D33 B88 33
60X_D32 A88 32
DP7 B90 7
DP6 A90 6
DP5 B91 5
DP4 A91 4
DP3 B92 3
DP2 A92 2
DP1 B93 1
DP0 A93 0
DPE B35
D<63..0>
BI 5C8<> 11D2<>
(CPU DATA MSB-0)
(CPU PARITY)
DP<7..0>
BI 5A2> 11B2<>
DPE_601*
OUT 12D1>8C7<
IBM
UPGRADE CONNECTOR
DRAWING 06/01/94
SIZE B
REVISION A
DRWNG PART NUMBER
SHEET 16 OF 22

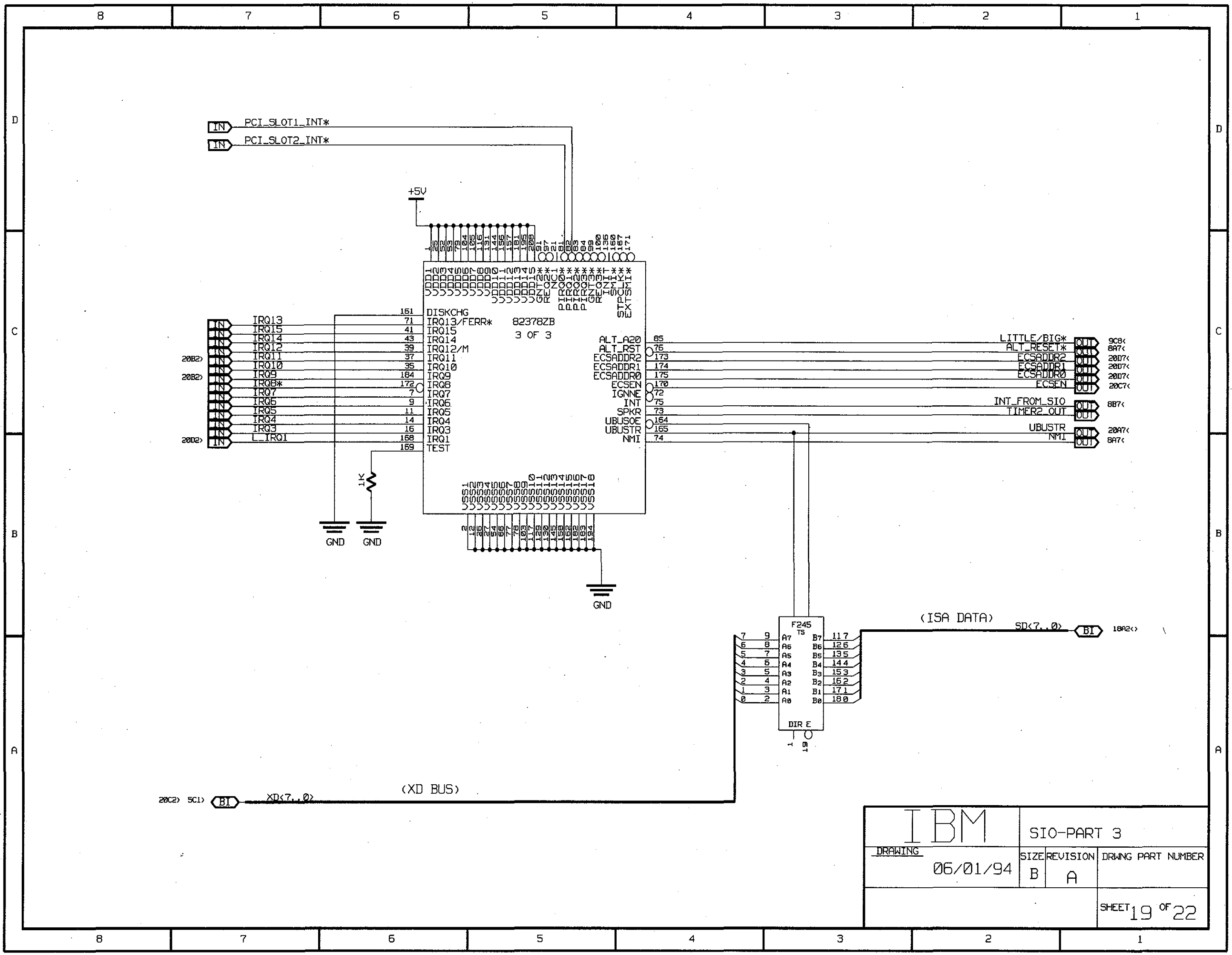


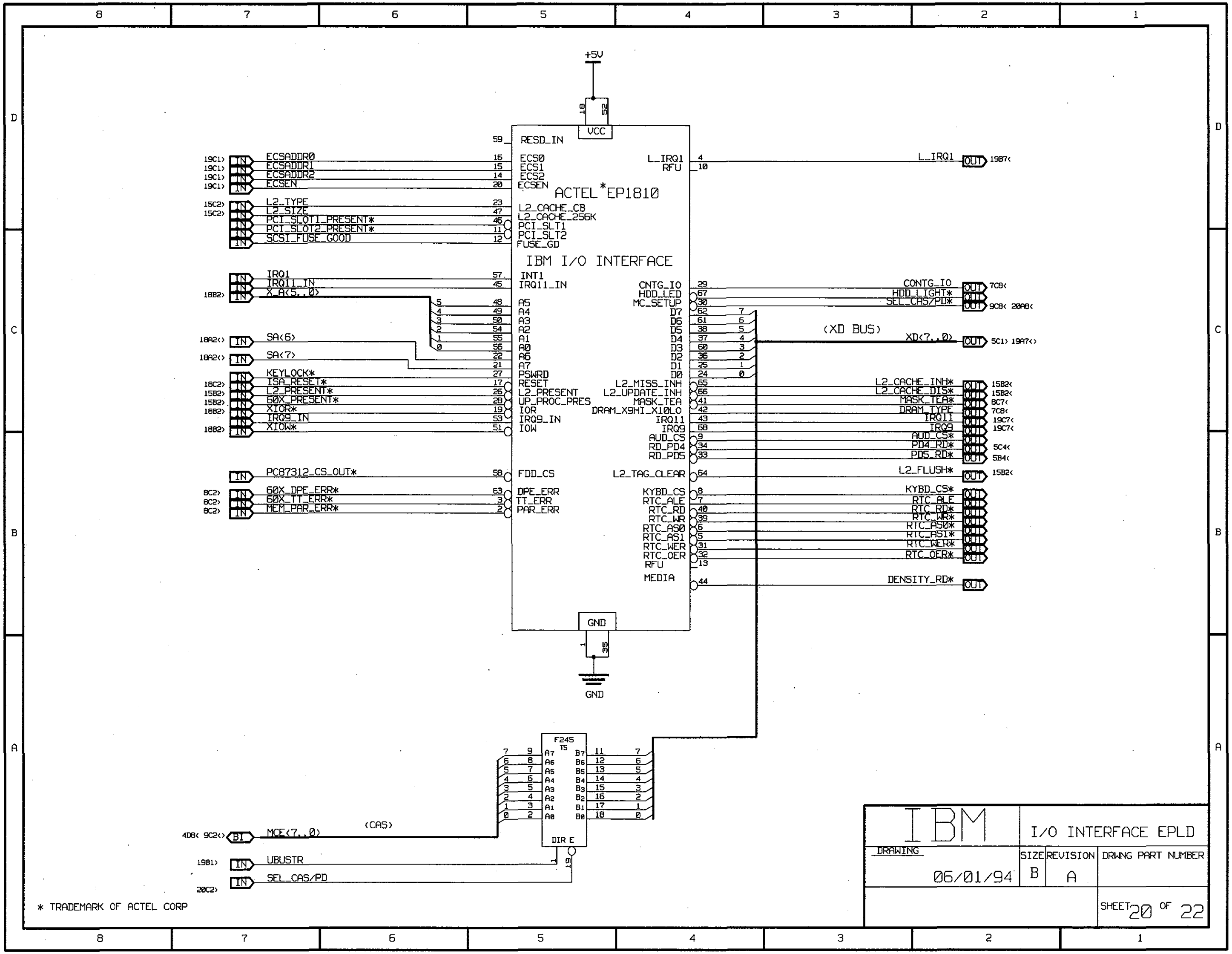
REQ1 IS HIGH DURING RESET
SO SIO DRIVES AD, C/BE, PAR
WHEN PCIRESET ASSERTED

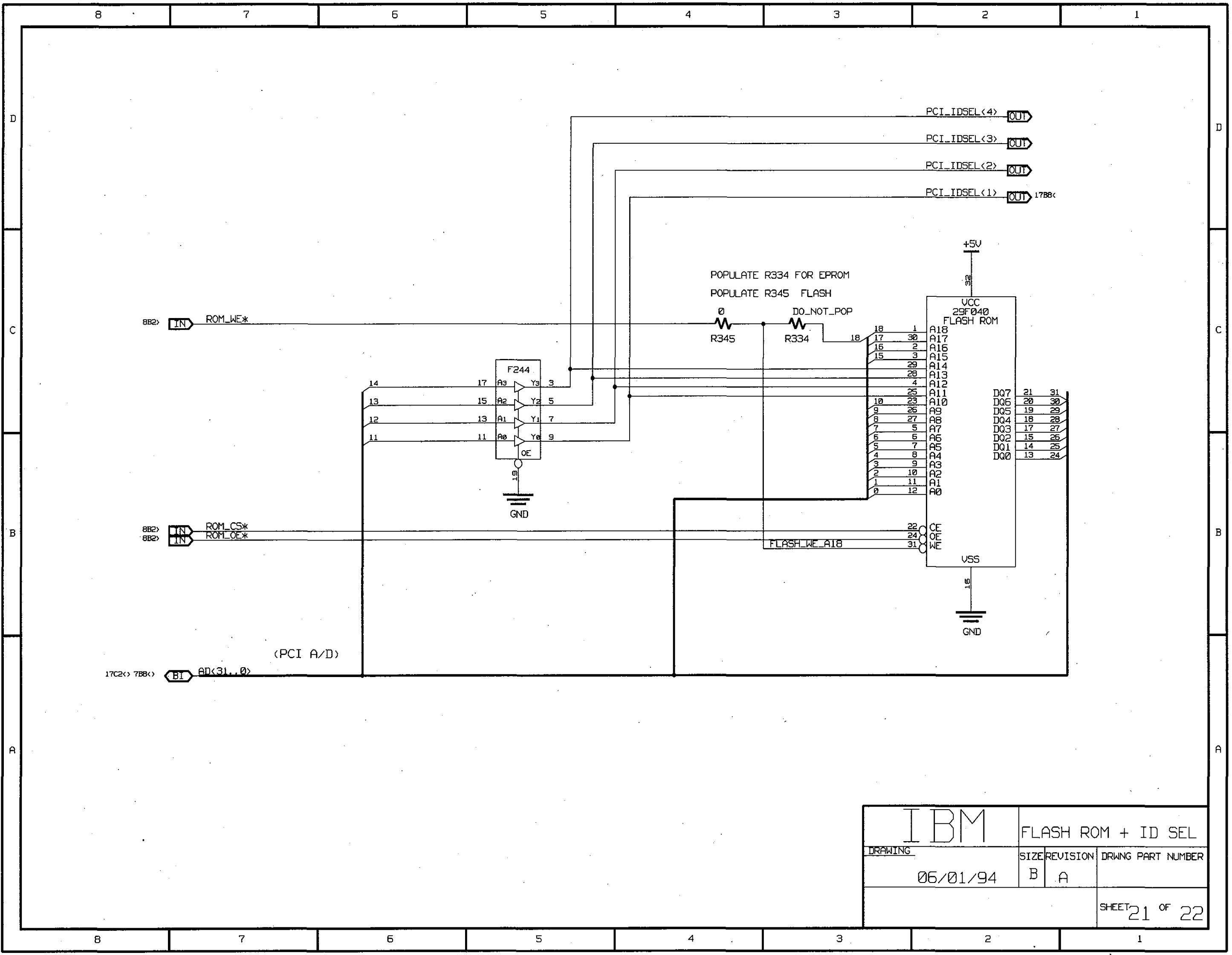
* TRADEMARK OF INTEL CORP

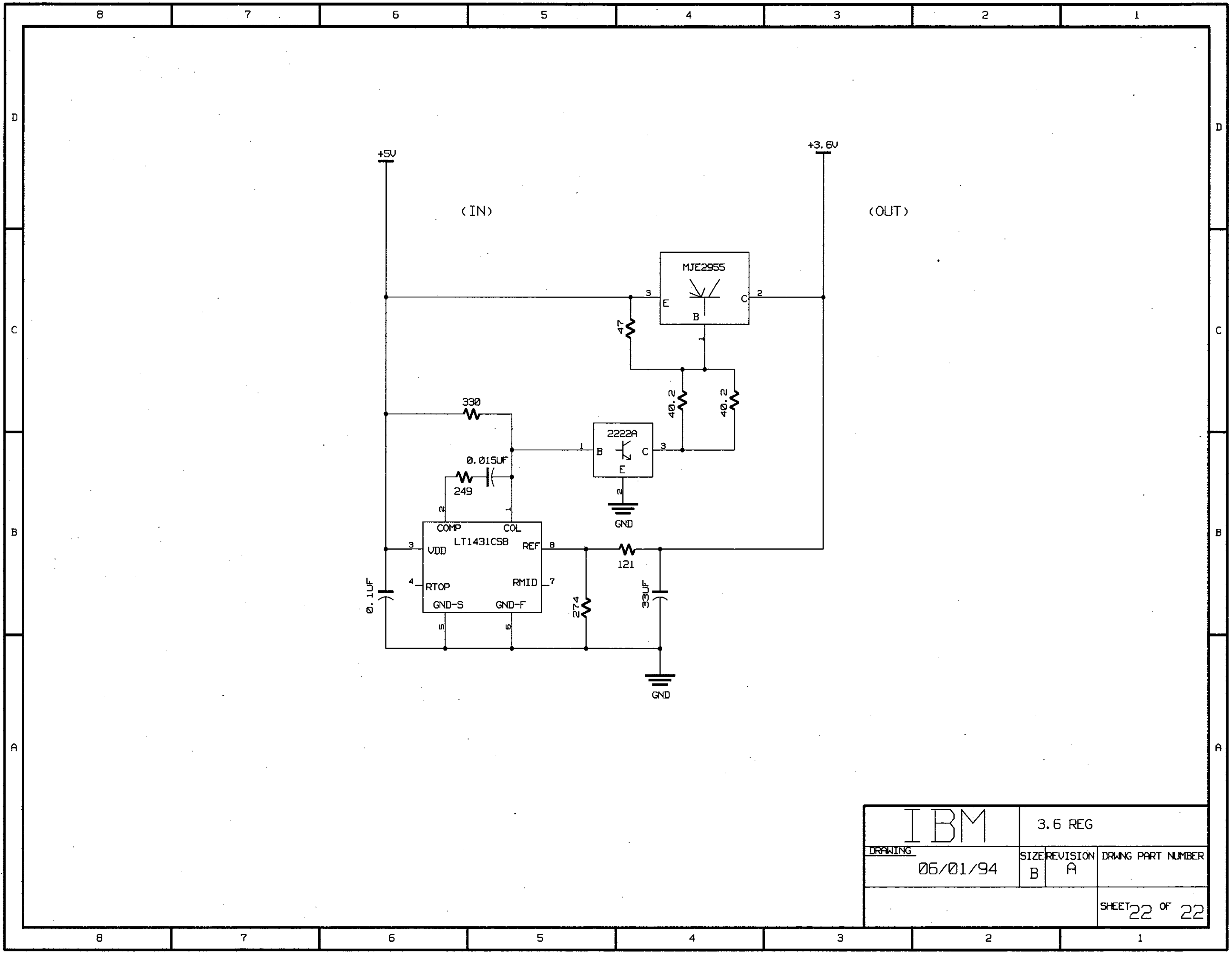
IBM	SIO-PCI PORTION		
DRAWING 06/01/94	SIZE B	REVISION A	DRWNG PART NUMBER
			SHEET 17 OF 22


+5V
VCC
OSCILLATOR
EN
OUT
14.3181MHZ
GND
GND
33
14.31818MHZ
OUT
82378ZB
2 OF 3
AEN
BALE
DACK7
DACK6
DACK5
DACK3
DACK2
DACK1
DACK0
EOP
IOCHRDY
IOR
IOW
ISA_AEN
ISA_BALE
ISA_DACK7*
ISA_DACK6*
ISA_DACK5*
ISA_DACK3*
ISA_DACK2*
ISA_DACK1*
ISA_DACK0*
ISA_TC
ISA_IOCHRDY
ISA_IOR*
ISA_IOW*
ISA_LA<23..17>
(ISA SIGNALS)
LA23
LA22
LA21
LA20
LA19
LA18
LA17
MEMCS16
MEMR
MEMW
RSTDRV
ISA_MEMCS16*
ISA_MEMR*
ISA_MEMW*
ISA_RESET*
ISA_RESET
20C7<
HCT14
ISA_DRQ7
ISA_DRQ6
ISA_DRQ5
ISA_DRQ3
ISA_DRQ2
ISA_DRQ1
ISA_DRQ0
ISA_IOCHCK*
ISA_IOCS16*
ISA_MASTER*
ISA_0WS*
IN
DREQ7
DREQ6
DREQ5
DREQ3
DREQ2
DREQ1
DREQ0
IOCHK
IOCS16
MASTER
OSC
ZEROWS
SA19
SA18
SA17
SA16
SA15
SA14
SA13
SA12
SA11
SA10
SA9
SA8
SA7
SA6
SA5
SA4
SA3
SA2
SA1
SA0
SBHE
F244
XIOW*
XIOR*
20B7<
20C7<
X_A<5..0>
(X BUS)
OE
GND
SA<19..0>
ISA_SBHE*
SD<7..0>
19B1<>
BI
SD15
SD14
SD13
SD12
SD11
SD10
SD9
SD8
SD7
SD6
SD5
SD4
SD3
SD2
SD1
SD0
SMEMR
SMEMW
SYSCLK
ISA_SMEMR*
ISA_SMEMW*
ISA_CLK
10
22
IBM
SIO-ISA PORTION
DRAWING
06/01/94
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 18 OF 22


PCI_SLOT1_INT*
PCI_SLOT2_INT*
IN
+5V
82378ZB
3 OF 3
DISKCHG
IRQ13/FERR*
IRQ15
IRQ14
IRQ12/M
IRQ11
IRQ10
IRQ9
IRQ8
IRQ7
IRQ6
IRQ5
IRQ4
IRQ3
IRQ1
TEST
IRQ13
IRQ15
IRQ14
IRQ12
IRQ11
IRQ10
IRQ9
IRQ8*
IRQ7
IRQ6
IRQ5
IRQ4
IRQ3
L_IRQ1
20B2>
20B2>
20D2>
1K
GND
GND
GND
ALT_A20
ALT_RST
ECSADDR2
ECSADDR1
ECSADDR0
ECSEN
IGNNE
INT
SPKR
UBUSOE
UBUSTR
NMI
LITTLE/BIG*
ALT_RESET*
ECSADDR2
ECSADDR1
ECSADDR0
ECSEN
INT_FROM_SIO
TIMER2_OUT
UBUSTR
NMI
OUT
9C8<
8A7<
20D7<
20D7<
20D7<
20C7<
8B7<
20A7<
8A7<
F245
TS
DIR E
(ISA DATA)
SD<7..0>
BI
18A2<>
(XD BUS)
XD<7..0>
20C2> 5C1>
IBM
SIO-PART 3
DRAWING
06/01/94
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 19 OF 22


+5V
VCC
18
52
RESD_IN
ACTEL*EP1810
IBM I/O INTERFACE
ECSADDR0
ECSADDR1
ECSADDR2
ECSEN
ECS0
ECS1
ECS2
ECSEN
L2_TYPE
L2_SIZE
PCI_SLOT1_PRESENT*
PCI_SLOT2_PRESENT*
SCSI_FUSE_GOOD
L2_CACHE_CB
L2_CACHE_256K
PCI_SLT1
PCI_SLT2
FUSE_GD
IRQ1
IRQ11_IN
X_A<5..0>
INT1
IRQ11_IN
A5
A4
A3
A2
A1
A0
A6
A7
SA<6>
SA<7>
KEYLOCK*
ISA_RESET*
L2_PRESENT*
60X_PRESENT*
XIOR*
IRQ9_IN
XIOW*
PSWRD
RESET
L2_PRESENT
UP_PROC_PRES
IOR
IRQ9_IN
IOW
PC87312_CS_OUT*
FDD_CS
60X_DPE_ERR*
60X_TT_ERR*
MEM_PAR_ERR*
DPE_ERR
TT_ERR
PAR_ERR
L_IRQ1
RFU
CNTG_IO
HDD_LED
MC_SETUP
D7
D6
D5
D4
D3
D2
D1
D0
(XD BUS)
XD<7..0>
L2_MISS_INH
L2_UPDATE_INH
MASK_TEA
DRAM_X9HI_X10LO
IRQ11
IRQ9
AUD_CS
RD_PD4
RD_PD5
L2_TAG_CLEAR
KYBD_CS
RTC_ALE
RTC_RD
RTC_WR
RTC_AS0
RTC_AS1
RTC_WER
RTC_OER
RFU
MEDIA
L_IRQ1
CONTG_IO
HDD_LIGHT*
SEL_CAS/PD*
L2_CACHE_INH*
L2_CACHE_DIS*
MASK_TEA*
DRAM_TYPE
IRQ11
IRQ9
AUD_CS*
PD4_RD*
PD5_RD*
L2_FLUSH*
KYBD_CS*
RTC_ALE
RTC_RD*
RTC_WR*
RTC_AS0*
RTC_AS1*
RTC_WER*
RTC_OER*
DENSITY_RD*
IN
OUT
BI
GND
1
35
F245
TS
A7 A6 A5 A4 A3 A2 A1 A0
B7 B6 B5 B4 B3 B2 B1 B0
DIR E
MCE<7..0>
(CAS)
UBUSTR
SEL_CAS/PD
4D8< 9C2<>
19B1>
20C2>
19C1>
15C2>
18B2>
18A2<>
18C2>
15B2>
8C2>
19B7<
7C8<
9C8< 20A8<
5C1> 19A7<>
15B2<
8C7<
19C7<
5C4<
5B4<
* TRADEMARK OF ACTEL CORP
IBM
DRAWING
I/O INTERFACE EPLD
06/01/94
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 20 OF 22
8 7 6 5 4 3 2 1
D C B A


PCI_IDSEL<4> OUT
PCI_IDSEL<3> OUT
PCI_IDSEL<2> OUT
PCI_IDSEL<1> OUT 17B8<
+5V
32
POPULATE R334 FOR EPROM
POPULATE R345 FLASH
Ø
DO_NOT_POP
8B2> IN ROM_WE*
R345
R334
VCC
29F040
FLASH ROM
A18 A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
DQ7 DQ6 DQ5 DQ4 DQ3 DQ2 DQ1 DQ0
CE OE WE
VSS
16
GND
F244
A3 A2 A1 A0
Y3 Y2 Y1 Y0
OE
19
GND
8B2> IN ROM_CS*
8B2> IN ROM_OE*
FLASH_WE_A18
(PCI A/D)
17C2<> 7B8<> BI AD<31..Ø>
IBM
FLASH ROM + ID SEL
DRAWING
06/01/94
SIZE B
REVISION A
DRWNG PART NUMBER
SHEET 21 OF 22



+5V
(IN)
+3.6V
(OUT)
MJE2955
3
E
B
C
2
1
47
40.2
40.2
330
2222A
1
B
C
3
E
2
GND
0.015UF
249
2
1
COMP
COL
LT1431CS8
3
VDD
REF
8
121
4
RTOP
RMID
7
GND-S
GND-F
5
5
0.1UF
274
33UF
GND
IBM
3.6 REG
DRAWING
06/01/94
SIZE
B
REVISION
A
DRWNG PART NUMBER
SHEET 22 OF 22
8
7
6
5
4
3
2
1
D
C
B
A

Appendix C
653 Buffer Details of Operation

C.1 653 Buffer Highlights

- Companion chip to IBM27-82654 PCI Bridge Controller
- Address and data buffer/multiplexer for CPU, memory, and PCI buses
  - 64-bit CPU data bus
  - 32-bit CPU address bus
  - 64-bit memory data bus
  - 12-bit memory address bus
  - 32-bit PCI multiplexed address and data bus
- Generates PCI parity for the PCI_AD lines
- Presents one load to the PCI_AD lines
- 0Hz to 33MHz PCI bus frequency
- Low power
  - Static operation (no clock for most circuits)
  - Less than 400mW active power
- 3.3V or 3.6V power supply (timings differ)
- 5V TTL-compatible I/O
- PCI drivers are compliant with PCI Specification, Revision 2.0
- 304-pin quad flat pack, 0.8 micron IBM CMOS4LP technology

C.2 653 Buffer Pin Descriptions

The # symbol at the end of a signal name indicates that the active or asserted state of the signal occurs with a low voltage level. When the # symbol is not present after the signal name, the signal is asserted with a high voltage level.

The terms *asserted* and *negated* are used extensively. The term *asserted* indicates that a signal is active, regardless of whether that level is represented by a high or low voltage. The term *negated* means that a signal is inactive. The term *deasserted* is also used to indicate a signal that is negated.

The following terms are used to describe the signal type:

in Input is a standard input-only signal.
out Output is a standard active driver
I/O Bi-directional

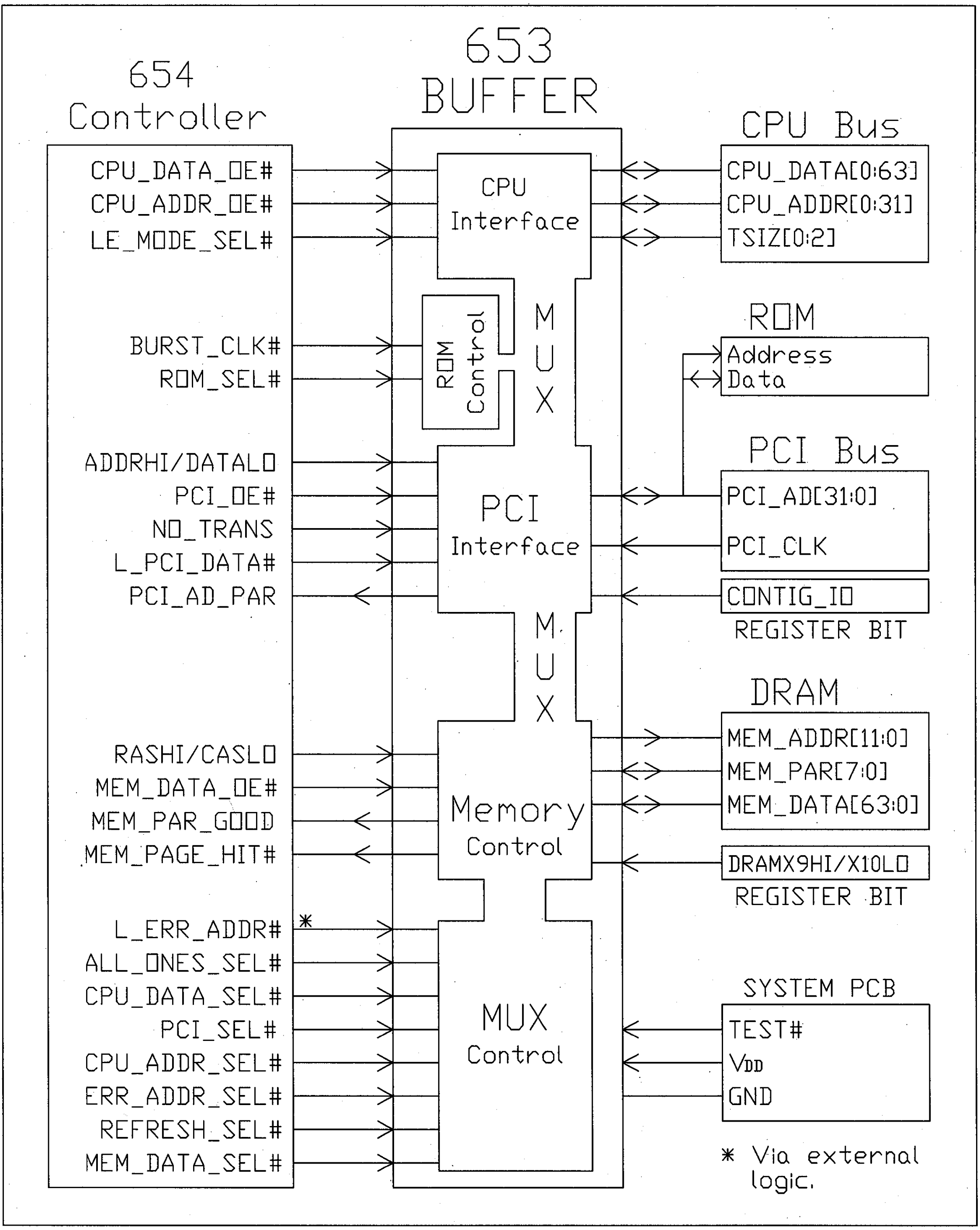


Figure C–1. 653 Buffer Pin Attachments

C.2.1 60X CPU Bus Interface Signals

Table C–1 describes the signals that interface the 653 Buffer to the 60X CPU bus.

Table C–1. 653 Buffer Signals—60X CPU Bus Interface

Signal Name	Type	Description
CPU_ADDR[0:31]	I/O	The 60X CPU address bus. The pin names of the 653 Buffer match the big-endian 60X CPU bus names, but these signals are renumbered in little-endian order as they pass through the 653 Buffer transceivers as cpu_addr_in[31:0] (inputs), and cpu_addr_out[31:0] (outputs). All internal 653 Buffer buses use little-endian nomenclature. See Section C.5.5.
CPU_DATA[0:63]	I/O	The 64-bit 60X CPU data bus. The 653 Buffer pins are numbered in big-endian 60X CPU bus order. CPU_DATA[0:31] connect to the 60X CPU signals DH[0:31]. CPU_DATA[32:63] connect to the 60X CPU signals DL[0:31]. The names of these signals are changed at the CPU data byte lane swappers (see Section C.5.21), but the significance of the bits within each byte is unchanged (for example D3h remains D3h). This bit renumbering matches the names of the 60X CPU data lines to the names of the PCI bus and memory data lines. If the system is operating in little-endian mode (LE_MODE_SEL# is asserted), the byte order is reversed by the swappers as data leaves or enters the 60X CPU from memory or the PCI bus. If the 60X CPU is in big-endian mode, there is no byte reordering.
TSIZ[0:2]	in	60X CPU bus transfer size—number of bytes. The 650 Bridge supports transfers of 1, 2, 3, 4, 8, and 32 bytes. When the system is operating in big-endian mode (LE_MODE_SEL# is asserted), the 653 Buffer unmunges addresses generated by the CPU based on the transfer size. See Section C.5.6.

C.2.2 System Memory Interface Signals

Table C–2 describes the signals that interface the 653 Buffer to system memory.

Table C–2. 653 Buffer Signals—System Memory Interface

Signal Name	Type	Description
MEM_ADDR[11:0]	out	Memory address bus, 12 bits, multiplexed, little-endian. While RASHI/CASLO is high, the MEM_ADDR lines contain row addresses selected from the internal data bus. While RASHI/CASLO is low, these lines contain the column addresses selected from the internal data bus.
MEM_ADDR0_B	out	A duplicate of MEM_ADDR[0]. (Required by some SIMMs.)
MEM_DATA[63:0]	I/O	64-bit memory data bus, with bit 63 = most significant bit. These signals are numbered in little-endian order.
MEM_PAR[7:0]	I/O	8-bit memory parity bus, bit 7 = most significant bit. Bit 7 corresponds to MEM_DATA[63:56]. Even parity is generated and written on memory write cycles. Parity is checked on memory read cycles.

C.2.3 PCI Bus Interface Signals

Table C–3 describes the signals that interface the 653 Buffer to the PCI bus.

Table C–3. 653 Buffer Signals—PCI Bus Interface

Signal Name	Type	Description
PCI_AD[31:0]	I/O	PCI address and data bus, 32 bits, multiplexed address and data. The PCI_AD bus is numbered in little-endian order.
PCI_CLK	in	PCI clock. PCI_CLK is used in the 653 Buffer to time the PCI data hold and address hold (demultiplexer) latches and the PCI Address/Data output multiplexer.

C.2.4 654 Controller Interface Signals

Table C–4 describes the signals that interface the 653 Buffer to the 654 Controller.

Table C–4. 653 Buffer Signals—654 Controller Interface

Signal Name	Type	Description
ADDRHI/DATALO	in	PCI address and data bus phase indicator, driven high by the 654 Controller to prepare the 653 Buffer for a PCI address phase, and driven low to prepare the 653 Buffer for a PCI data phase. (The PCI_AD bus is a multiplexed address/data bus).
ALL_ONES_SEL#	in	All ones select, asserted by the 654 Controller to the 653 Buffer to place all one-bits on the 653 Buffer internal data bus. ALL_ONES_SEL# is used during PCI configuration read transactions to return 64 one-bits to the CPU data bus when no PCI device responds. See Section C.5.22.
CPU_ADDR_OE#	in	CPU address output enable. While asserted, the 653 Buffer drives the internal address bus to the 60X CPU address bus. This allows snooping during PCI accesses to system memory.
CPU_ADDR_SEL#	in	CPU address select. While CPU_ADDR_SEL# is asserted or no other address select input is active, the 653 Buffer uses the 60X CPU address bus as the source of address information for a transaction. The CPU address hold latch is held transparent, the CPU burst counter (see Section C.5.8) is enabled to count, and the address MUX places the address from the CPU (via blocks 6, 7, and 8) on the internal data bus. After power up, this signal must be asserted and deasserted by the 654 Controller before any bus cycles are initiated to initialize the CPU burst counter.
CPU_DATA_OE#	in	CPU data output enable. While CPU_DATA_OE# is asserted, the 653 Buffer drives the contents of the 653 Buffer internal data bus onto the 60X CPU data bus. CPU_DATA_OE# is asserted by the 654 Controller during CPU to system memory reads and CPU to PCI reads.
CPU_DATA_SEL#	in	CPU data select. While CPU_DATA_SEL# is asserted or no other data select input is active, the 653 Buffer uses the 60X CPU data bus as the source of the data for a transaction. The data MUX (see Section C.5.22) places the data (byte-swapped in little-endian mode) from the CPU data bus onto the internal data bus for transmission to the PCI bus or system memory.
ERR_ADDR_SEL#	in	Error address select. While ERR_ADDR_SEL# is asserted, the contents of the error address latch (Section C.5.14) are placed on the internal data bus by the data MUX (Section C.5.22), for the CPU data bus. The 32-bit address is driven onto both halves of the 64-bit 60X CPU data bus.

Table C–4. 653 Buffer Signals—654 Controller Interface (Continued)

Signal Name	Type	Description
L_PCI_DATA#	in	Latch PCI data. While L_PCI_DATA# is not asserted and the PCI_CLK is low, the PCI data latch is transparent to the PCI_AD bus. When data is required from the PCI bus (during a CPU to PCI read or a PCI bus master to system memory write), the 654 Controller asserts this signal following the rising edge of the PCI_CLK for the current data phase. This latches the current data phase data into the PCI data latch. The 4-byte data is then duplicated as an 8-byte quantity, and placed on both halves of the 8-byte internal data bus. See Section C.5.14.
LE_MODE_SEL#	in	Little-endian mode select. LE_MODE_SEL# is asserted by the 654 Controller to set the 653 Buffer to little-endian mode. While LE_MODE_SEL# is asserted, CPU data byte lanes are swapped (see Section C.5.21) and the CPU addresses are unmunged (see Section C.5.6). While LE_MODE_SEL# is negated, the CPU sourced data is not swapped and CPU-sourced addresses are not unmunged. This signal can only be changed between bus cycles.
MEM_DATA_OE#	in	Memory data output enable. While MEM_DATA_OE# is asserted, the 653 Buffer drives the 64-bit internal data bus and its eight parity signals onto the memory data bus and the memory parity bus. MEM_DATA_OE# is asserted by the 654 Controller during memory write cycles.
MEM_DATA_SEL#	in	Memory data select. MEM_DATA_SEL# is asserted by the 654 Controller during a memory read transaction. When MEM_DATA_SEL# is asserted, the 653 Buffer uses the memory data bus as the source for the current transaction. The data MUX (Section C.5.22) places the memory data onto the internal data bus for the PCI bus or CPU data bus.
MEM_PAGE_HIT#	out	Memory page hit. The page hit comparator (see Section C.5.9) compares the address on the 60X CPU address bus to the address of the previous memory access (from any source) in the page hold latch (see Section C.5.16). The 654 Controller uses this signal to detect DRAM page hits.
MEM_PAR_GOOD	out	Memory parity good. This is an unqualified parity check output from the 653 Buffer. It is derived from the current contents of the memory data and memory parity buses. This signal becomes valid one delay time (t41) following the assertion of valid data and parity signals by the system DRAM. Additionally MEM_PAR_GOOD is forced high while the MEM_DATA_SEL# input is high
NO_TRANS	in	No translation. NO_TRANS forces no translation of the two most-significant bits of the address from the CPU or PCI buses. During most cycles mastered by the 60X CPU (see Section C.5.12) or a PCI bus master (see Section C.5.3) address bits [31:30] are translated to implement the system memory maps. To defeat this translation during PCI bus master cycles initiated by the I/O bus bridge for an ISA bus master, the 654 Controller asserts NO_TRANS to the 653 Buffer.
PCI_AD_PAR	out	PCI address/data parity, even parity across the PCI_AD[31:0] lines only. This is an unqualified signal that is only valid when the PCI_AD bus is valid. The 654 Controller combines PCI_AD_PAR with PCI_C/BE[3:0] to generate PCI_PAR, the PCI even parity bit.
PCI_OE#	in	PCI output enable. While PCI_OE# is asserted, the 653 Buffer drives the internal address or data buses onto the PCI_AD bus. PCI_OE# is asserted whenever the CPU has busmastership except during the data phase of reads from the PCI. Also see the ADDRHI/DATALO signal.

Table C–4. 653 Buffer Signals—654 Controller Interface (Continued)

Signal Name	Type	Description
PCI_SEL#	in	PCI select. While PCI_SEL# is asserted by the 654 Controller, the 653 Buffer treats the PCI bus as the source of addresses and data. While asserted it allows the following operations: During PCI bus master transactions, addresses of PCI bus master transactions to system memory are latched into the PCI address latch (see Section 9.4.1). The PCI burst counter (see Section C.5.4) is enabled to operate on these addresses during PCI bursts. The address MUX places these PCI sourced addresses onto the internal address bus. During PCI bus master writes to system memory and during 60X CPU reads from the PCI bus, the 654 Controller causes data sourced by the PCI_AD bus to be placed onto the 653 Buffer internal data bus by the data MUX (Section C.5.22) by asserting PCI_SEL#.
RASHI/CASLO	in	RAS# high, CAS# low. While RASHI/CASLO is driven high, the 653 Buffer asserts the row (RAS#) address onto the memory address lines, and the page hold latch is transparent. The falling edge of RASHI/CASLO latches the row address into the page hold latch. While RASHI/CASLO is low, the column (CAS#) address is driven onto the memory address lines.
REFRESH_SEL#	in	Refresh cycle select. Configures the 653 Buffer to accomplish a DRAM refresh cycle. While low, the 653 Buffer places the refresh counter address on the internal address bus. Then the row/column address MUX places this row address on the memory address bus (RASHI/CASLO must be high). Then the 654 Controller strobes the RAS# lines to refresh the DRAMs. The rising edge of REFRESH_SEL# increments the refresh counter.
BURST_CLK#	in	ROM and burst counter clock. While ROM_SEL# is active (during ROM accesses) the falling edge of BURST_CLK# increments the ROM read burst counter (see Section C.5.13) and shifts the data in the ROM read shift register (see Section C.5.19). While CPU_ADDR_SEL# is active (during 60X CPU mastered transactions) the CPU burst counter (see Section C.5.8) is clocked. While PCI_SEL# is active or if ADDRHI/DATALO is low (during PCI bus mastered transactions) the PCI burst counter (see Section C.5.4) is clocked.
ROM_SEL#	in	ROM select. The 654 Controller asserts ROM_SEL# during a ROM burst read cycle.

C.2.5 External Logic and System Interface Signals

Table C–5 describes the signals that are used to interface the 653 Buffer to the rest of the system via external logic, command bit storage elements, and the test interface.

Table C–5. 653 Buffer Signals—External Logic and System Interface

Signal Name	Type	Description
CONTIG_IO	in	Contiguous I/O. CONTIG_IO is asserted high by external logic to enable direct mapping of addresses from 2G to 2G + 8M. When CONTIG_IO is driven low, it enables non-contiguous addressing in the 2G to 2G + 8M address range. Non-contiguous I/O is a mapping of the low 32 bytes of each 4kB page of CPU memory space to 32 bytes of PCI/ISA IO space. See Section C.3.4.
DRAMX9HI/X10LO	in	DRAM type, asserted high for addressing DRAMs with 9 column address bits (x9 mode), low for x10 mode. This signal is used by the refresh counter (Section C.5.10) and the row/column address MUX (Section C.5.15) to format the addresses presented to the DRAMs.

Table C–5. 653 Buffer Signals—External Logic and System Interface (Continued)

Signal Name	Type	Description
L_ERR_ADDR#	in	Latch error address. The address on the 653 Buffer internal address bus is latched into the 653 error address latch on the falling edge of L_ERR_ADDR#, which can be derived by external logic from the 654 Controller signals TT_ERR#, MEM_PAR_ERR# and, optionally, any other signal indicating an error condition requiring the address to be latched. L_ERR_ADDR# must be held asserted to hold the contents of the latch. Any signal used with TT_ERR# and MEM_PAR_ERR# to derive L_ERR_ADDR# must also be held until after the latch is read. See Section C.5.14.
TEST#	in	IBM LSSD test mode input. Tie to V_{DD} with a 10k ohm resistor during normal operation.

C.3 The 653 Buffer

The IBM 27-82653 (653 Buffer) is one part of the IBM 27-82650 PowerPC™ 60X CPU to PCI Bridge Chip Set. The 653 Buffer interconnects the 60X CPU 32-bit address and 64-bit data buses with the PCI 32-bit multiplexed address-data bus. The 653 Buffer also generates the address and data buses to DRAM memory. This chip operates under the control of the IBM 27-82654 chip (654 Controller) which decodes all cycle types and asserts output signals to the 653 Buffer to select address and data paths.

Most timing in the 650 Bridge is controlled by the 654 Controller. Output timings of the 653 Buffer are usually combinatorial—they are measured from a data or address input or a path control signal, not a clock. Switching the PCI_AD line outputs from the address phase to the data phase is an exception—it is measured from the PCI clock. PCI addresses and data are latched with the PCI clock because the PCI standard specifies zero hold time on inputs.

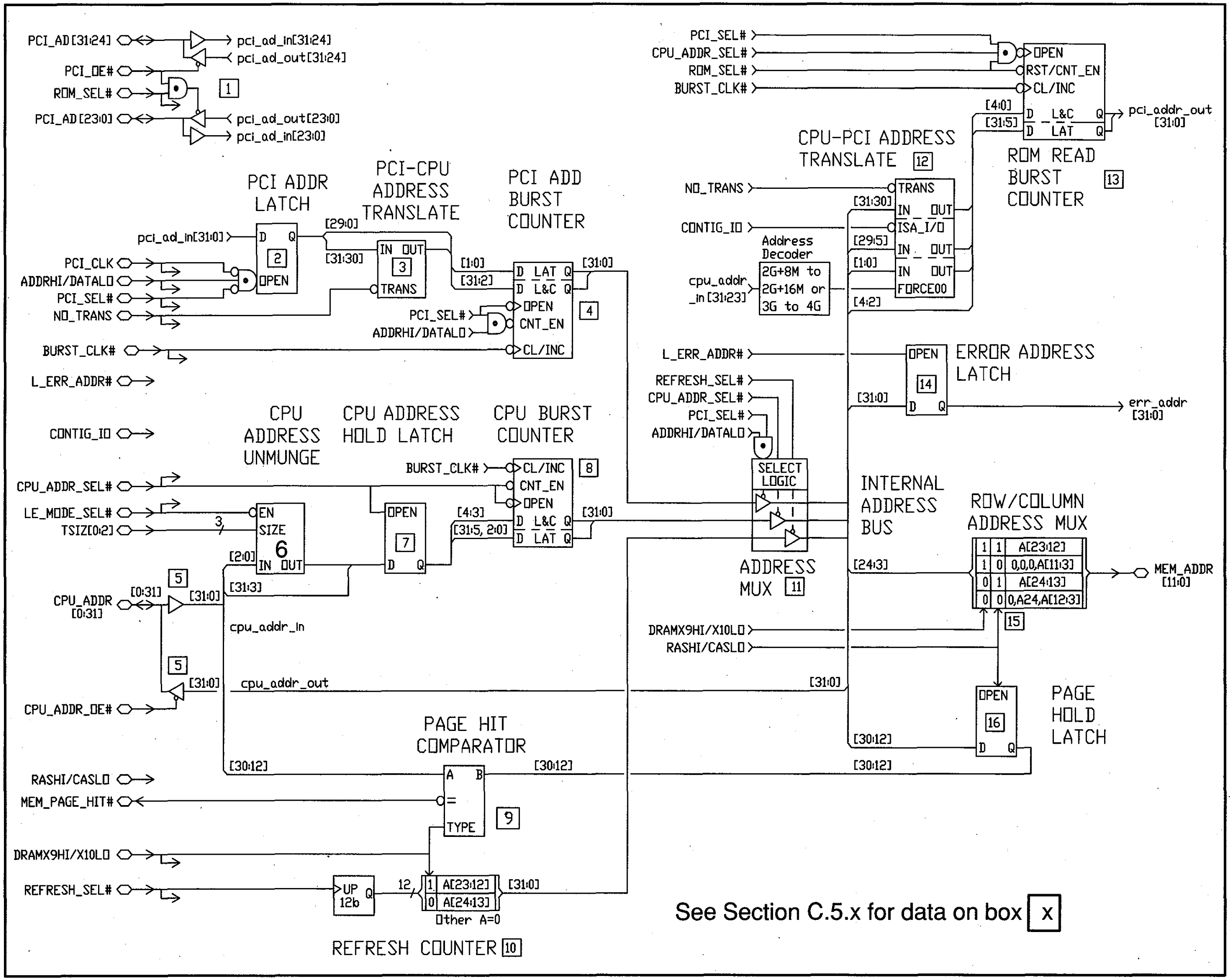
Although the 653 Buffer is used primarily in conjunction with the 654 Controller, it could be used with a different controller in order to design a bridge for a different 64-bit processor. Or it could be used to design a special-application bridge for a PowerPC processor.

This document describes the address and data paths and the control signals in two levels of detail. In the first level, enough detail is presented to enable a designer to utilize the chip with the 654 Controller. The second level of detail is for designers who need a much deeper understanding of the paths and control signals in order to design a controller or to make a special adaptation.

C.3.1 Architectural Overview Showing Address and Data Flow

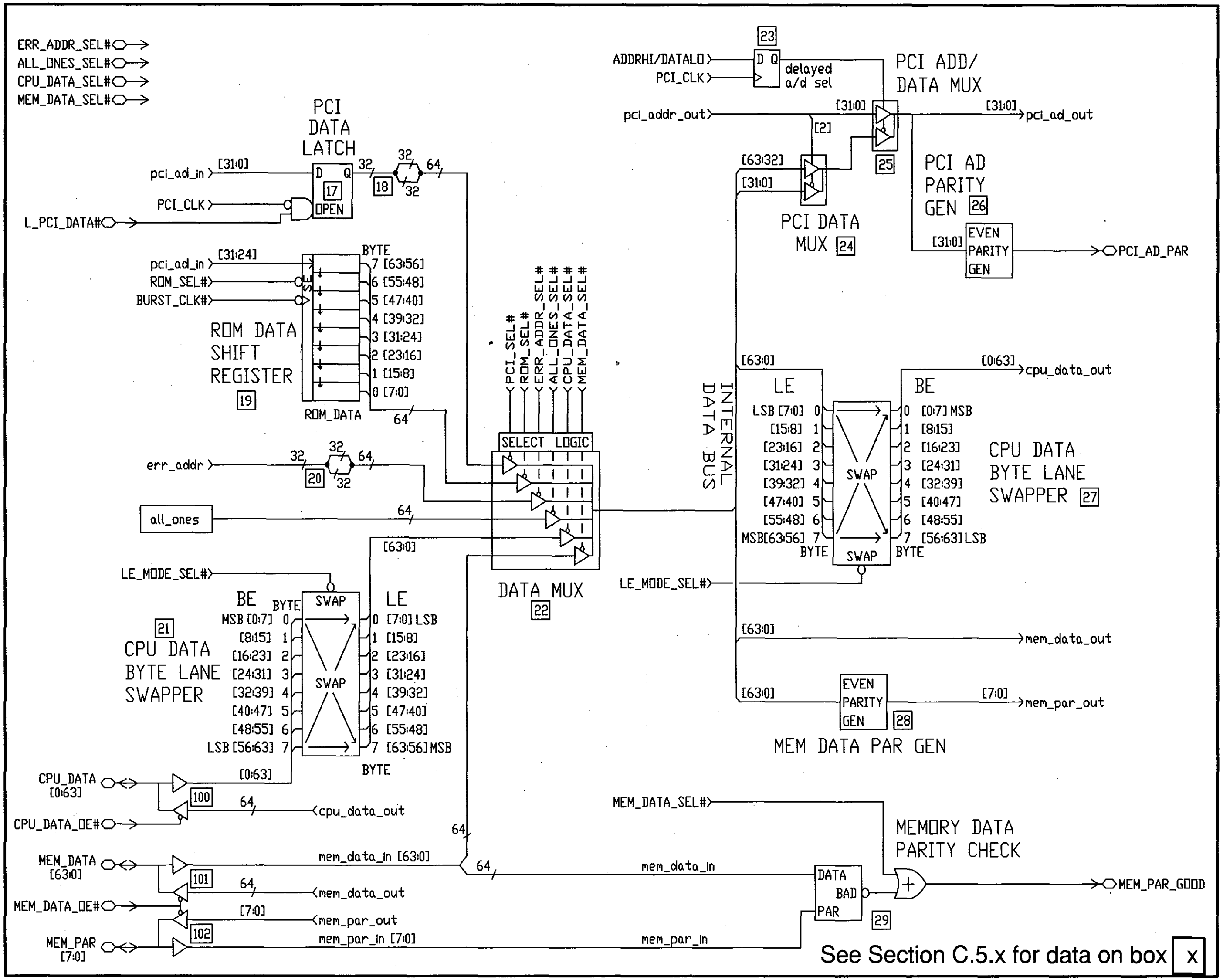
This section gives an overview of the architecture of the 653 Buffer. Figure C–2 is a block diagram of the *address* flow within the 653 Buffer. Figure C–3 is a block diagram of the *data* flow within the 653 Buffer. These diagrams explain function and are not intended to show the actual internal chip construction. For example, there are no three-state devices in the 653 Buffer except at the off-chip drivers.

The discussions in this section refer to the block diagrams. Address and data flows are illustrated in reference to functional cycles. Note that the address and data portions interconnect so that the diagrams together describe the complete 653 Buffer chip. Within this section, references to portions of the block diagrams use a block number within brackets, like [x], to refer to the portion of the diagram with the corresponding number.



See Section C.5.x for data on box $\boxed{x}$

Figure C–2. 653 Buffer Address Flow Functional Diagram



See Section C.5.x for data on box x

Figure C–3. 653 Buffer Data Flow Functional Diagram

C.3.2 Two High-order PCI Address Bits—NO_TRANS Pin

The CPU-PCI ADDRESS TRANSLATE block [12] on Figure C–2 receives its inputs from the internal address bus. During 60X to PCI cycles, this address corresponds to the address emitted by the 60X processor. If NO_TRANS is low, the two high-order signals are forced to 00b. This function supports the memory mapping scheme of the PowerPC Hardware Reference Platform. (All types of PCI transactions have addresses in the range of 0 to 1G.

Note that this address translation only occurs when the 60X CPU accesses the PCI bus and it can be defeated by the NO_TRANS input pin.

C.3.3 Two Low-Order PCI Address Bits

In order to conform to the requirements of the PCI revision 2 specification, the low-order two address bits are set to 00b in certain circumstances by the CPU-PCI address translate block [12] on Figure C–2. This block [12] decodes the input address from the CPU (cpu_addr_in[31:23]) and modifies these two bits as shown in Table C–6.

The controller drives the 653 Buffer inputs so that the output of this block is driven to the PCI_AD lines only during the address phase of a PCI cycle.

Table C–6. Low Order PCI Address Bit Settings

Input 60X Address	Output at block [12] of bits[1:0]	Cycles Supported at the 653 Buffer
0 to 2G	Same as input	Broadcast of system memory address to AD lines
2G to 2G + 8M	Same as input	PCI I/O
2G + 8M to 2G + 16M	00b	PCI Configuration
2G + 16M to 3G	Same as input	PCI I/O
3G to 4G – 8M	00b	PCI Memory
4G – 8M to 4G	00b*	Transmission of ROM address using AD lines

* Set to 00b, this increments during ROM read operation.

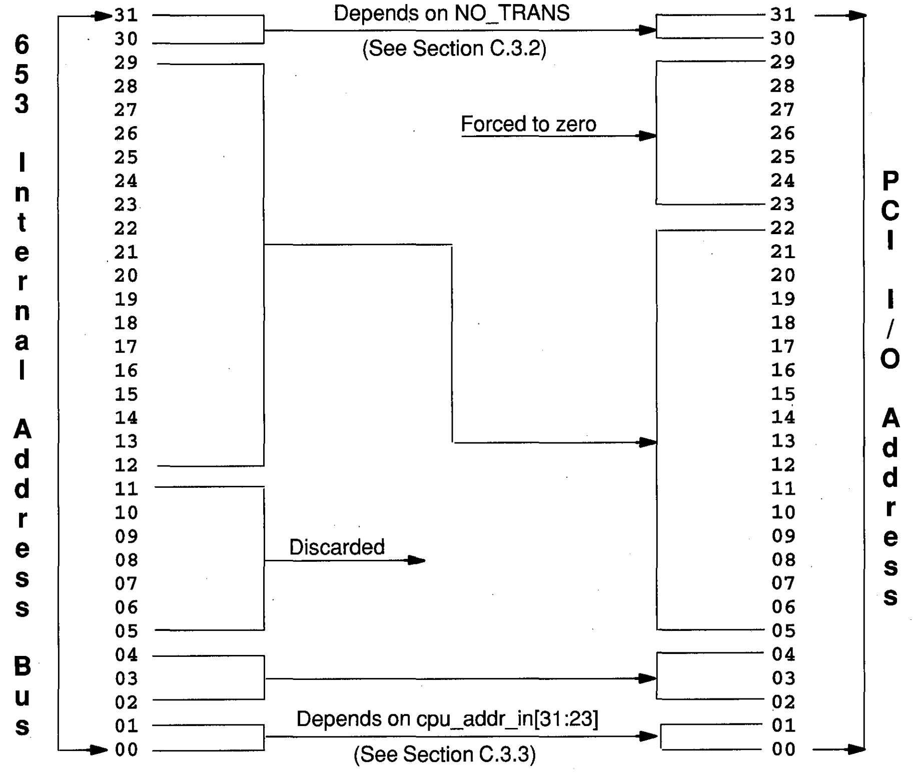
C.3.4 Contiguous I/O Pin

When the CONTIG_IO pin is high, 60X CPU addresses in the 2G to 2G + 8M address range are mapped directly as PCI I/O transactions from 0 to 8M. (See Table C–6.)

When the CONTIG_IO pin is low, non-contiguous I/O is activated. Non-contiguous I/O is an optional mode of operation where the memory-mapped address space corresponding to the 64K ISA address space can be remapped to the 8M region from 2G to 2G + 8M. Within this 8M region each 32 bytes of ISA address space is assigned to a different 4K page of CPU address space so that protection attributes can be assigned to the 32 ISA addresses. Figure C–4 shows the address transformation that occurs when the CONTIG_IO pin is low, activating non-contiguous I/O.

The CPU-PCI ADDRESS TRANSLATE block [12] decodes the address on the internal address bus (which corresponds to the CPU input address for 60X to PCI cycles). When the address is in the range from 2G to 2G+8M and CONTIG_IO is low, this block shift-translates its output as shown in Figure C–4.

If CONTIG_IO is high, the shift-translate illustrated above does not occur and all inputs (29:0) are passed to the outputs. This mode supports operating systems that do not require I/O port protection. The two high-order and two low-order outputs are controlled as explained in Section C.3.2 and Section C.3.3 respectively.



A31 to A30 are passed subject to NO_TRANS. A29 to A12 are shifted to A22-A5. A11 to A5 are discarded. (On the input side A2 to A0 are unmunged in LE mode.) A29 to A23 are set to zero, and A1 to A0 may be forced to zero.

Figure C–4. Non-Contiguous PCI I/O Address Transformation

C.3.5 60X to ROM Read Cycles

Figure C–5 shows how the 653 Buffer supports 8-bit ROM, EPROM, or flash devices connected to the AD bus.

The AD bus driver/ receivers [1] are split into two groups so that the address can be driven to the ROM device on PCI_AD[23:0] while data is received on PCI_AD[31:24]. Address flow is generally the same as described for CPU to memory address except that only PCI_AD[23:0] are driven from the internal address bus. Also, the low-order address bits latched into the ROM read burst counter [13], are initially set to 000b regardless of the state of the input address lines.

Receivers for AD[24:31] are active in this case. When the controller has allowed enough time for ROM read data to be valid, it must pulse BURST_CLK# low. This low-going edge clocks the data into ROM data shift register [19] on Figure C–2 and shifts all previous data bytes down one stage.

The same edge also increments the rom read burst counter [13] of Figure C–2 by a count of one in order to present a new address to the ROM.

After eight pulses on BURST_CLK#, 64 bits of data have been collected in [19]. These data bits are passed through multiplexer [22] to the internal data bus, through byte swap [27] and to the CPU data bus at [100]. The controller must not activate PCI_FRAME# or other PCI control lines during ROM read or write.

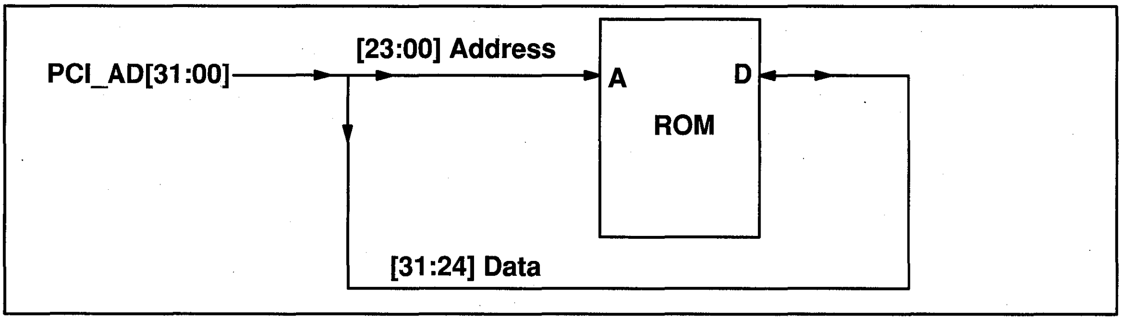


Figure C–5. 60X to ROM PCI_AD Flow

C.3.6 60X to ROM Write Cycles—Address and Data Flow

A flash memory or other writeable device must be connected as shown in the ROM read explanation. The data and the address must be encoded in the data field of the 4-byte store instruction which the controller decodes as a ROM write cycle. The address lines are immaterial to the 653 Buffer in this case. The 32 bits of data (either the high or low half of the 60X data bus is meaningful depending on the write address) propagates through byte swapper [21] to the internal address bus. The controller must activate CPU_DATA_SEL#, not ROM_SEL#, in order to propagate the data field to the internal data bus.

The meaningful 32 bits (selected by pci_addr_out[2]) propagate through [24] and [25] to the PCI_AD lines at [1]. All 32 bits must be enabled to drive the AD lines in this case.

C.3.7 Error Address Latch

This register, which is shown on Figure C–2, can be used to support the trapping of certain errors such as a memory parity error or an unsupported alignment. The register is normally open. When the controller senses an error it can change the state of L_ERR_ADDR# so that the address currently on the internal address bus is held. Later when the CPU runs a read cycle at some designated address, the controller can activate ERR_ADDR_SEL# at multiplexer [22] in order to provide the trapped address to the CPU.

C.3.8 Refresh Address Generation

The REFRESH counter is shown at block [10] of Figure C–2. The controller activates REFRESH_SEL# approximately every 15 usec when there is no other bus activity. The output of the refresh counter flows through multiplexer [11] to the internal address bus and to the ROW/COLUMN multiplexer [15]. When the controller changes REFRESH_SEL# to high, the 12-bit refresh counter increments. The counter wraps to zero following a maximum count.

C.3.9 All_Ones Generator

The 653 Buffer has an all_ones generator shown as an input to block [22] on Figure C–2. This device drives all of the internal data lines to a logical high voltage. It is useful in situations such as when the CPU tries to read a memory address which is out-of-range. Activating ALL_ONES_SEL# provides all one-bits as a response to a CPU or PCI read.

C.3.10 Page Hit Generation

The 653 Buffer contains logic, shown on blocks [9] and [16] of Figure C–2, to compare an incoming memory 4K (or 8K if DRAMX9HI/X10LO is low) page address with the last page address. Each time a new row address is output to the memory, it is latched into the PAGE HOLD latch [16] when the controller drives RASHI/CASLO to the low state. Bits (30:12) are saved. Hence, the 650 supports system memory up to 2G.

Whenever the CPU presents a new address at the address input pins, the comparator [9] indicates if the new address compares to the page address in the hold latch [16]. In the case of a new PCI memory address, the comparator works in the same way because the controller enables the incoming PCI address to be broadcast to the CPU address for snooping.

C.3.11 Special Considerations

Following power up, the 653 Buffer input CPU_ADDR_SEL# must be asserted and deasserted at least once to initialize the CPU burst counter to a known state. The 654 Controller performs this task.

All of the pins of the 653 Buffer will be tri-stated following the assertion of a TTL low state on the TEST#, DI1# (L_ERR_ADDR#), and DI2# (ERR_ADDR_SEL#) inputs. Refer to the IBM LSSD Test Procedure Specification (CMOS4LP book).

C.3.12 Warm Reset

The 653 Buffer provides an input and output pin to synchronize and hold a warm boot reset to the CPU. External logic asserts SRESET_REQ# to request a warm reset and the 654 Controller responds by asserting SRESET_CPU# to the 60X CPU.

C.4 Detailed Analysis of Address and Data Flow

C.4.1 60X to Memory Cycle Address Flow—Read or Write

60X addresses enter the chip at [5] on Figure C–2. The pins are named to correspond to 60X nomenclature, but internally the signals are named with little-endian notation. The three low-order signals are applied to the address translate [6] where they transformed in little-endian mode or unchanged in big-endian mode. This is explained in Section 5.3.3.

The address enters the CPU burst counter[8]. The purpose of this counter is to increment bits 4:3 (60X A[27:28]) during CPU burst cycles. These two bits can start at any value and only these two bits are incremented when the BURST_CLK# input falls. This implements CPU sequential burst-mode addressing with the starting address on any 8-byte boundary.

C.4.2 60X to Memory Cycle Data Flow—Write

60X data is presented simultaneously with the addresses and it flows to byte lane swapper [21] on Figure C–3. The naming convention on the pins is big-endian in order to correspond to the 60X convention, but signals in the interior of the chip are named with little-endian conventions. The byte lane swapper swaps lanes in little-endian mode and passes the data lines without swap in big-endian mode. In all cases the swapper maintains the significance of bits within a byte. The operation of the swapper is explained in Section C.5.21.

The 64 bits of write data flows to the internal data bus when the 654 Controller activates CPU_DATA_SEL# at multiplexer [22] and the data signals are applied to the memory drivers [101]. Note that eight bits of parity are generated at [28] and output along with the data.

C.4.3 60X to Memory Cycle Data Flow—Read

During a memory read cycle the memory data is input at receivers [101] and applied to data multiplexer [22]. The assertion of 654 Controller signal MEM_DATA_SEL# selects this memory data, and the multiplexer places it onto the internal data bus. The 64 bits of data are applied to byte lane swapper [27]. In little-endian mode the lanes are swapped as read data is passed through and in big-endian mode no swap is made. In all cases the significance of bits within each byte is maintained. The operation of the swapper is explained more fully in Section C.5.21.

The output of the swapper is connected to the CPU bus drivers at [100] on Figure C–3. The incoming data and parity are compared in MEMORY DATA PARITY CHECK [29] block to produce the unqualified output signal MEM_PAR_GOOD.

C.4.4 60X to PCI Cycle Address Flow—Read or Write

60X addresses enter the chip at [5]. The pins are named to correspond to 60X nomenclature, but internal signals are named with little-endian notation. The three low-order signals are applied to the address translate [6] where they are transformed in little-endian mode or unchanged in big-endian mode.

This transformation is explained in Section 5.3.3. Note that the same transform applies whether the address is for 60X to memory or 60X to PCI cycles.

The address enters the CPU burst counter [7]. BURST_CLK# is not activated by the 654 Controller since the 654 does not support 60X bursts to PCI so the output is the same as the CPU input address. multiplexer [11] is gated by CPU_ADDR_SEL# in this case so that the address flows to the internal address bus.

The CPU address on the internal address bus is applied to the ROM READ BURST counter [13]. In this case the counter is open because CPU_ADDR_SEL# is active and all 32 bits flow through. The outputs are applied to the PCI ADDRESS/DATA multiplexer [25] on Figure C–3. This multiplexer is controlled by flip flop [23] which is in a high state during the address phase of a 60X to PCI cycle.

The external controller places the ADDRHI/DATALO input in a high state prior to the beginning of a 601 to PCI cycle. The controller must change the ADDRHI/DATALO input to low prior to the rising edge of the PCI clock that terminates the address phase so that flip flop [25] can toggle multiplexer [25] to the data state.

C.4.5 60X to PCI Cycle Data Flow—Write

60X data is presented at the receivers[5] at the same time that addresses are presented. The data flows to byte lane swapper [21] on Figure C–3. The operation of the swapper is explained in Section C.5.21. Note that the same transformation is applied whether the 60X CPU accesses memory or the PCI bus.

The 64 bits of write data flow to the internal data bus when CPU_DATA_SEL# is asserted to multiplexer [22]. This data flows to the PCI data multiplexer [24], which passes either the high or low 32 bits of data to the PCI Address/Data MUX. If pci_address_out[2] is high, data bits [63:32] are passed; if pci_address_out[2] is low, data bits [31:0] are passed. Note that the value of this internal address bit is the result of the endian-mode low-order address transformation so it represents the actual target PCI address in either endian mode.

The output of multiplexer [24] connects to multiplexer [25]. During the data phase of 60X to PCI write transactions, multiplexer [25] is gated by flip flop [23] to pass the output of the PCI data multiplexer [25]. Flip flop [23] was explained in Section C.4.4. These outputs are connected to parity generator[26] and to the PCI driver/receivers at [1]. AD parity is passed to the companion chip so that it can generate PCI_PAR.

C.4.6 60X to PCI Cycle Data Flow—Read

During the data phase of a 60X to PCI read cycle, the read data is received at driver/receivers [1] on Figure C–2 and passed to the PCI data latch [17] on Figure C–3. Data is latched at the rising edge of the PCI clock and held when the controller activates L_PCI_DATA#. The 32 bits of latched data are replicated into 64 bits in order to drive the 64-bit data multiplexer [22]. The same data is present on the low-order 32 bits and the high-order 32 bits.

On this cycle the controller activates PCI_SEL# in order to gate the replicated PCI read data through data multiplexer [22] to the internal data bus. The internal data bus connects to byte lane swapper [27].

In little-endian mode the lanes are swapped as read data is passed through. In big-endian mode no swap is made. In all cases the significance of bits within each byte is maintained. The operation of the swapper is explained more fully in Section C.5.21.

C.4.7 PCI Bus Master Cycles Address Flow—Read or Write

This section describes the address flow when the controller grants the PCI bus to a master other than itself. For example when a SCSI agent becomes PCI bus master.

The PCI_AD lines [1] carry address information on the first clock(s) of a PCI cycle. This address is latched at the PCI ADDR latch [2] on Figure C–2 with the rising edge of the PCI clock. The controller must assert PCI_SEL# low and have ADDRHI/DATALO in a high state. It then must change ADDRHI/DATALO to a low state to hold the address in [1], and it must hold the address in [2] throughout the PCI to memory cycle.

The two high-order PCI address bits are modified by PCI-CPU address translate block [3] in order to reverse the translation that occurs when the CPU accesses the PCI bus. This translation, which can be omitted by asserting NO_TRANS, is shown in Table C–7. (MSB means most-significant bit.)

Table C–7. PCI to 60X CPU and System Memory Address Translation

PCI Address (Source)			60X CPU Address NO_TRANS = 0			60X CPU Address NO_TRANS = 1		
MSB A31	**A30**	**RANGE**	**MSB A0**	**A1**	**RANGE**	**MSB A0**	**A1**	**RANGE**
0	0	0 to 1G	1	1	3G to 4G	0	0	0G to 1B
0	1	1G to 2G	1	1	3G to 4G	0	1	1G to 2G
1	0	2G to 3G	0	0	0G to 1G	1	0	2G to 3G
1	1	3G to 4G	0	1	1G to 2G	1	1	3G to 4G

The PCI address is latched in PCI ADDRESS BURST counter [4]. This counter passes bits (1:0). The rest of the bits beginning with bit 2 are incremented when the controller changes BURST_CLK# to low. The controller drives BURST_CLK# low when the snoop or the memory no longer needs the address during PCI burst cycles. The PCI address is gated through the ADDRESS multiplexer [11] when PCI_SEL# is low. The PCI address on the internal address bus is presented on the memory address lines in the same way as was explained in Section C.4.1.

The PCI address can also be presented to the 60X bus for bus snooping by activating CPU_ADDR_OE# at block [5]. In the normal mode of operation, the 654 Controller does not activate a memory r/w cycle if the high-order PCI address bit is zero when it is on the PCI bus (set to one after the translate). Snooping devices normally ignore the cycle if the highest order CPU address line is one. PCI bus masters access system memory with PCI memory transactions addressed from 2G to 4G. These transactions are mapped to system memory and the 60X CPU bus as transfers in the 0 to 2G range.

C.4.8 PCI to Memory Cycles Data Flow—Write

This section describes the data flow on write cycles when the controller grants the PCI bus to a master other than itself. For example when a SCSI agent becomes PCI bus master.

The PCI data is latched at PCI DATA latch [18] on the rising edge of PCI_CLK. The 32 bits of data are replicated on both the high and low portions of the input to 64-bit multiplexer [22]. They are enabled onto the internal data bus by PCI_SEL#. The 64 bits, along with parity, are output to the memory when the controller activates MEM_DATA_OE# at [102] on Figure C–3.

There is no byte swap or low-order address translation in the path from PCI either to or from memory. So the data order on the PCI and the memory are the same.

C.4.9 PCI to Memory Cycles Data Flow—Read

This section describes the data flow on read cycles when the controller grants the PCI bus to a master other than itself. For example when a SCSI agent becomes PCI bus master.

When the controller recognizes a PCI to memory read, it must activate MEM_DATA_SEL# so that data memory data entering the receivers at [101] on Figure C–3 can be gated through multiplexer [22] to the internal data bus. From this point, 32 bits (high or low) are selected at multiplexer [24] and flow through multiplexer [25] to PCI_AD drivers [1] on Figure C–2.

C.5 653 Buffer Detailed Internal Descriptions

This section contains detailed explanations of the operation of each part of the 653 Buffer shown in Figure C–2 and Figure C–3, which show the function of the 653 Buffer but not its actual internal construction. The number of each subsection matches the block number of each part shown in the figures.

C.5.1 PCI_AD Transceivers

The PCI_AD output drivers are enabled in two different groups (see Figure C–6) to allow the PCI_AD lines to be used to access the boot ROM device during ROM cycles. The upper byte (AD[31:24]) is enabled by PCI_OE#, and the lower three bytes (AD[23:0]) are enabled whenever either PCI_OE# or ROM_SEL# goes active low.

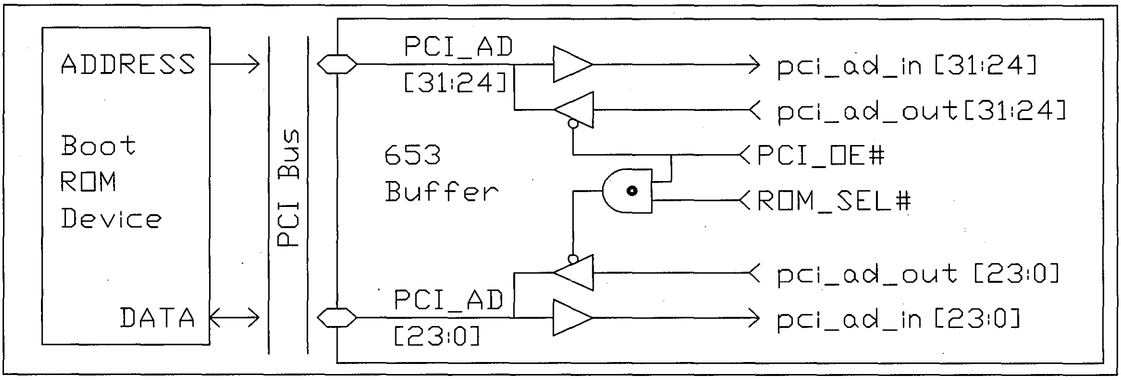


Figure C–6. PCI_AD Transceivers

C.5.2 PCI Address Latch

The purpose of the PCI address hold latch is to capture the address information from the PCI bus during the address phase of a PCI transaction. This function is accomplished using a hold latch.

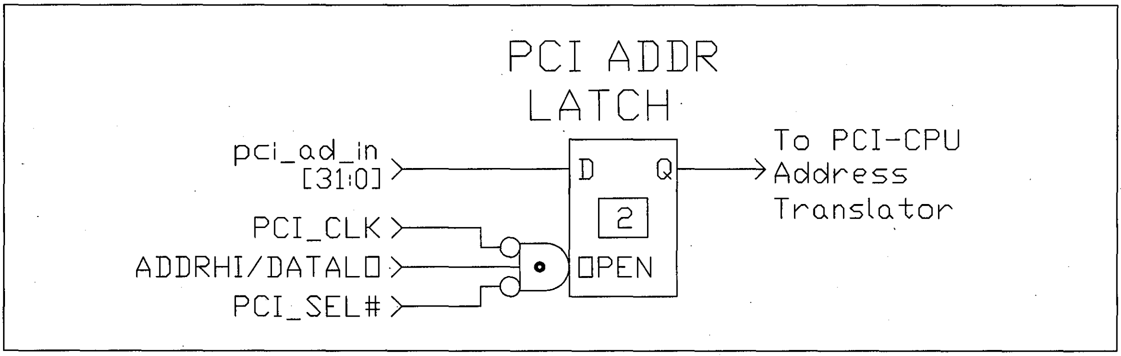


Figure C–7. PCI Address Latch

The hold latch (see Figure C–7) is a level sensitive, transparent D-latch. Addresses appearing on the D inputs are transferred to the Q outputs while the OPEN input is active high. Addresses appearing on the D inputs one setup time before the high to low transition of the OPEN input is held on the Q outputs until the OPEN input is again returned high.

The PCI address hold latch derives the OPEN signal from three other signals. The contents of the PCI_AD lines flow through this latch while PCI_SEL# is active low and ADDRHI/DATALO is high and PCI_CLK is low. In normal operation the combination of PCI_SEL# low and ADDRHI/DATALO high indicates that a PCI address phase is in progress, and the PCI_CLK transition from low to high latches the data. ADDRHI/DATALO would then be negated before the PCI_CLK again went low.

C.5.3 PCI to 60X CPU Address Translation

The purpose of the PCI to 60X CPU address translation block [3] is to map PCI addresses from PCI bus masters onto system memory and 60X CPU address space. This translation affects the upper two address lines [31:30] when NO_TRANS is inactive. When NO_TRANS is active high (ISA master cycles), no translation takes place.

PCI address bits 29:0 bypass this translation block and are not affected by this translation.

Table C–8. PCI to 60X CPU and System Memory Address Translation

PCI Address (Source)			60X CPU Address (BE) NO_TRANS = 0			60X CPU Address (BE) NO_TRANS = 1		
A31	A30	RANGE	A0	A1	RANGE	A0	A1	RANGE
0	0	0 to 1G	1	1	3G to 4G	0	0	0G to 1B
0	1	1G to 2G	1	1	3G to 4G	0	1	1G to 2G
1	0	2G to 3G	0	0	0G to 1G	1	0	2G to 3G
1	1	3G to 4G	0	1	1G to 2G	1	1	3G to 4G

C.5.4 PCI Burst Counter

The PCI burst counter (see Figure C–8) supports PCI burst accesses to system memory by latching in the initial address and incrementing it for each succeeding data phase of the burst. This is implemented using a combination latch/counter (see Figure C–9). There are two paths through the latch/counter—the latch-only path and the latch and counter path.

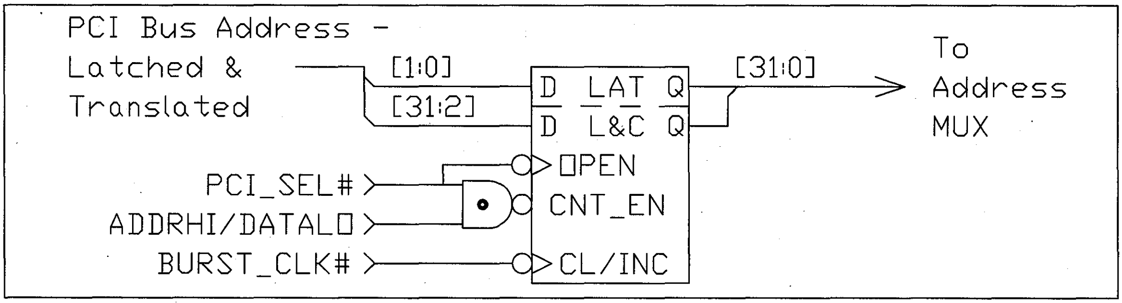


Figure C–8. PCI Burst Counter

Assume that the initial state of the latch/counter is CLOSED. A high to low transition of PCI_SEL# (on the *OPEN#* input) causes the latches to open (become transparent). This is the OPEN state (see Figure C–10). The *OPEN#* input is edge sensitive only.

While the *CNT_EN#* (CouNT_ENable) input is asserted, the *CL/INC#* (active falling edge CLose/INCrement) input is enabled, otherwise it is ignored. *CNT_EN#* is asserted whenever either PCI_SEL# or ADDRHI/DATALO is low.

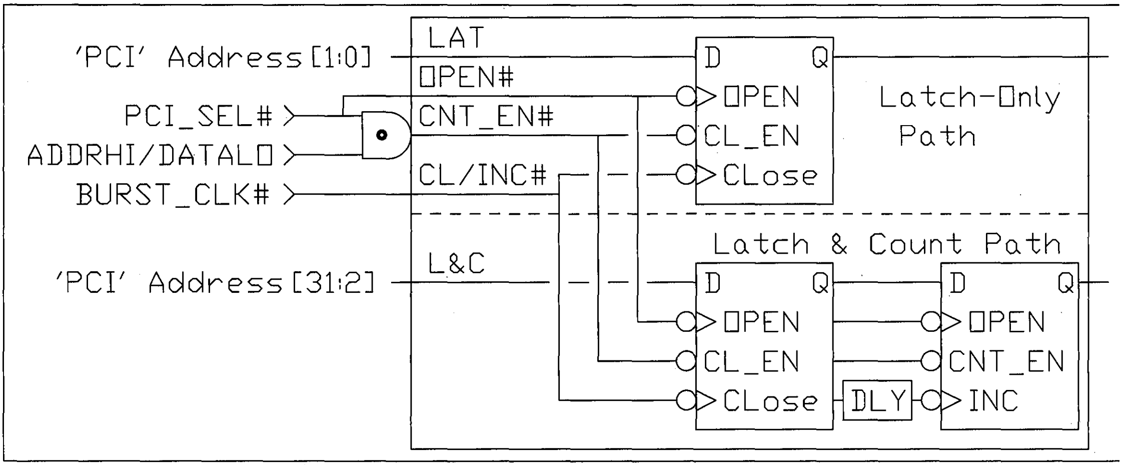


Figure C–9. Combination Latch/Counter—PCI Burst Counter

While the *CNT_EN#* input is asserted, sending BURST_CLK# (connected to *CL/INC#*) from high to low causes all the latches to close, and then causes the count on bits [31:2] to increment by one. Thus the address appearing on inputs [1:0] one setup time before the falling edge of BURST_CLK# appears on outputs [1:0]. One plus the address appearing on inputs [31:2] one setup time before the falling edge of BURST_CLK# appears on outputs [31:2].

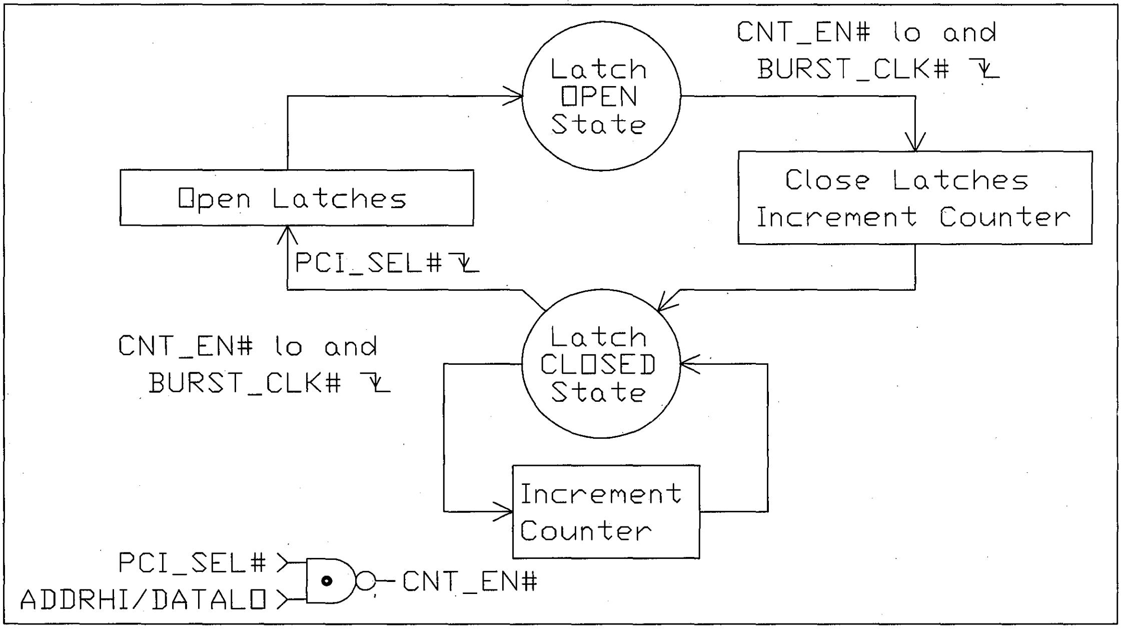


Figure C–10. Latch/Counter Flow Diagram—PCI Burst Counter

Successive high to low transitions of BURST_CLK# continue to increment bits [31:2] and have no effect on bits [1:0]. The *CL/INC#* input is edge sensitive only. Any high to low transition of the *OPEN#* input returns the device to the open (transparent) state.

C.5.5 60X CPU Address Bus Transceivers

The internal nomenclature of the 653 Buffer is little-endian, while the 60X CPU bus is labeled in big-endian sequence. Since the 653 Buffer internal structure uses little-endian nomenclature, the 60X CPU signals were renamed in little-endian sequence to minimize confusion. This allows all of the 653 Buffer to be discussed using the same nomenclature.

For example, in Figure C–11, A0 on the 60X CPU bus (and the 653 Buffer pin) corresponds to cpu_addr_in[31] and cpu_addr_out[31] in the 653 Buffer. A31 on the 60X CPU bus corresponds to cpu_addr_in[0] and cpu_addr_out[0] in the 653 Buffer.

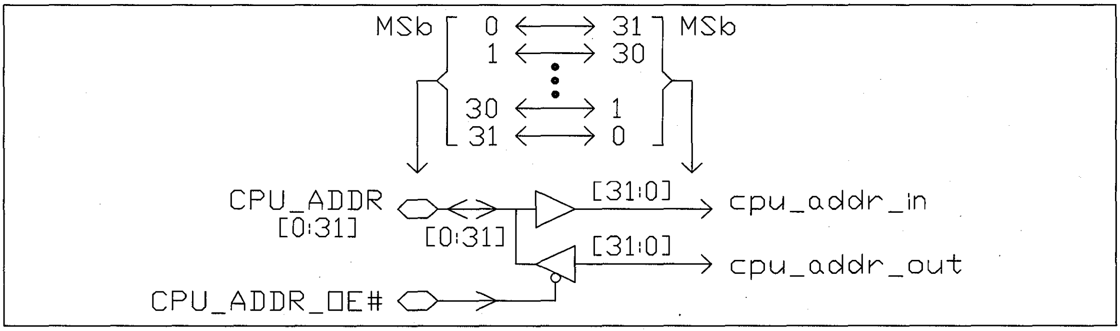


Figure C–11. 60X CPU Address Bus Transceivers

C.5.6 60X CPU Address UnMunger

The 60X CPU address unmunger is used to support big-endian and little-endian operation of the system. When LE_MODE_SEL# is asserted, the 653 Buffer unmunges the three least significant address bits from the 60X CPU bus to the memory bus or PCI bus (see Table C–9). The unmunge by the 653 Buffer is identical to the munge operation performed by the 60X CPU. Addresses which have been munged and then unmunged are identical to addresses that have not been manipulated.

Table C–9. Unmunging Address Bits in Little-Endian Mode

Lowest Order Address Bits Before Unmunge	Unmunged Three Lowest Order Address Bits [2:0]			
	1-Byte Transfer (XOR with 111)	2-Byte Transfer (XOR with 110)	4-Byte Transfer (XOR with 100)	8-Byte Transfer (No Change)
000	111	110	100	000
001	110	---	---	---
010	101	100	---	---
011	100	---	---	---
100	011	010	000	---
101	010	---	---	---
110	001	000	---	---
111	000	---	---	---

C.5.7 60X CPU Address Hold Latch

The 60X CPU address hold latch [7] is transparent while CPU_ADDR_SEL# is high. Addresses appearing on the D inputs one setup time before the high to low transition of CPU_ADDR_SEL# is held on the Q outputs until CPU_ADDR_SEL# is again returned high.

C.5.8 60X CPU Burst Counter

The 60X CPU burst counter (see Figure C–12) supports 60X CPU bus (60X or L2) single-beat transfers and four-beat burst transfers to system memory by latching in the initial address and (for bursts) incrementing it for each succeeding beat of the burst. This is discussed in terms of the 60X CPU, and works the same way for transfers mastered by the L2 cache. The burst counter is implemented using a combination latch/counter (see Figure C–13) which is very similar to the latch-counter described in Section C.5.4. There are two paths through the latch/counter: the latch-only path, and the latch and counter path.

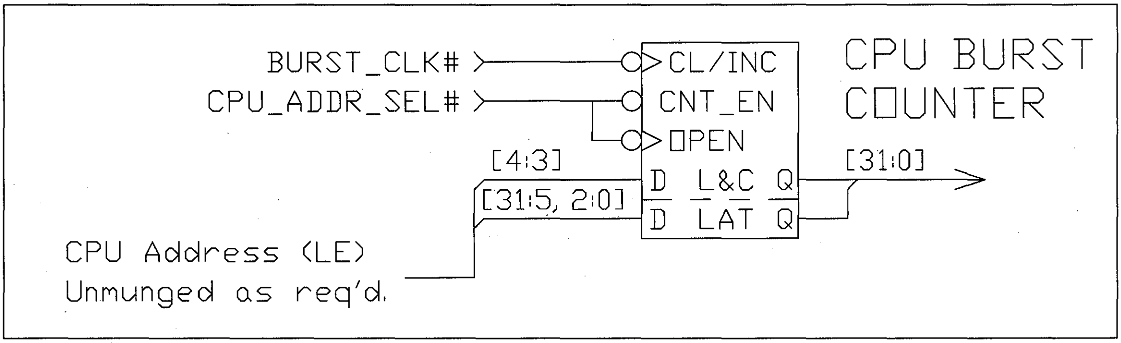


Figure C–12. 60X CPU Burst Counter

During a single-beat transfer, up to 8 bytes of data are transferred by the 60X CPU, as determined by a decode of the three lowest order address bits and TSIZ[0:2]. During each beat of a burst transfer (TBST# asserted) 8 bytes are transferred. Internal address bits [2:0] (LE) flow through the latch-only path of the burst counter. The value of these bits is controlled by the 60X CPU. Bits [31:5] also flow through the latch-only section. They are not incremented by the counter because any given 60X CPU burst transfer must not cross a 32-byte boundary.

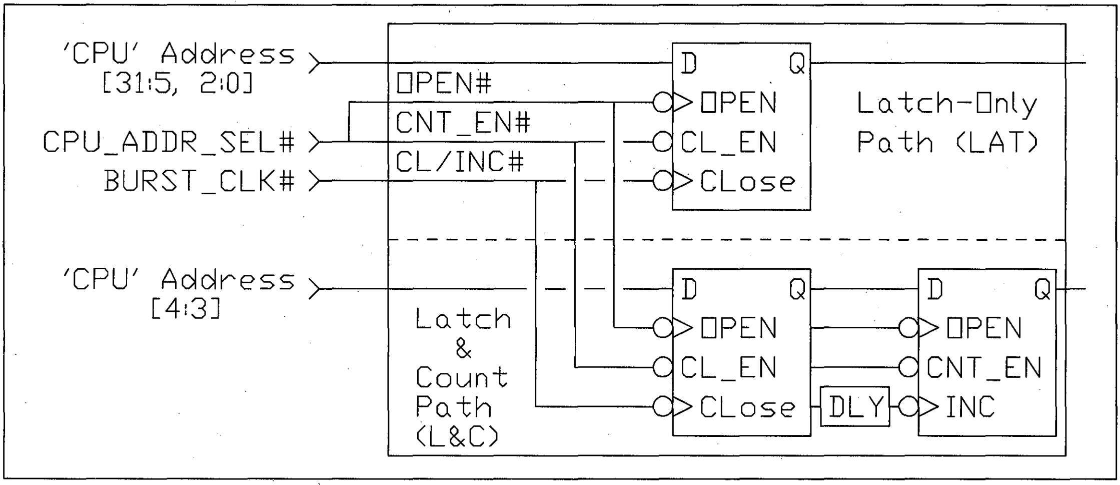


Figure C–13. Combination Latch/Counter—CPU Burst Counter

Since the 60X CPU burst is composed of four beats, a two-bit counter is required to support the burst. Bits [4:3] flow through the latch and counter path. The 60X CPU can initiate a burst at any value of bits [4:3]. The count sequence is 00, 01, 10, 11, 00, 01, and so on (the counter is linear and wraps).

To track the operation of the device, assume that the initial state of the latch/counter is CLOSED. The high to low transition of CPU_ADDR_SEL# (on the *OPEN#* input) causes the latches to open (become transparent). This is the OPEN state (see Figure C–14).

While the CPU_ADDR_SEL# is asserted (on *CNT_EN#*), the *CL/INC#* input is enabled, so changing BURST_CLK# from high to low causes all the latches to close, and then causes the count on bits [4:3] to increment by one. Thus the address appearing on inputs [2:0] and [31:5] one setup time before the falling edge of BURST_CLK# appears on outputs [2:0] and [31:5]. One plus the address appearing on inputs [4:3] one setup time before the falling edge of BURST_CLK# appears on outputs [4:3].

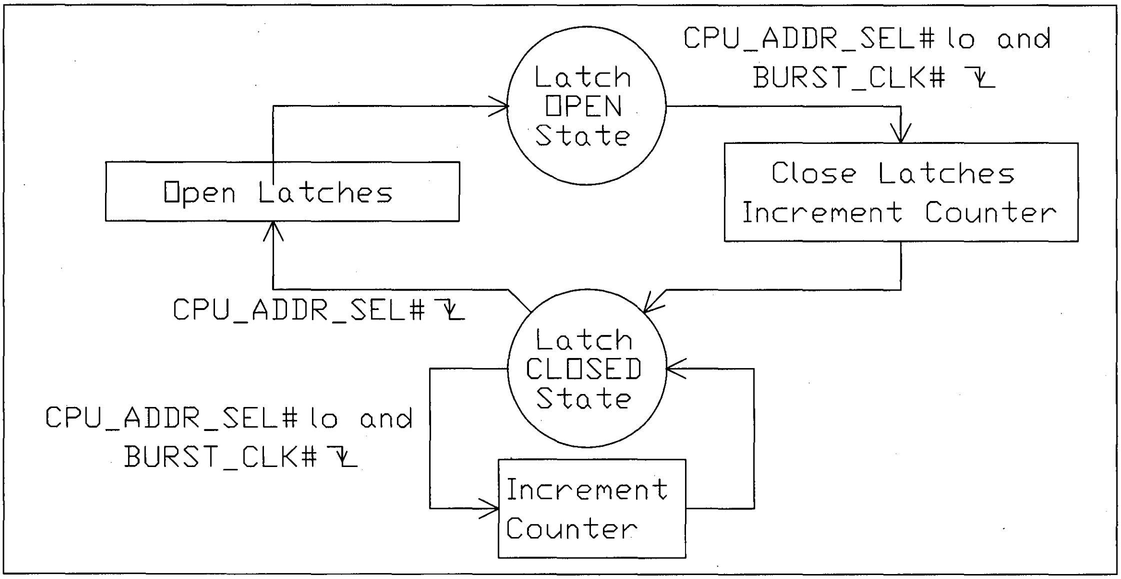


Figure C–14. Latch/Counter Flow Diagram—CPU Burst Counter

Successive high to low transitions of BURST_CLK# on CL/INC# continue to increment bits [4:3] with no effect on bits [2:0] and [31:5]. Any high to low transition of CPU_ADDR_SEL# (on the *OPEN#* input) returns the device to the open (transparent) state.

C.5.9 Page Hit Comparator

The 12-bit page hit comparator is used to support fast accesses to memory locations in the same page of DRAM as the previous DRAM access. Bits [30:12] of the previous page address, stored in the page hold latch, are compared to bits [30:12] of the current address coming from the cpu_addr_in bus. MEM_PAGE_HIT# is unqualified and is only guaranteed to be valid one delay time after the inputs to the comparator are valid.

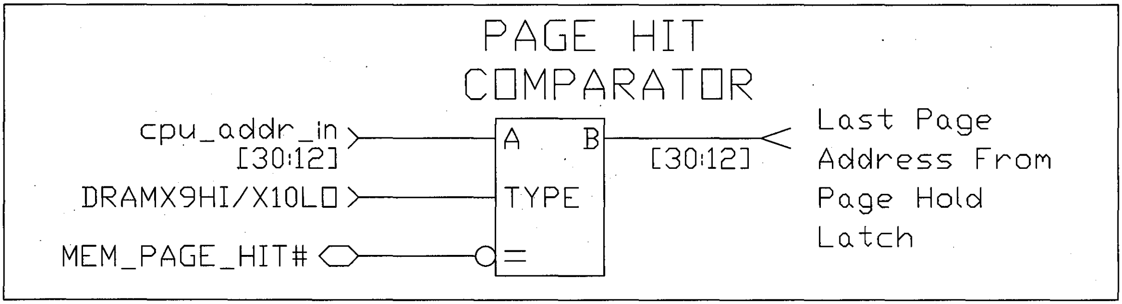


Figure C–15. Page Hit Comparator

The page hit comparator operates with two different sets of address lines, depending on the value of DRAMX9HI/X10LO. When DRAMX9HI/X10LO is high, the device expects a 4K page size. Addressing within a 4K page requires address lines [11:0]. Address lines [30:12] are then compared to determine page hits.

When DRAMX9HI/X10LO is low, the device expects an 8K page size. Addressing within an 8K page requires address lines [12:0]. Address lines [30:13] are then compared to determine page hits. Address line [31] is not used by the page hit comparator because all system memory must be mapped below 2G, so A31 is always low.

C.5.10 Refresh Counter

The refresh counter is used to determine the row address for refresh operations. The refresh counter is composed of a 12-bit counter and some steering logic (see Figure C–16).

The value of the 12-bit counter on power up is indeterminate. The counter increments on the rising edge of the REFRESH_SEL# input. The count sequence (decimal) is 0, 1, ... , 4095, 0, 1, ... etc., and is not affected by any other input.

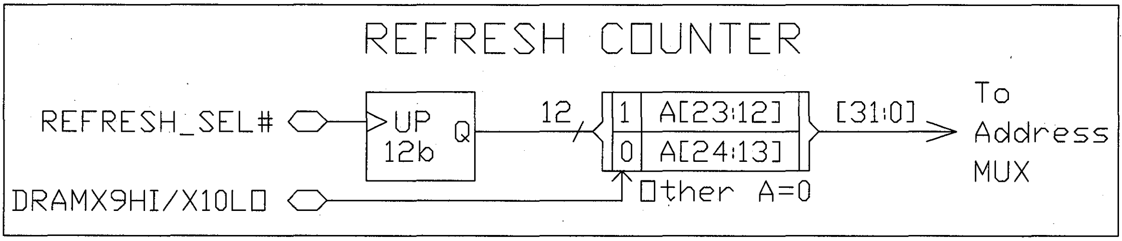


Figure C–16. Refresh Counter

When DRAMX9HI/X10LO is high, the memory controller is in x9 mode. When DRAMX9HI/X10LO is low it is in x10 mode. The 12-bit refresh address produced by the 12-bit counter is placed on the internal address bus with zero-fill, depending on the value of DRAMX9HI/X10LO, as follows:

```
 A    A    A    A    A    A    A    A  A
31   27   23   19   15   11    7    3  0---- Address Line
 |    |    |    |    |    |    |    |  |
                         abcd_efgh_ijkl---- 12-bit refresh address

 0000 0000 abcd efgh ijkl 0000 0000 0000---- Refresh address placed on
                                             internal bus, x9 Mode

 0000 000a bcde fghi jkl0 0000 0000 0000---- Refresh address placed on
                                             internal bus, x10 Mode
```

C.5.11 Address Multiplexer

The address multiplexer places a 32-bit address on the 653 Buffer internal address bus. This address comes from one of three sources—the PCI Burst Counter, the 60X CPU Burst Counter, or the Refresh Counter (see Figure C–17).

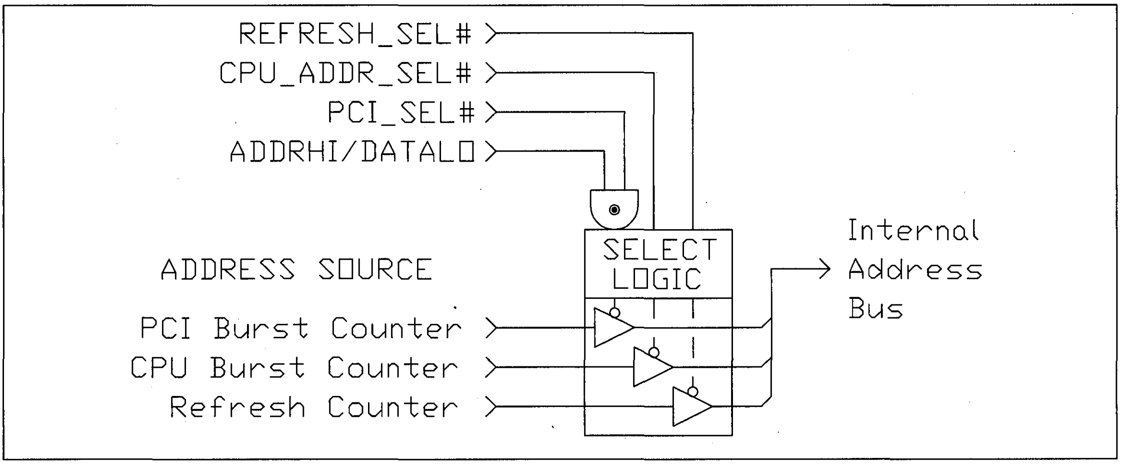


Figure C–17. Address Multiplexer

Only one of the address sources is selected at any time. In general, the refresh address select has the highest priority, the 60X CPU address select has second priority, and the PCI address select has third priority (see Table C–10). In the table (which exactly describes the operation of the multiplexer), note that the PCI address select signal (*from_pci#*), is an internal signal that is asserted low whenever PCI_SEL# is low or ADDRHI/DATALO is low.

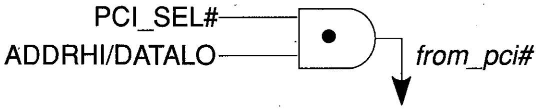


Table C–10. Address Multiplexer Source Selection Priority

CPU_ADDR_SEL#	*from_pci#*	REFRESH_SEL#	Selected Address Source
0	0	0	Refresh Counter
0	0	1	CPU Burst Counter
0	1	0	Refresh Counter
0	1	1	CPU Burst Counter
1	0	0	Refresh Counter
1	0	1	PCI Burst Counter
1	1	0	Refresh Counter
1	1	1	CPU Burst Counter

C.5.12 60X CPU to PCI Address Translation

Addresses sourced by the 60X CPU or the L2 cache are transmitted to the PCI bus after going through a translation block (Note that the addresses pass through the ROM read burst counter unchanged during PCI transactions). The address information comes to the translation block via the internal address bus A[31:0] (see Figure C–18). The address lines are processed in four groups—bits A[1:0], bits A[4:2], bits A[29:5], bits A[31:30]. Operations on each group are independent of operations on the other groups.

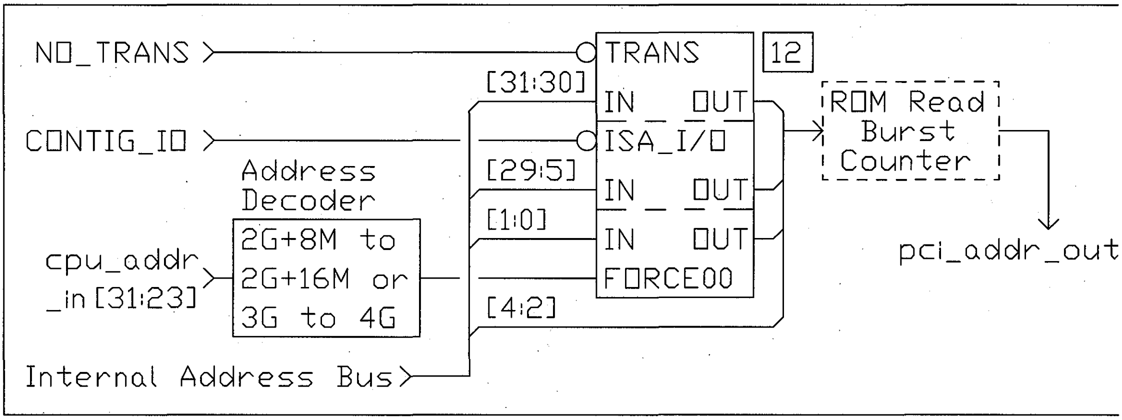


Figure C–18. 60X CPU To PCI Address Translator

C.5.12.1 A[1:0] Translation—PCI Bus Special Requirements

Address bits A[1:0] are forced to 00b to meet the special requirements of the PCI bus during CPU to PCI configuration transactions (CPU address range 2G+8M to 2G+16M), and during CPU to PCI memory transactions (CPU address range 3G to 4G). The 653 Buffer detects accesses to these address ranges by internally decoding the 8 highest order CPU address lines (from cpu_addr_in[31:23] (LE)). No pin or register control of this translation is provided. It is hardwired.

Note that system ROM space is mapped to CPU address range 4G–8M to 4G. There is no problem with A[1:0] being forced to '00' during ROM reads because the ROM read burst counter forces A[2:0] to '000' at the start of a ROM read; thereafter the state of A[1:0] is determined by the burst counter. During ROM writes, the address appearing on the ROM address lines comes from the CPU data bus via the 653 Buffer.

C.5.12.2 A[4:2] Non-Translation

Address bits A[4:2] pass through the translator with no change under any conditions. Note that A[2:0] will have been unmunged upstream at the 60X CPU Address Unmunger if the system is operating in little-endian mode.

C.5.12.3 A[31:30] Translation—System Address Map Implementation

The 653 Buffer allows pin control of the high order address mapping function from the 60X CPU bus to the PCI bus. While NO_TRANS is high, the address mapping function is disabled, and A[31:30] are passed through unchanged (see Table C–11).

While NO_TRANS is low, this mapping function is enabled, and address bits A[31:30] are translated as shown in Table C–11.

Table C–11. 60X CPU To PCI Address Translation—High Order

NO_TRANS	A[31:30] from CPU (Little-Endian)	CPU Address	PCI_AD[31:30]	PCI Address
0	0 0	0G to 1G	0 0	0G to 1G
0	0 1	1G to 2G	0 1	1G to 2G
0	1 0	2G to 3G	0 0	0G to 1G

Table C–11. 60X CPU To PCI Address Translation—High Order (Continued)

NO_TRANS	A[31:30] from CPU (Little-Endian)	CPU Address	PCI_AD[31:30]	PCI Address
0	1 1	3G to 4G	0 0	0G to 1G
1	0 0	0G to 1G	0 0	0G to 1G
1	0 1	1G to 2G	0 1	1G to 2G
1	1 0	2G to 3G	1 0	2G to 3G
1	1 1	3G to 4G	1 1	3G to 4G

C.5.12.4 A[29:5] Translation—PCI/ISA I/O Page Mapping

The 653 Buffer also allows pin control of the PCI/ISA IO mapping function, which concerns bits A[29:5] (see Figure C–19). Address bits A[29:5] are passed through the translator unchanged while CONTIG_IO is high, which maps the 60X CPU addresses into PCI space at 1:1 (for these bits). Operation in *ISA contiguous mode* is straightforward, the address space is contiguous. However, this mode allows protection attributes to be assigned to ISA ports only as is allowed by the 1:1 mapping to memory space—each 4k-byte *page* of ports has definable attributes that apply to all of the ports in that *page*.

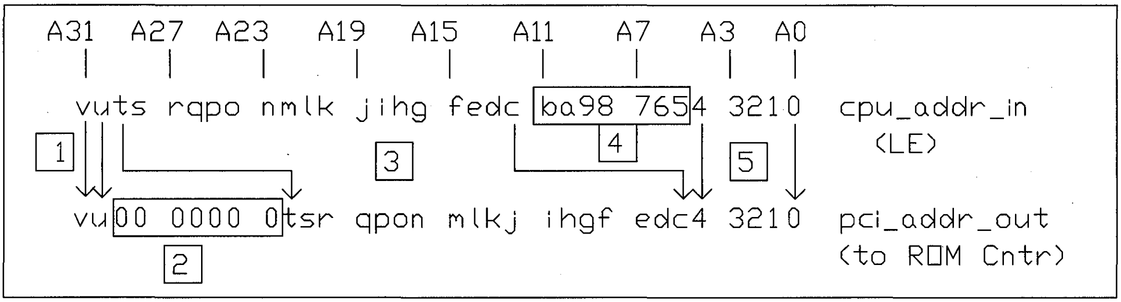

A31 A27 A23 A19 A15 A11 A7 A3 A0
vuts rqpo nmlk jihg fedc ba98 7654 3210
cpu_addr_in
(LE)
1
2
3
4
5
vu00 0000 0tsr qpon mlkj ihgf edc4 3210
pci_addr_out
(to ROM Cntr)

Figure C–19. 60X CPU To PCI Address Translation—PCI/ISA IO

While CONTIG_IO is low, bits A[29:5] are translated as shown in Figure C–19. This translation implements the mapping of 4k-byte pages in 60X CPU memory space onto 32-byte port groups in PCI/ISA space. This mapping allows protection attributes to be assigned to each group of 32 ports as a separate page. In this 'ISA non-contiguous mode', the lowest 32 bytes in each 4k-byte page of CPU memory space is mapped to a 32 byte group of ports in ISA I/O space. (The other 32 byte groups in each 4k-byte page are shadowed to the same 32-byte port group.) While CONTIG_IO is low:

1) Internal address bits A[31:30] are not affected by this block of the translator. These bits may have been translated by the high order bit translator (see Section C.5.12.3).

2) Address bits [29:23] going out of the translator (to the PCI bus ROM counter) are set to 0000000b.

3) Internal address bits A[29:12] are passed to address bits [22:5] from the translator.

4) Internal address bits A[11:5] are not used, and are not passed through.

5) Address bits A[4:0] are always passed through unchanged.

C.5.13 ROM Read Burst Counter

The ROM read burst counter (see Figure C–20) is part of the boot ROM system, which provides the 60X CPU with read access to bytewide EPROM, EEPROM, or Flash memory devices. During a ROM read, the ROM burst counter and the ROM data shift register (see Section C.5.19) are used to stack up 8 bytes of 1-byte wide ROM data into an 8-byte wide doubleword, which is sent to the 60X CPU. The 653 Buffer also supports 32-byte (4 beat x 8 bytes/beat) burst reads from ROM. Details of these operations are found in the 654 Controller data sheet. Note that the 650 Bridge also supports writes to ROM space as described in Section C.3.6 and in the 654 Controller data sheet.

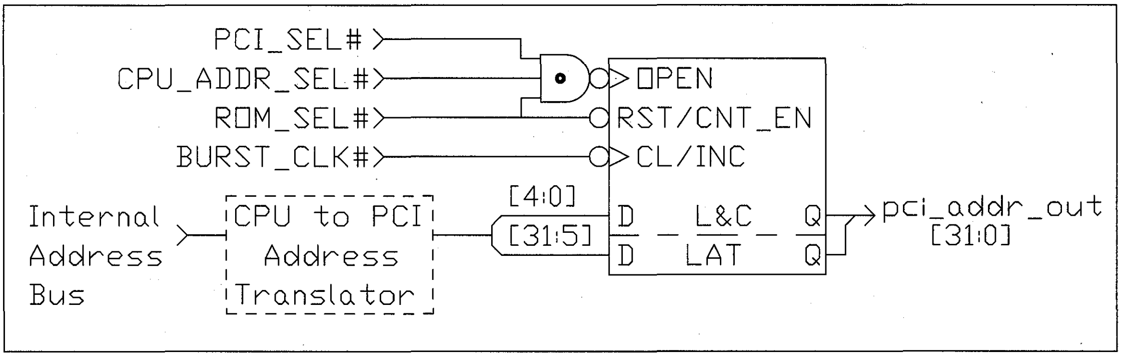


Figure C–20. ROM Read Burst Counter

The ROM read burst counter is a combination latch/counter, similar to the 60X CPU and PCI burst counters. Of the two paths through the device, the latch-only path works identically, but the latch and counter section is somewhat different.

The counter is able to count through 32 byte locations, so the five lowest order address bits [4:0] go through the latch and count section of the device, while bits [31:5] go through the latch-only section (see Figure C–21).

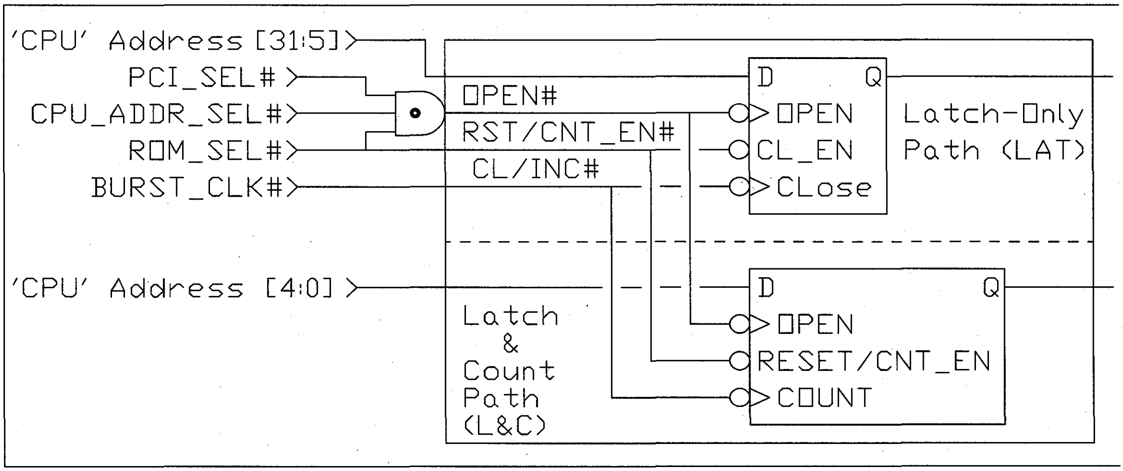


Figure C–21. Combination Latch/Counter—ROM Read Burst Counter

To track the operation of the device, assume that the initial state of the latch/counter is CLOSED. A falling edge of PCI_SEL# or CPU_ADDR_SEL# or ROM_SEL# (producing a falling edge on the *OPEN#* input) causes the latches to open, making the device transparent to all 32 bits (which is the sole function of this device during PCI and normal 60X CPU cycles). This is the OPEN state (see Figure C–22). The *OPEN#* input is edge sensitive only.

During ROM read cycles, the falling edge of CPU_ADDR_SEL# causes the latches to open, making the device transparent to all 32 bits. When the 654 Controller asserts ROM_SEL#, its falling edge forces bits [2:0] to 000b (this function is not found in the other counters). At this point, the 654 Controller strobes one byte of data out of ROM location 'x----x x000' into the ROM Data Shift Register (see Section C.5.19).

While ROM_SEL# is low, the latches in the latch–only path are enabled and the counter is enabled, so the next falling edge of BURST_CLK# latches bits [31:3] and increment bits [2:0] to 001. Next the 654 Controller strobes one byte of data out of ROM location 001 into the ROM data shift register. The 654 Controller continues to cycle BURST_CLK# and strobe data out of the ROM and into the shift register until eight bytes are read. The latch/counter's *CL/INC#* input is edge sensitive only.

Since the latch/counter is now closed, any high to low transition of PCI_SEL# or CPU_ADDR_SEL# or ROM_SEL# (on the *OPEN#* input) returns the device to the open (transparent) state.

Although the 654 Controller implements an 8-byte transfer during both single-beat and burst transactions, the 653 Buffer is capable of four-beat transfers. After the end of the first beat (counter states 0 through 7), the Controller asserts TA# to the 60X CPU to transfer the 8 bytes of data stored in the ROM shift register. Then the Controller would shift another 8 bytes into the shift register (counter states 8 through 15), TA# the 60X CPU again, and so on until all 32 bytes were transferred.

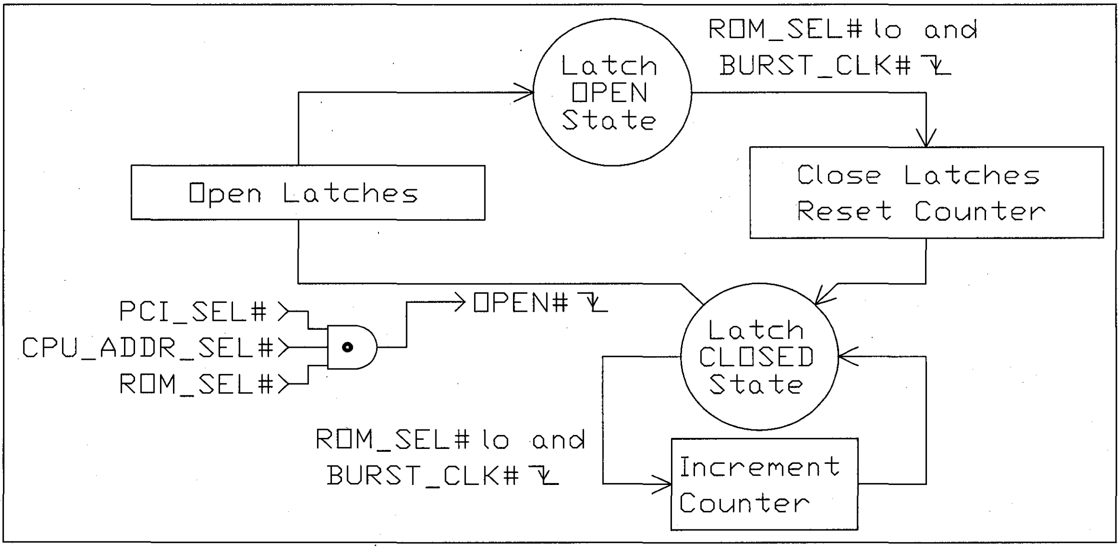


Figure C–22. Latch/Counter Flow Diagram—ROM Read Burst Counter

C.5.14 Error Address Latch

The error address latch is intended to allow system diagnostics to trap accesses to addresses that cause exceptions. Control of the latch resides in the 654 Controller, and can be enhanced by additional logic.

The latch, shown as block [14] in Figure C–2, is implemented as a hold latch, a level-sensitive, transparent D-latch. The address on the internal address bus of the 653 Buffer flows through to the *err_addr* lines as long as L_ERR_ADDR# is high. The address on the internal address bus one setup time before L_ERR_ADDR# goes low is held in the latch.

Note that L_ERR_ADDR# is asserted low to latch the error address, and must be held low to preserve the error address. The latch again becomes transparent (and the error address is lost) when L_ERR_ADDR# is negated.

C.5.15 Row/Column Address Multiplexer

The row/column address multiplexer places the required address information onto the memory address lines in the proper format, under pin control. While RASHI/CASLO is high, the multiplexer places the row address on the memory address lines. While RASHI/CASLO is low, the multiplexer places the column address on the memory address lines. While DRAMX9HI/X10LO is high, addresses appropriate to DRAMs having 9 column address bits (10x9, 11x9, or 12x9 RxC) are selected. While DRAMX9HI/X10LO is low, addresses appropriate to DRAMs having 10 column address bits (10x10, 11x10, or 12x10 RxC) are selected. Additionally, an 11th column address bit is generated, which is identical to the 12th row bit. This is useful for 11x11 addressing, or for 12x10/11 addressing, where 12x10 and 11x11 addressing is used together, such as for an 8M SIMM.

Table C–12. Memory Row and Column Address Generation

Type	DRAMX9HI/X10LO	RASHI/CAS-LO	R/C	MA11	MA10	MA9	MA8	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0
9 col	1	1	row	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12
9 col	1	0	col	0	0	0	A11	A10	A9	A8	A7	A6	A5	A4	A3
10 col	0	1	row	A24	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13
10 col	0	0	col	0	A24	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3

C.5.16 Page Hold Latch

The page hold latch is used to store the page address of the previous DRAM memory access for comparison against the page address of the next DRAM memory access. Addresses flow through the page hold latch while RASHI/CASLO# is high. An address appearing on the D inputs one setup time before the high to low transition of RASHI/CASLO# is held on the Q outputs until RASHI/CASLO# is again returned high. Also see Section C.5.9.

C.5.17 PCI Data Latch

The purpose of the PCI data latch is to capture the data from the PCI bus during the data phase of a PCI transaction. This function is accomplished using a hold latch.

The hold latch (see Figure C–23) is a level sensitive, transparent D-latch, as described in Section C.5.2. The latch is closed while L_PCI_DATA# is low and while PCI_CLK is high. Data on the

PCI_AD bus flows through the latch only while L_PCI_DATA# is high and PCI_CLK is low. To capture the PCI_AD bus data, the 654 Controller sends L_PCI_DATA# high to indicate a data phase, and the PCI_CLK transition from low to high latches the data. L_PCI_DATA# would then be sent low before the PCI_CLK again went low.

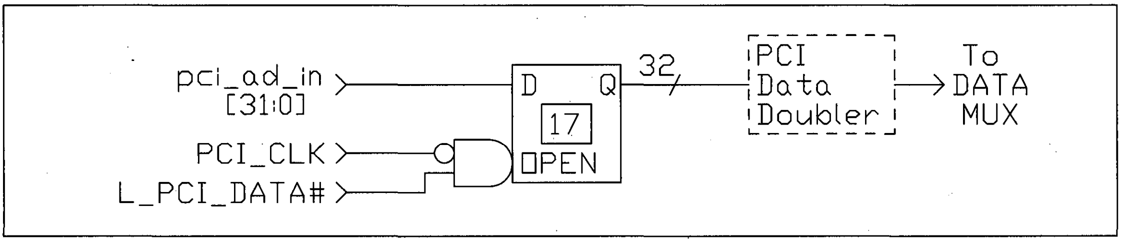


Figure C–23. PCI Data Latch

C.5.18 PCI Data Doubler

To support the transmission of data from the 4-byte wide PCI_AD bus to the 8-byte wide CPU bus, the PCI data doubler places the 4-byte data from the PCI_AD bus onto internal address lines [31:0]. It also places an identical copy of the 4-byte data on lines [63:32].

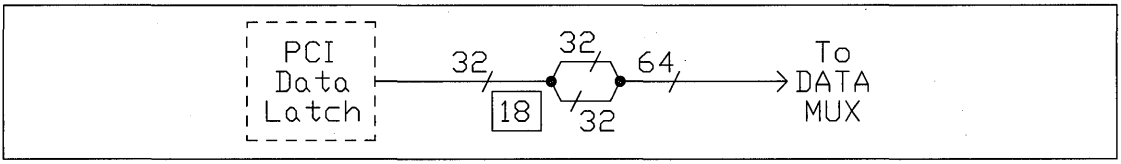


Figure C–24. PCI Data Doubler

C.5.19 ROM Data Shift Register

The ROM data shift register works with the ROM read burst counter (see Section C.5.13) to read 8 bytes of bytewide data out of the boot ROM device and transfer it to the CPU bus as 8-byte wide data. The data on the upper byte of the PCI_AD lines connects to the shift register bytewide data input (see Figure C–25).

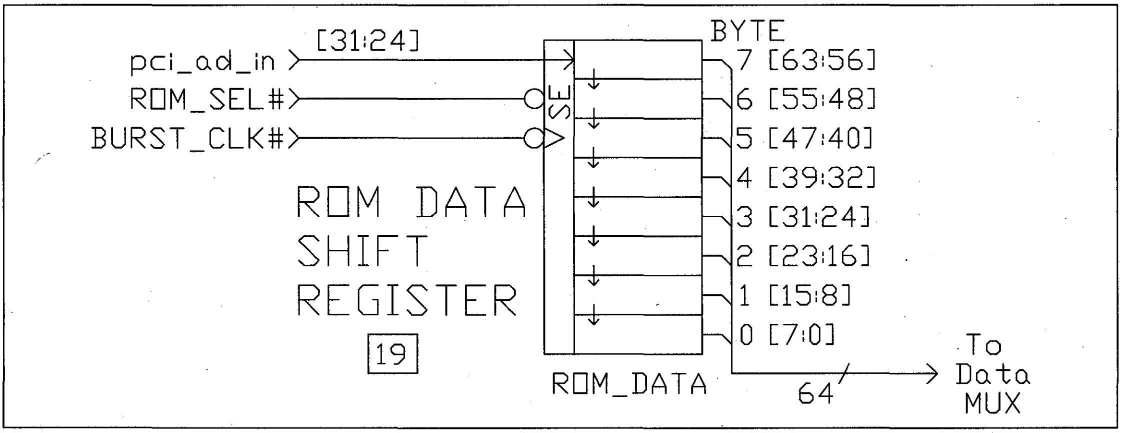


Figure C–25. ROM Data Shift Register

When ROM_SEL# is active, shifting is enabled. The bytes are shifted following the falling edge of BURST_CLK# in the following order (bit order is preserved):

1. Byte 1 is shifted into byte 0. Previous contents of byte 0 are lost.
2. Byte 2 is shifted into byte 1.
3. Byte 3 is shifted into byte 2.
4. Byte 4 is shifted into byte 3.
5. Byte 5 is shifted into byte 4.
6. Byte 6 is shifted into byte 5.
7. Byte 7 is shifted into byte 6.
8. PCI_AD[31:24] are shifted into byte 7.

Note that data on PCI_AD[31:24] is not on the internal data lines until shifted into byte 7.

C.5.20 Error Address Doubler

The error address trapping system is designed to capture addresses at which exceptions occurred. This requires getting the address information from the 4-byte wide address bus onto the 8-byte wide data bus. Maximum system flexibility was achieved by using the error address doubler to place the error address on both the high 4 bytes and the low 4 bytes of the data bus.

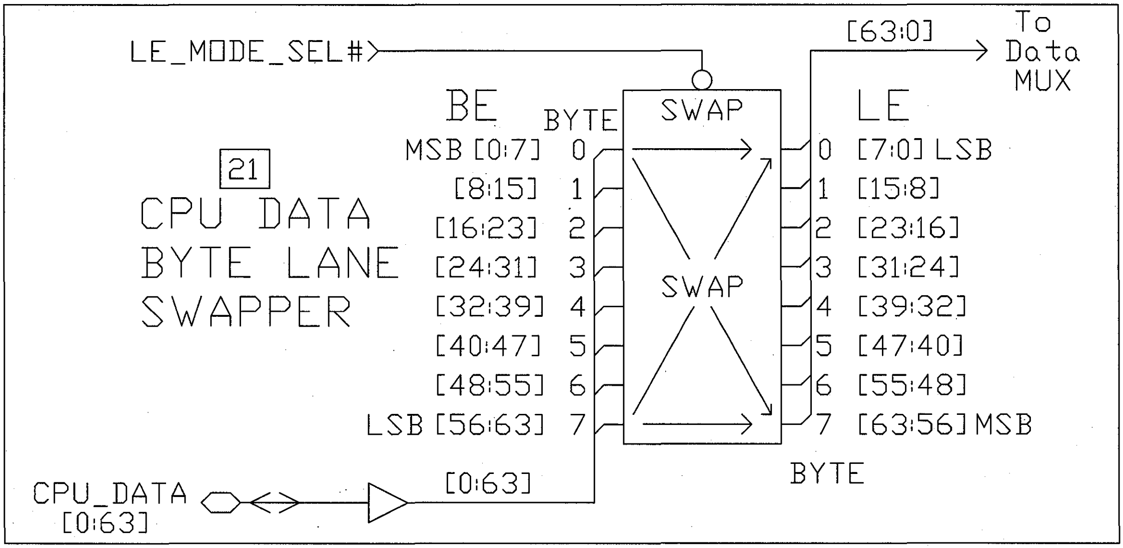


Figure C–26. 60X CPU Data Byte Lane Swapper—Input Side

C.5.21 60X CPU Byte Lane Swapper—Input Side

The 650 Bridge bi-endian operation support allows the 60X CPU to operate with either big-endian or little-endian code and data storage formats. The 60X CPU data byte lane swappers (see Figure C–26) implement the data byte reordering required to achieve bi-endian operation.

The internal nomenclature of the 653 Buffer is little-endian, while the 60X CPU bus is labeled with big-endian nomenclature. Since the 653 Buffer internal structure uses little-endian nomenclature, the 60X CPU signals are renamed on the inside of the 653 Buffer in little-endian sequence to minimize confusion.

Figure C–26 shows that the 60X CPU data bus retains its big-endian nomenclature from the 653 Buffer pins up to the input side of the swapper. As the data bytes go through the swapper onto the 653 Buffer internal address bus (by way of the data multiplexer) there is in all cases a bit-wise reversal of the numbering of the bits within the byte. (See Figure C–27.)

The reversal in bit-wise nomenclature is only a name change—there is no change in the significance of the bits. For example, a byte that has a value of A3h on the 60X CPU bus has the same value inside the 653 Buffer, and it has the same value (A3h) when it gets out of the 653 Buffer.

When the 60X CPU is operating in big-endian mode, the signal LE_MODE_SEL# is negated (high). The data on byte 0 is placed on byte 0 of the internal address bus. Byte 1 is placed on byte 1 of the internal address bus. Byte 2 goes to internal byte 2, ..., byte 7 goes to internal byte 7.

When the 60X CPU is operating in little-endian mode, the signal LE_MODE_SEL# is asserted low. The swapper is *on*. The data on byte 0 is placed on byte 7 of the internal address bus. Byte 1 is placed on byte 6 of the internal address bus. Byte 2 goes to internal byte 5, ..., byte 7 goes to internal byte 0.

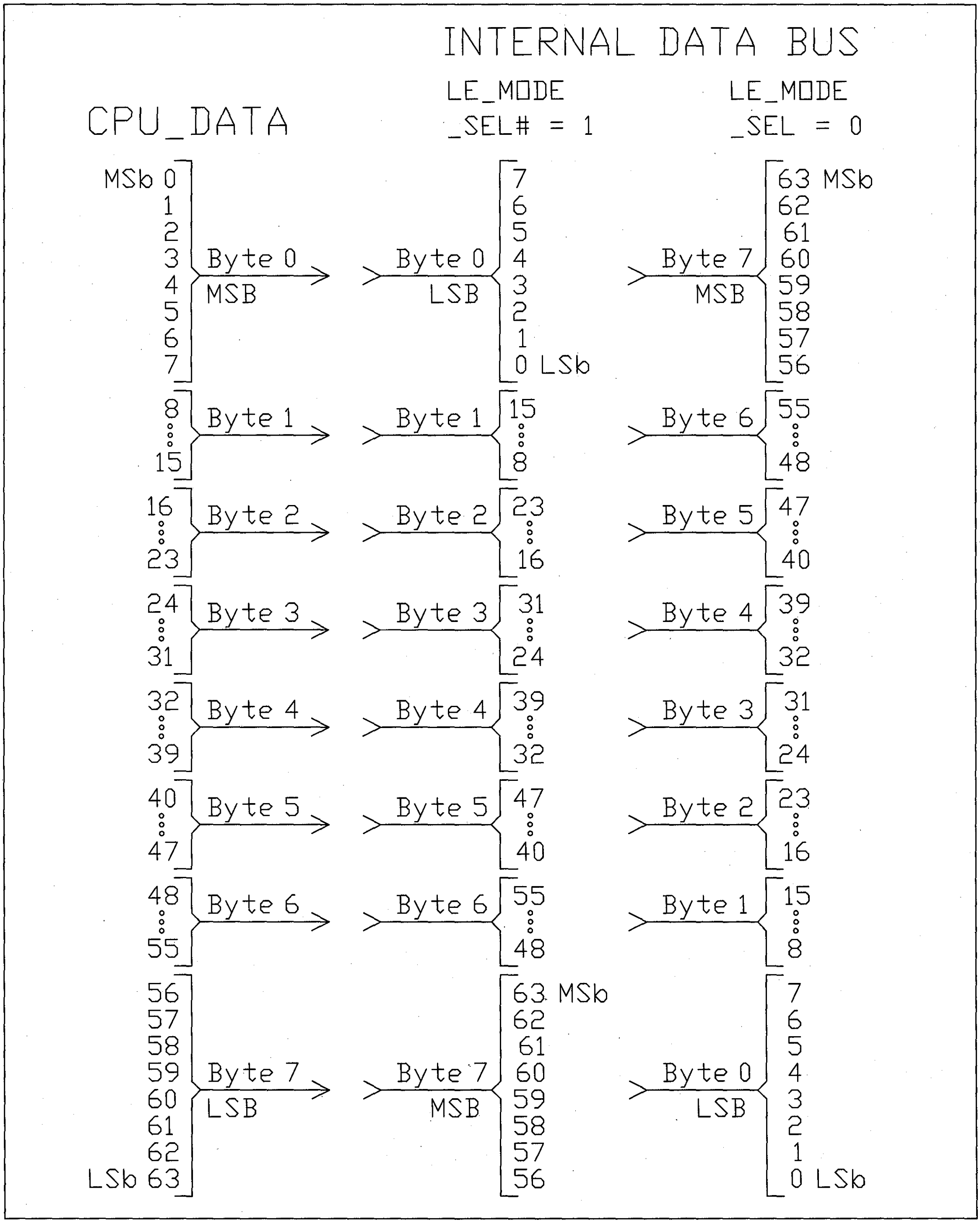


Figure C–27. CPU Data Byte Lane Swapper Operation—Input Side

C.5.22 Data Multiplexer

The 653 Buffer data multiplexer (see Figure C–28) selects one of six sources for the data appearing on the internal data bus—the PCI data latch (doubled), the ROM data shift register, the error address latch (doubled), the all_ones register (which contains FFFF FFFF FFFF FFFFh), the input side CPU data byte lane swapper, and the memory data bus. Exactly one of the sources is selected at any given time. If more than one source select line is active, the source is selected according to Table C–13.

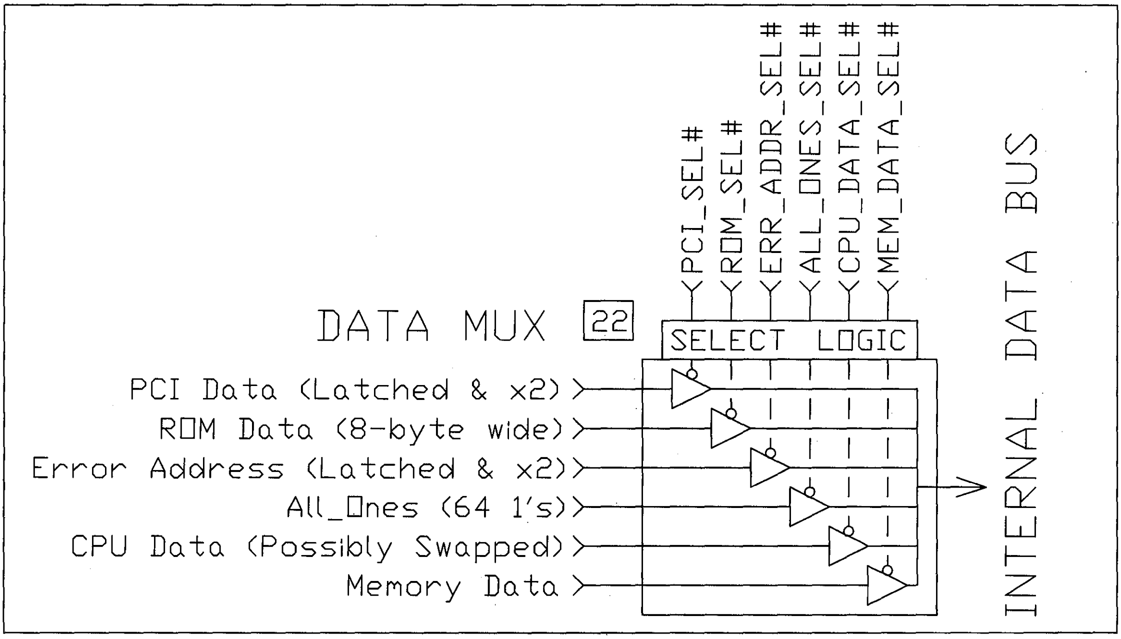


Figure C–28. Data Multiplexer

Table C–13. Data Multiplexer Source Selection Priority.

Data Source	Enable Signal	Priority
Error Address (x2)	ERR_ADDR_SEL#	0 (Top)
CPU Data	CPU_DATA_SEL#	1
All_Ones	ALL_ONES_SEL#	2
ROM Data	ROM_SEL#	3
Memory Data	MEM_DATA_SEL#	4
PCI Data (x2)	PCI_SEL#	5

C.5.23 PCI Address/Data Select Delay Flop

The ADDRHI/DATALO signal is switched by the 654 Controller in advance of the transition from PCI address phase to PCI data phase, in order to achieve the minimum clock to output time on the AD lines. The PCI address/data multiplexer (see Section C.5.24) uses this signal to switch the PCI bus between address and data information, so it is delayed until the PCI_CLK makes the low to high transition that signals the start of a data phase. This delay is implemented (see

Figure C–29) by a simple positive-edge-triggered D flipflop, which is clocked by PCI_CLK. Thus the 653 Buffer only switches the PCI_AD lines from address to data immediately following the positive edge of PCI_CLK.

Following the last data phase of the current transaction, the 654 Controller switches ADDRHI/DATALO from high to low to prepare the 653 Buffer address/data multiplexer for another PCI address phase. As above, the multiplexer actually switches the source of the PCI_AD lines immediately following the positive transition of PCI_CLK. This transition usually occurs while the PCI_AD lines are tri-stated for a PCI bus turnaround cycle.

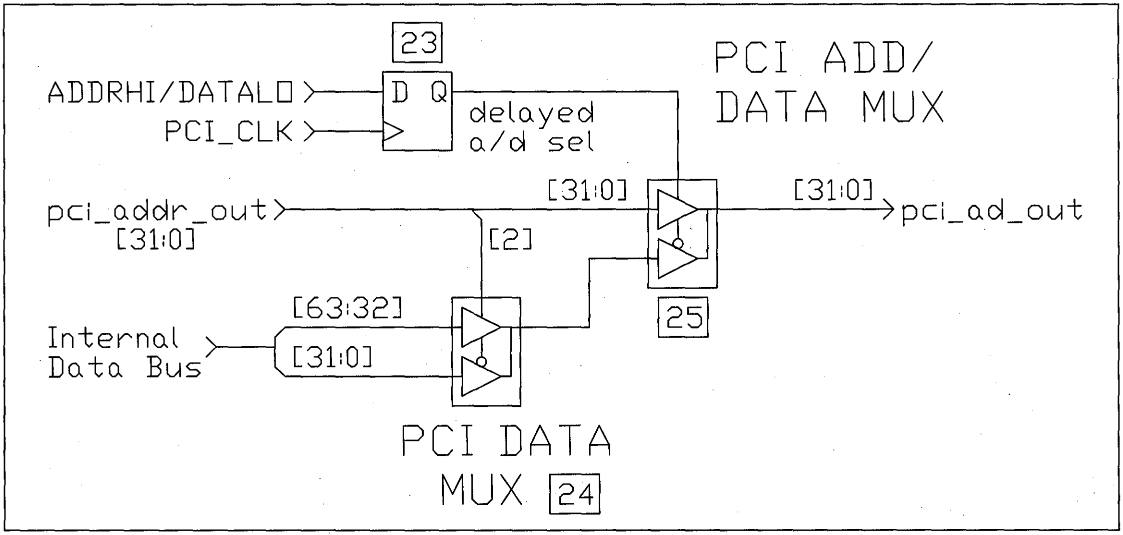


Figure C–29. PCI Delay Flop, Data Multiplexer, and Address/Data multiplexer.

C.5.24 PCI Data Multiplexer

During data flows from system memory to the PCI bus (PCI to memory reads) or from the 60X CPU to the PCI bus (CPU to PCI writes), the 653 Buffer places the 8-byte data from one of the above sources on its internal data bus (in response to the appropriate data select control signals from the 654 Controller). If *pci_addr_out[2]* (from the transaction master via the relevant translation stages) is low, internal data bits [31:0] are routed to PCI_AD[31:0] during the current data phase. If *pci_addr_out[2]* is high, internal data bits [63:32] are routed to PCI_AD[31:0] during the current data phase. The PCI data multiplexer implements this routing. The output of the PCI data multiplexer goes to the PCI address/data multiplexer. (See Section C.5.25.)

C.5.25 PCI Address/Data Multiplexer

The PCI_AD bus is a multiplexed bus. Each transaction can have an address phase and one or more data phases. The PCI address/data multiplexer routes the address information to the PCI_AD bus during the address phase, and routes the data information to the PCI_AD bus during the data phase(s). The multiplexer control line is the delayed ADDRHI/DATALO signal from the 654 Controller (see Section C.5.23). The address information enters the multiplexer on the pci_addr_out lines. This address can be the (possibly unmunged, translated, and/or burst incremented) contents of the CPU address bus, or the refresh address (possibly translated), or the ROM byte address from the ROM read burst counter. The data information enters the multiplexer from the PCI data multiplexer (see Section C.5.24), and can have come from the CPU (possibly byte swapped), the memory, or the all_ones generator. The output of the multiplexer flows onto the pci_ad_out[31:0] lines, which go to the off-chip drivers for the PCI_AD lines.

C.5.26 PCI Parity Generator

The PCI_AD bus requires an even parity signal (PAR) to be generated over AD[31:0] and C/BE#[3:0] such that the total number of 1's on AD[31:0], C/BE#[3:0], and PAR is an even number. The PCI parity generator inside the 653 Buffer generates an even parity signal (PCI_AD_PAR) for the PCI_AD[31:0] lines only (see Figure C–30). This signal and the C/BE#[3:0] lines are used by the 654 Controller to generate the PCI PAR signal.

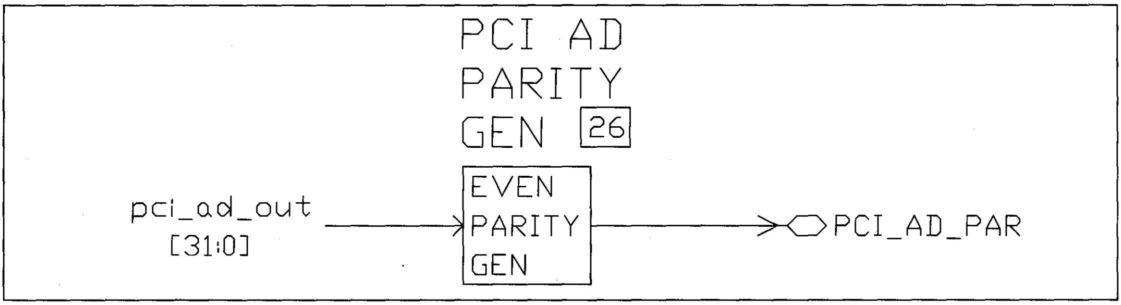


Figure C–30. PCI Parity Generator

C.5.27 60X CPU Data Byte Lane Swapper—Output Side

The 60X CPU data byte lane swapper on the output side of the data multiplexer (see Figure C–31) performs the same operation as the swapper on the input side of the multiplexer (see Section C.5.21). Like address munging and unmunging, the byte lane swap is its own inverse.

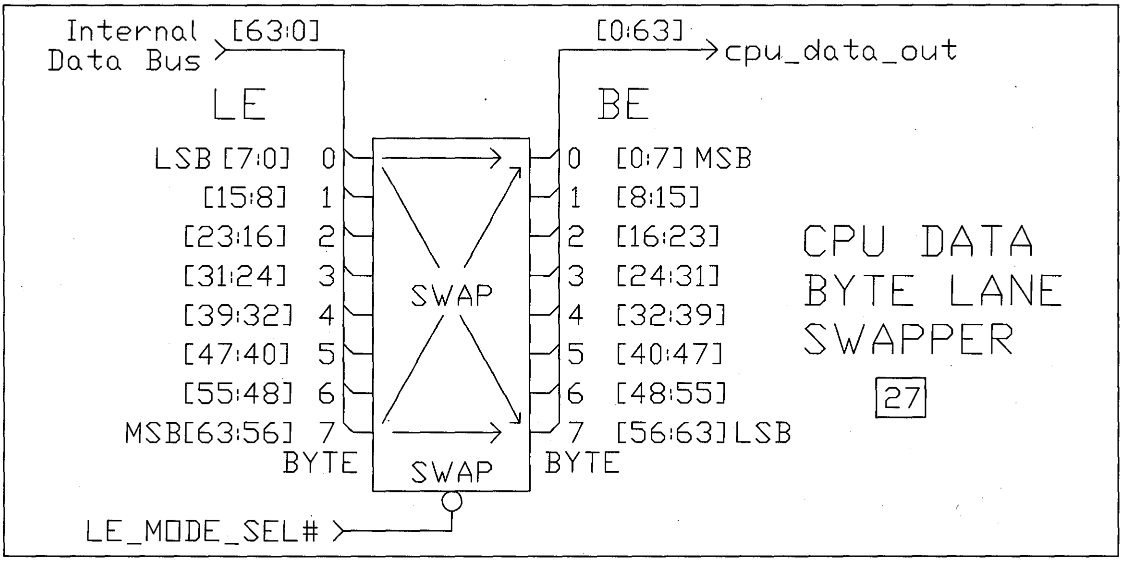


Figure C–31. 60X CPU Data Byte Lane Swapper—Output Side

Data that has been through a swapper twice on the same setting is the same as data that has not been swapped. For example, during a memory write and read from the same location when the 60X CPU is in big-endian mode (LE_MODE_SEL# = 1), data flows out of the 60X CPU through the input side swapper, producing the data arrangement shown in Figure C–27 under LE_MODE_SEL# = 1.

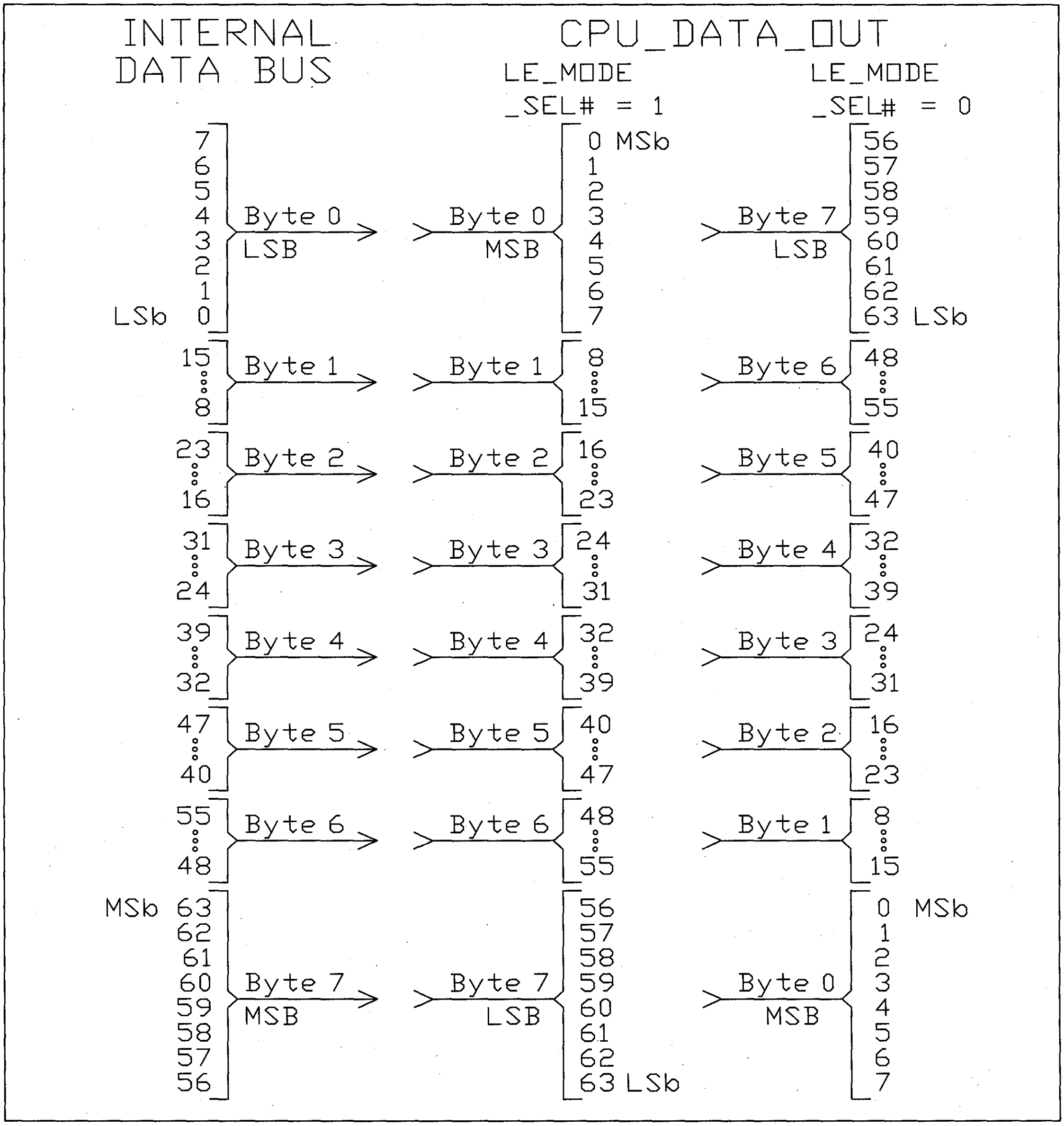


Figure C–32. 60X CPU Data Byte Lane Swapper Operation—Output Side

This is the arrangement of the data on the 653 Buffer internal data bus and in system memory. A subsequent read of the memory by the 60X CPU (still in big-endian mode) brings the data onto the internal data bus in the same arrangement. The data passes through the output side swapper on the way to the 60X CPU, and is swapped back.

Details of the output side swap operation from the internal data bus to the 60X CPU are shown in Figure C–32, under LE_MODE_SEL# = 1. The operation of the swappers (the 60X CPU in little-endian mode) is in Figure C–27 and Figure C–32 under LE_MODE_SEL# = 0.

C.5.28 Memory Data Parity Generator

The memory data parity generator (shown in Figure C–33) generates even byte parity for the eight bytes of data going to the system memory DRAMs.

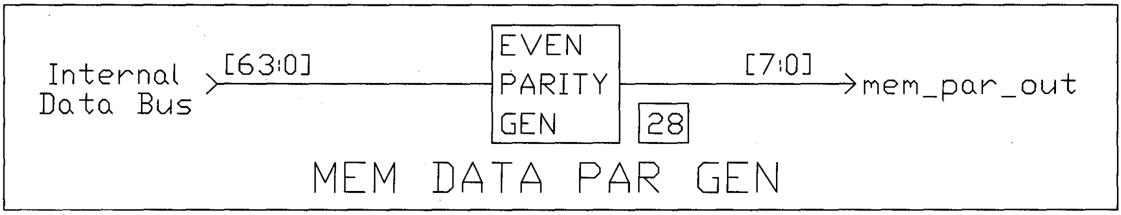


Figure C–33. Memory Data Parity Generator

C.5.29 Memory Data Parity Checker

The memory data parity checker (shown in Figure C–34) checks for even parity across the memory data and parity lines on a 1 bit per byte basis during memory operations. The output signal (MEM_PAR_GOOD) is not valid at all times. While there is no memory cycle running, MEM_PAR_GOOD has no meaning, and the 654 Controller uses MEM_DATA_SEL# to force it high. During memory cycles, MEM_PAR_GOOD is only valid one setup time after the data on both the memory data and parity lines is valid.

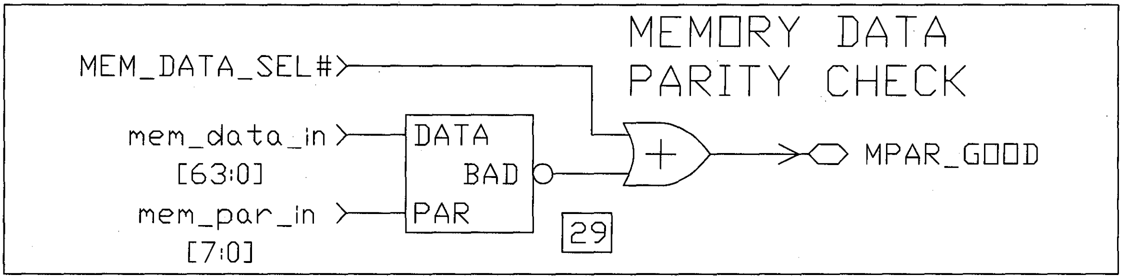


Figure C–34. Memory Data Parity Checker

Appendix D
Addresses of Sales Offices

D.1 USA

IBM Microelectronics, Mail Stop A25/862-1
PowerPC Marketing
1000 River Street
Essex Junction, VT 05452-4299
Tel: (800) PowerPC [(800) 769–3772]
Fax: (800) PowerFax [(800) 769–3732]

D.2 Europe

IBM Microelectronics
La Pompignane BP 1021
34006 Montpellier
France
Tel: (33) 6713–5757 (Français)
(33) 6713–5756 (Italiano)

IBM Microelectronics
Postfach 72 12 80
30532 Hannover
Germany
Tel: (49) 511 516 3444 (English)
(49) 511 516 3555 (Deutsche)

D.3 Japan

IBM
800 Ichimiyake
Yasu-cho, Yasu-gun
Shiga-ken, Japan 520-23
Tel: (81) 775–87–4745
Fax: (81) 775–87–4735

USA:
IBM Microelectronics Division
11400 Burnet Rd.
Austin, TX 78758
Tel: (800) PowerPC [(800) 769-3772]
Fax: (800) PowerFax [(800) 769-3732]

EUROPE:
IBM Microelectronics
La Pompignane BP 1021
34006 Montpellier
France
Tel: (33) 6713-5757 (Français)
(33) 6713-5756 (Italiano)

IBM Microelectronics
Postfach 72 12 80
30532 Hannover
Germany
Tel: (49) 511 516 3444 (English)
(49) 511 516 3555 (Deutsche)

JAPAN:
IBM
800 Ichimiyake
Yasu-cho, Yasu-gun
Shiga-ken, Japan 520-23
Tel: (81) 775-87-4745
Fax: (81) 775-87-4735

Printed in the United States of America, 7-94

IBM Order Number:
MPR650UMU-01

IBM®