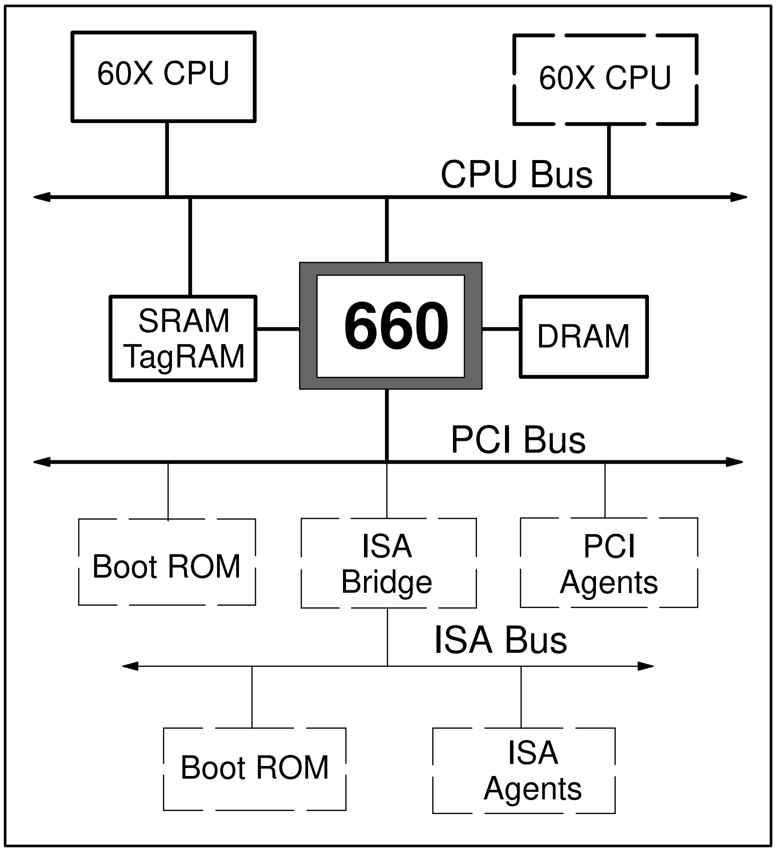
SC09-3026-01

IBM27–82660

PowerPC to PCI Bridge and Memory Controller User's Manual

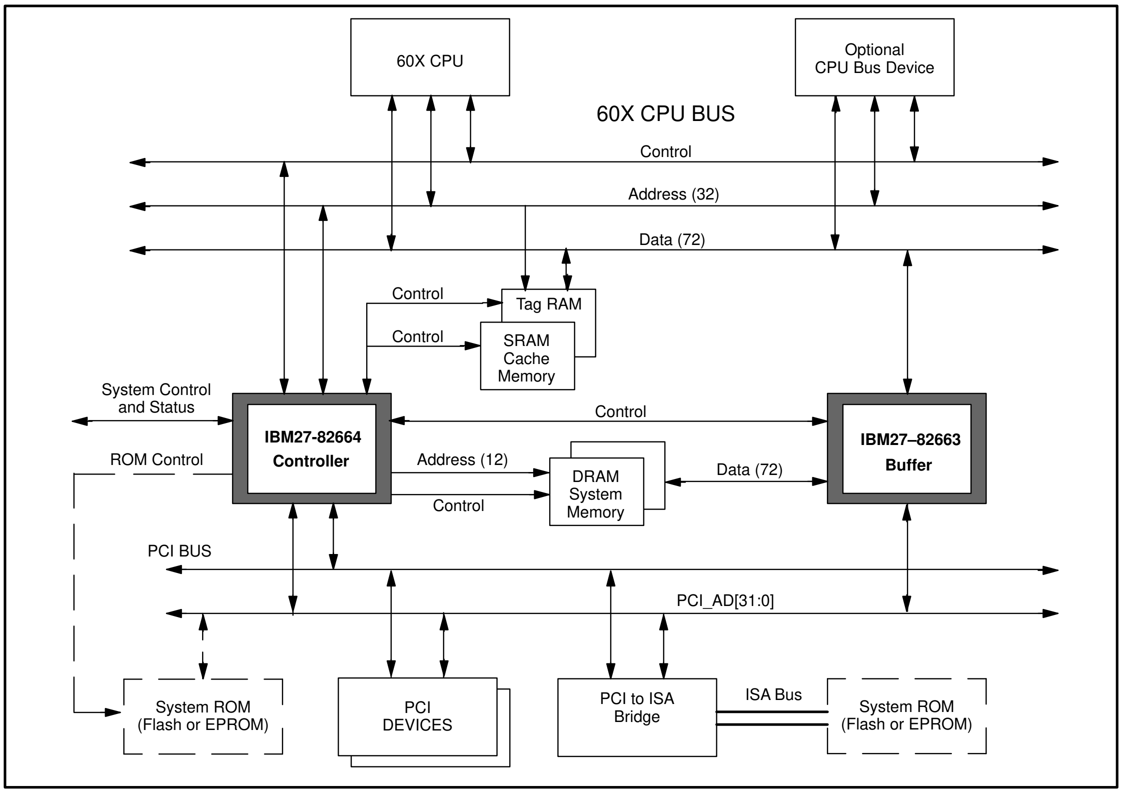


IBM

PowerPC

The IBM27-82660 PowerPC™ to PCI Bridge

Memory, L2, ROM, and System Controller



IBM 660 Bridge Chip Set in a Typical System Configuration

Minimum Cycle Times For Pipelined CPU to Memory Transfers at 66 MHz

Responding Device	Read	Write
L2 (9ns Synchronous SRAM)	–2–1–1–1	Snarf
L2 (15ns Asynchronous SRAM)	–3–2–2–2	Snarf
Page DRAM (70ns) Pipelined	–4–4–4–4	–3–3–4–4
EDO DRAM (60ns) Pipelined	–5–3–3–3	–3–3–3–3

Typical PCI to Memory Performance at 66 MHz CPU Clock and 33MHz PCI Clock

Read	8-1-1-1 -1-1-1-1 7-1-1-1 -1-1-1-1 7-1-1-1 -1-1-1-1 ... 7-1-1-1 -1-1-1-1
Write	5-1-1-1 -3-1-1-1 3-1-1-1 -3-1-1-1 3-1-1-1 -3-1-1-1 ... 3-1-1-1 -3-1-1-1

Features

The IBM27-82660 PowerPC to PCI Bridge (the 660 Bridge) interfaces the PowerPC 60x bus to the PCI bus, DRAM, and ROM, controls SRAM and tagRAM to form an L2 cache, and provides the system central resource. The 660 Bridge is PowerPC Reference Platform compliant, and includes the IBM27-82663 data buffer, and the IBM27-82664 controller.

General

- PowerPC Reference Platform 1.0/1.1
- Extensive programmability
- Flexible & programmable error handling
- Low cost plastic quad flat packs
- Dual split bus structure CPU–PCI
- Bi-Endian operation

CPU

- PowerPC 601, 603, 603e(v), 604, & 604e(v) families
- Up to 2 CPUs at 66MHz on the CPU bus
- Address pipelining
- MCP# & TEA#/INT# error reporting
- No-DRTRY#/Fast-L2 mode support.

PCI

- PCI 2.0/2.1, 33MHz, 3.3v/5v
- Memory accesses snooped to L1 & L2
- ISA master support
- PCI resource locking
- Type 0 & type 1 configuration cycles.

DRAM

- ECC or parity DRAM error checking
- Page mode or EDO DRAM
- Up to 1G DRAM with 168-pin DIMMs
- Up to 1G DRAM with 72-pin SIMMs
- Up to 8 memory banks
- Extensive programmability & flexibility
- Onboard refresh timer/counter.

L2 Cache Controller

- Look aside, direct mapped, write through
- 256k, 512k or 1M SRAM support
- Sync or asynch SRAM & tagRAM
- Can be disabled.

ROM

- Up to 2M of ROM
- Flash ROM read, write, and lock-out
- 8 to 64 bit conversion on reads
- Single-beat & burst reads.

Table of Contents

Figures

Tables

About This Book

Audience:
This book is designed for engineers who are familiar with the PowerPC family of processors and the PCI bus.

Reference Material:

- *PowerPC 604 User's Manual*, IBM document MPR604UMU-01.
- *PowerPC 603 User's Manual*, IBM document MPR603UMU-01.
- *PowerPC 603e User's Manual*, IBM document MPR603EUMU-01.
- *PowerPC 601 User's Manual*, IBM document MPR601UMU-02.
- *PCI Local Bus Specification*, Revision 2.1, available from the PCI SIG.
- *32MB SIMM Engineering Specification,* IBM document number MMDL02DSU-00.
- *8MB SIMM Engineering Specification,* IBM document number MMDL01DSU-00.
- *PowerPC Reference Platform Specification,* Version 1.1, IBM document MPRPRPPKG-02.
- *The Power PC Architecture,* second edition, Morgan Kaufmann Publishers (800) 745–7323, IBM document MPRPPCARC–02.
- *PowerPC System Architecture*, Tom Shanley, Mindshare, Inc., Addison–Wesley Publishing 1–800–822–6339 (Order # 0–201–40990–9).
- *IBM PowerPC 603/604 Reference Design Technical Specification*, IBM document MPRH01TSU-02.
- *PowerPC 604 SMP Reference Design Technical Specification*, IBM document MPRZAPTSU-04.
- *Open Programmable Interrupt Controller (PIC) Register Interface Specification*, Revision 1.2.
- FAQ and application notes are available at http://www.chips.ibm.com/products/ppc/apnote_files/index.html

The example implementation schematics have been replaced by the *IBM PowerPC 603/604 Reference Design Technical Specification* and the *PowerPC 604 SMP Reference Design Technical Specification*.

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.

In this document a word is 32 bits, a double-word is 64 bits, and a half-word is 16 bits.

The term SIMM is often used to mean DRAM module.

Hexadecimal values are identified (where not clear from context) with a lower-case letter h at the end of the value. For example, 001Fh means a hex value of 1F. Binary values are identified (where not clear from context) with a lower-case letter b at the end of the value. For example, 0101b means a 4-bit binary value of 0101.

The range statement from 0 to 2M means from and including zero up to (but not including) two megabytes. The hexadecimal value for the range from 0 to 64K is: 0000h to FFFFh.

The terms *asserted* and *negated* are used extensively. The term *asserted* indicates that a signal is active (logically true), regardless of whether that level is represented by a high or low voltage. The term *negated* means that a signal is not asserted. The # symbol at the end of a signal name indicates that the active state of the signal occurs with a low voltage level.

Signal ranges are always shown with the most significant number first (SIGNAL[MSb:LSb]). Signals ranges that have the first number greater than the second number (PCI_AD[31:0]) are shown in little-endian nomenclature. Those shown with the second number greater than the first (TT[0:3] for example), are shown using big-endian nomenclature. Signal range names used without an explicit range indication refer to the entire set of signals (PCI_AD means PCI_AD[31:0]).

Acronyms and Abbreviations:
The term 60X CPU and CPU refers to the PowerPC 601, 603, and 604 families of microprocessors, generally including "e" versions.

In general, the 660 supports PowerPC 740 and 750 as members of the 603 family and PID9q-, PID9v-, PID9t-, and PID10q-604e as members of the 604e family.

The term I/O Bridge or I/O Bus 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 write-back means the same as copy-back in reference to a mode of cache operation.

The IBM 27-82660 is generally referred to as the Bridge, the 660 Bridge, or the 660; likewise, the IBM27-82664 is generally called the 664 Controller or the 664. Also, the IBM27-82663 is generally called the 663 Buffer or the 663.

Contributors:
This document was written and edited by Ben Drerup and Dale Elson, with substantial contributions by Robert Stevens, Sean Curry, and the Kauai-Lanai design, simulation, and testing team.

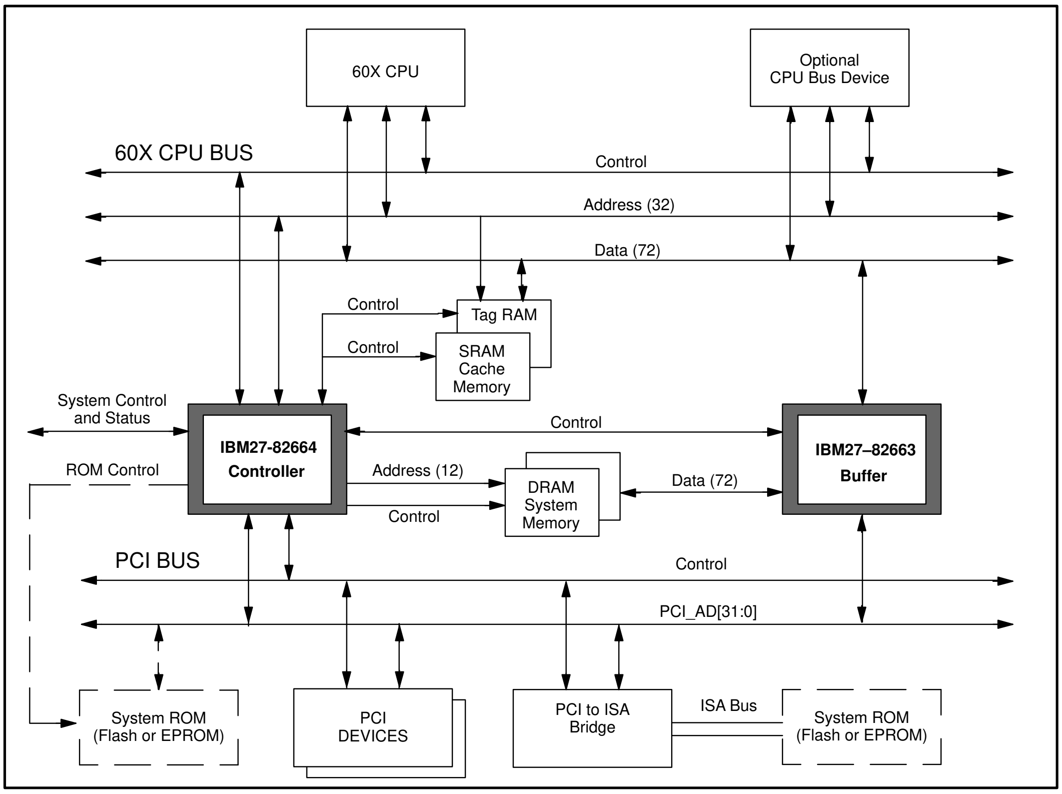


Figure 1-1. IBM 660 Bridge Chip Set in a Typical System Configuration

Section 1 Overview

This section summarizes the features of the 660 Bridge—including microprocessor support, the memory controller, the PowerPC CPU bus, the PCI expansion bus, the L2 cache controller, the system ROM controller, the bridge control register set, and the interrupt and exception handler.

- Complies with the PowerPC Reference Platform specification, versions 1.0 & 1.1
- Complies with the PCI Revision 2.0 & 2.1 specification.

1.1 Packaging and Technology

- Designed in IBM YASU 3.3 to 3.6V CMOS4LP logic which allows I/Os that are compatible with 3.3V and 5.0V logic
- 208-pin (664) and 240-pin (663) low-cost plastic wire-bond flatpacks
- Operates from 3.0V to 3.78V, allowing either 3.3V or 3.6V power sources.

1.2 Microprocessor Support

- Supports PowerPC 601, 603, 603e(v), 604, 604e(v), 740, 750, and 760 microprocessor families
- Supports CPU bus speeds up to 66Mhz. Internally, the CPU can run at a faster rate
- Directly supports two CPU bus masters
- Supports little-endian and big-endian operating modes
- 64 bit wide CPU data bus and 32 bit wide CPU address bus
- Utilizes address bus pipelining
- Dual bus structure between CPU bus and PCI bus
- Data path latches enhance effective bandwidth
- Supports all clock modes for CPUs (internal CPU clock to CPU bus)
- The MCP# signal on the 603 and 604 is supported for error reporting
- For reduced read latency, the no-DRTRY# mode of the 604 is supported
- The 603 no-DRTRY# mode is also supported except when the 603 is in 1:1 mode
- The 601 is always in DRTRY# mode
- Error reporting is by means of TEA# or MCP# on the 603 and 604
- Error reporting is by means of TEA# and INT_CPU# on the 601
- The 604 store multiple 8-byte transfer is supported to the PCI bus

1.3 L2 Cache Controller

- Controls external burst-mode or asynchronous SRAMs for cached data
- Controls external tag RAMs for tag information
- Look-aside, direct-mapped, and write-through protocols
- Supports all cache sizes
- Externally determined cache size
- Responds to snoop cycles for PCI reads and writes of system memory
- Option to check parity of data stored in L2 on read cycles
- L2 cache controller can be disabled if external L2 cache is used.

1.4 PCI Expansion Bus

- PCI bus frequency up to 33 MHz. CPU:PCI bus ratios of 1:1 and 2:1 supported.
- 32-bit multiplexed PCI address and data path
- Support for I/O bus bridge (ISA, EISA, MicroChannel™)
- PCI to DRAM access—with L1 and L2 cache snooping
- Support for memory mapping of 60X address space into PCI transactions
- Supports ISA bus master access to system memory with ISA bridge on the PCI bus
- Supports contiguous ISA I/O and non-contiguous ISA I/O mappings (non-contiguous I/O so operating systems can memory-protect 32-byte blocks of ISA I/O space)
- Uses external PCI arbiter (usually from the I/O bridge chip)
- Support for PCI resource locking of system memory
- Supports type 0 and type 1 PCI configuration cycles.

1.5 Memory Controller

- Supports memory operations for the PowerPC Architecture™
- Data bus path 72 bits wide—64 data bits and eight bits of optional ECC or parity data
- Supports Eight SIMM sockets, with empty SIMM sockets allowed at any position
- Eight RAS# outputs, eight CAS# outputs, and two write-enable outputs
- Supports industry-standard 8-byte (168-pin) SIMMs of 8M, 16M, 32M, 64M, and 128M that can be individually installed for a minimum of 8M and a maximum of 1G
- Supports industry-standard 4-byte (72-pin) SIMMs of 4M, 8M, 16M, 32M, 64M, and 128M that must be installed in pairs for a minimum of 8M and a maximum of 1G
- Programmable memory timings enable optimization of DRAM timings for a large variety of CPU bus frequencies, DRAM speeds, and system topologies
- Mixed use of different size SIMMs, including mixed 4-byte and 8-byte SIMMs
- Support for parity, ECC, or neither
  - Generates ECC or odd parity, eight bits for eight bytes, on all memory writes
  - Checks ECC or parity eight-bytes wide on all memory reads
  - Detects and corrects all single-bit errors in ECC mode
  - Detects all two-bit errors in ECC mode
- Page-mode access—fast page-mode is supported
- Support for extended-data-out (EDO) DRAM (hyper-page mode) for higher bandwidth memory performance
- Provides row-address and column-address multiplexing for SIMMs requiring:
  - 10 row by 10 column
  - 11 row by 10 column
  - 12 row by 10 column and 11 row by 11 column (supported simultaneously)
  - 12 row by 11 column
  - 12 row by 12 column
- Non-interleaved memory access operation
- Programmable DRAM refresh timer with low-power mode
- Memory refresh address counter
- Burst-mode memory address generation logic
  - 32-byte CPU bursts to and from memory
  - Variable length PCI burst to and from memory
- Little-endian and big-endian modes
- Supports ISA master to DRAM access

Minimum Cycle Times For Pipelined CPU to Memory Transfers at 66 MHz

Responding Device	Read	Write
L2 (9ns Synchronous SRAM)	–2–1–1–1	Snarf
L2 (15ns Asynchronous SRAM)	–3–2–2–2	Snarf
Page DRAM (70ns) Pipelined	–4–4–4–4	–3–3–4–4
EDO DRAM (60ns) Pipelined	–5–3–3–3	–3–3–3–3

Other minimum timings at 66MHz CPU and 33MHz PCI and 70ns page mode DRAM:

- PCI to memory read:
  8-1-1-1 -1-1-1-1 7-1-1-1 -1-1-1-1 7-1-1-1 -1-1-1-1 ... 7-1-1-1 -1-1-1-1 (PCI clocks)
- PCI to memory write:
  5-1-1-1 -3-1-1-1 3-1-1-1 -3-1-1-1 3-1-1-1 -3-1-1-1 ... 3-1-1-1 -3-1-1-1 (PCI clocks)

1

1.6 System ROM Controller

- Supports ROM attached to PCI_AD bus
- Supports 8-bit flash ROM
  - Provides 8-bit to 64-bit conversion on reads
  - 21-bit address support for up to 2M addressable ROM
- Flash ROM write cycles generated for in-system flash ROM writes
- Flash ROM 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.

1.7 The Bridge Control Register Set

- Implemented PCI register model
- Includes 650 Bridge-compatible registers
- Configuration through register 0CF8h and 0CFCh configuration method
- Chip set options are configured in the register set
- Allows extensive and flexible programming of the 660.

1.8 Interrupt and Exception Handler

- 603/604 error reporting by means of TEA# or MCP#
- 601 error reporting by means of TEA#, or INT_CPU# and TEA#
- Supports error address and control capture registers
- Reports the following types of errors:
  - Memory parity or ECC error
  - CPU illegal transfer
  - PCI bus parity error
  - CPU data bus parity error
  - PCI cycle abort
  - L2 cache parity error
  - Write to locked flash ROM
  - Memory access out of range
- Drives CPU data lines to all one-bits on out-of-range memory reads
- PCI configuration read cycles return all one-bits when no device responds.

1.9 Part Identification, IBM27-82660

The release version of the 660 is revision 2.2.

IBM27-82660	IBM27-82663 (663 Buffer)		IBM27-82664 (664 Controller)	
Chipset Revision	Revision	Package Marking	Revision	Package Marking
1.0	1.0	94G0235	1.0	94G0232
1.1	1.0	94G0235	1.1	94G0176
2.0	2.0	94G0178	1.1	94G0176
2.2	2.0	94G0178	1.2	20H2842

1.10 Improvements Over 650 Bridge

- Includes L2 cache controller that supports synchronous and asynchronous SRAMs
- System performance improvements:
    - Programmable memory controller optimizes to memory speed and topology
    - Support for high-bandwidth memory technology (extended-data-out DRAM)
    - Up to 1G addressable DRAM
    - Utilizes CPU bus address pipelining
    - Dual bus structure between CPU bus and PCI bus
    - Data path latches provide bandwidth improvements
- ECC memory support
- Supports 603 in 1-to-1 mode (60X internal clock vs. CPU bus)
- The 604 store multiple 8-byte transfer is supported to the PCI bus
- Low-cost packaging—a 208-pin and a 240-pin plastic quad flatpack
- Uses an external PCI bus arbiter
- Provides basic multi-processor support
- Supports type 1 PCI configuration cycles in addition to type 0
- Implements Expanded error control and reporting
- Provides a DRAM refresh timer
- Implements PCI-compatible configuration register set.

1.11 Chipset Changes From Prerelease Versions

This section describes changes that were made to the 660 bridge during development, especially as reflected in the differences between this document (SC09-3026-00) and A previous revision (MPR660UMU-01) of the 660 User's Manual. Many of these items were noted in *The IBM27-82660 Revision 2.1 Product Update* (MPR660ESU-04). At the time of the printing of this document, there were no known errata associated with the 660.

1.11.1 CPU_RDL_OPEN Resistor

Add a 200Ω series resister to the CPU_RDL_OPEN net between the 664 and the 663. See section 2.2.

1.11.2 Parity Error Detection

Parity errors on the CPU and memory data busses are only detected by the 663 if there are an odd number of bit errors. All single bit errors are detected. See section 5.4.1.

1.11.3 ECIWX/ECOWX With 603

ECIWX and ECOWX are not supported for use in systems containing a 603 or 603e that is running at a 1:1 or 3:2 CPU core:bus clock ratio. These instructions are supported for systems featuring a 601, 604 or 604e CPU.

1.11.4 Error Simulation Registers

Error simulation BCRs 1 and 2 are not supported, and have been removed from the BCR section.

1.11.5 DRAM Performance

The DRAM performance and memory controller programming guidelines in the DRAM section have changed, as shown especially in section 5.2, DRAM Performance.

A special case has been identified which is described in section 5.2.4. Designers may wish to avoid this case or to incorporate a performance enhancement strategy.

1.11.6 CAS# Pulse Width

Setting the CAS# pulse width to four CPU clocks is not supported. See the Memory Timing Register 2 description (section 10.3.29).

1.11.7 CPU:PCI Bus Ratio

Operation of 660 systems at a CPU:PCI bus ratio of 2:1 is supported. Operation at 3:1 is not supported, and has been removed from the Introduction (section 1.4) and PCI Bus (section 4) sections.

Operation at 1:1 is supported only with the use of external logic, and only while the PCI arbiter is programmed to not park the PCI bus on the 660 (see section 4.6).

1.11.8 Parking the PCI Bus on the 660

If the CPU:PCI clock ratio is 2:1, and the PCI arbiter is programmed to park the PCI bus on the 660 (see section 4.1), and the 660 is in one of the following configurations:

1. Configuration 1:
   - 1.1 Possible 70ns timings (see section 5.2.1), of:
     - 1.1.1 RAS# precharge (RP) = 4 (CPU clocks), and
     - 1.1.2 RAS# to CAS# delay (RCD) = 3, and
     - 1.1.3 CAS# pulse width (CPW) = 3. And
   - 1.2 L2 type set to Asynchronous (see section), or
2. Configuration 2:
   - 2.1 Possible 70ns timings (see section 5.2.1), of:
     - 2.1.1 RAS# precharge (RP) = 4, and
     - 2.1.2 RAS# to CAS# delay (RCD) = 3, and
     - 2.1.3 CAS# pulse width (CPW) = 3. And
   - 2.2 ECC memory checking enabled (see section), or
3. Configuration 3:
   - 3.1 Possible 60ns timings (see section 5.2.1), of:
     - 3.1.1 RAS# precharge (RP) = 3, and
     - 3.1.2 RAS# to CAS# delay (RCD) = 3, and
     - 3.1.3 CAS# pulse width (CPW) = 3. And
   - 3.2 L2 type set to Asynchronous (see section), and
   - 3.3 ECC memory checking enabled (see section),

then program the PCI arbiter to not park the PCI bus on the 660. This typically results in adding two PCI clocks to the latency of CPU to PCI transactions, and typically reduces the latency of PCI to memory accesses by one PCI clock.

1.11.9 CPU Data Bus Parity With 64-Bit L2

When using 64-bit L2 SRAM, CPU bus parity error detection and L2 cache parity error detection must be disabled. No changes to operation with 72-bit SRAM or no SRAM. See section 6.4, L2 Error Checking Support.

1.11.10 ECC Single-Bit Error Counter BCR(B8h)

The bit ordering of this BCR is reversed. See section 10.3.33, Single-Bit Error Counter Register.

1.11.11 663 Pinout

The pinout of the 663 was rearranged to reduce ground bounce effects by distributing simultaneously switched signals more evenly around the perimeter of the device.

1.11.12 Added Signals

DBG# was added to the 664 Controller to support a 604 in fast L2 mode. This change is documented in the MPR660UMU-01 release of the 660 Bridge User's Manual, and is noted here for completeness (see Section 3.8).

PCI_TRDY# was connected to the 663 to improve performance.

1.11.13 Remote ROM

Remote ROM is supported as detailed in section 7. Information about remote ROM operation was removed from the -02 version of this document. All references to remote ROM operation have been restored to the documentation.

1.11.14 Power Management

Power management is not supported. Power management has been removed from the Additional Information section. The description of the Bridge Chipset Options 2 BCR (section10.3.36) has been changed. The Suspend Refresh Timer Register BCR(D3:D2) has been removed.

1.11.15 VI Curves

VI curves for the output drivers are not supplied by IBM. SPICE model information is available from IBM under Non-Disclosure Agreement. Contact your IBM technical representative for more information.

1.11.16 CPU Data Bus Resistors

To improve output signal quality, 33Ω series resistors were added to the 663 buffer connections to the CPU data bus lines CPU_DATA[0:63] and CPU_DPAR[0:8]. Each resister is placed between the 663 pin and all other CPU bus agent(s) attached to the net. For maximum benefit, place the series resistors as close to the 663 package as is feasible.

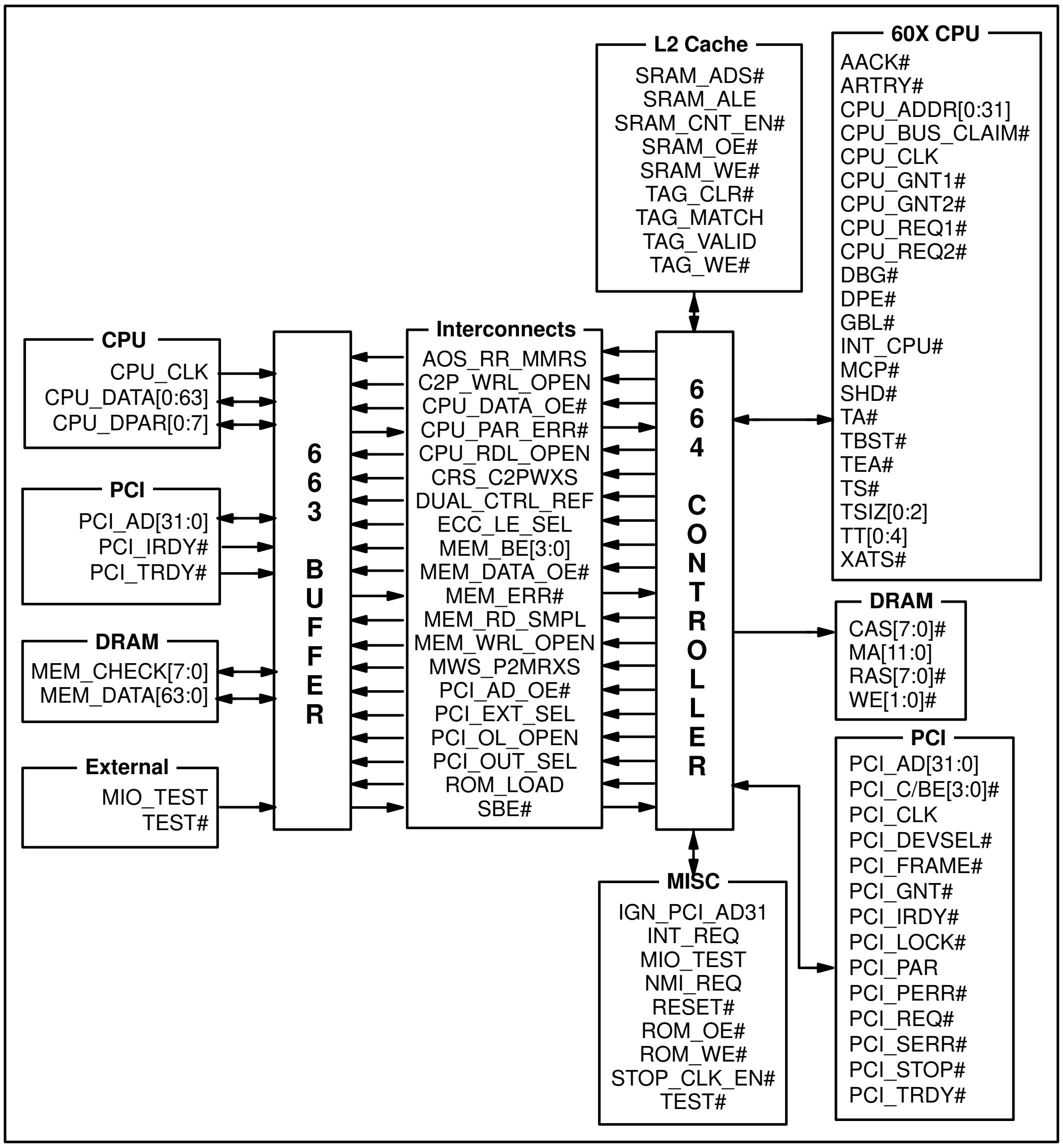


Figure 1-2. 660 Bridge Pin Connections

2

Section 2
Pin Descriptions

This section describes the connectivity of the 660 Bridge. See Figure 1-2.

The terms *asserted* and *active* indicate that a signal is logically true, regardless of the voltage level. The terms *negated*, *inactive*, and *deasserted* mean that a signal is logically false.

The # symbol at the end of a signal name listed in Table 2-1 indicates that the active or asserted state of the signal occurs with a low voltage level. Otherwise the signal is active at a high voltage level.

Pins which are marked s/t/s are sustained tri-state. This 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 active low must actively drive it high before letting it float. On the PowerPC bus, this is known as the restore function. On the PCI bus, the agent must drive the signal high for one clock and then tristate it. 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 maintain the inactive state.

Appendix C contains numeric and alphabetic lists of the pins in the 663 Buffer and 664 Controller. These lists include pin numbers.

2.1 Signal Description Table

Table 2-1. 660 Bridge Signal Descriptions

Signal Name	663	664	Description
CPU Bus Interface			
AACK#	—	I/O 109	CPU address acknowledge. 660 asserts AACK# to signal the end of the current address tenure. AACK# is an input to the 664 when a CPU bus target claims the current transaction by means of CPU_BUS_CLAIM#.
ARTRY#	—	I/O 110	Address retry. ARTRY# is asserted by a CPU bus device to signal that the current address tenure needs to be rerun at a later time. The 660 samples ARTRY# on the second clock after TS# is sampled active. The 660 will only assert ARTRY# on the clock after it asserts AACK# (during a PCI retry).

2

Table 2-1. 660 Bridge Signal Descriptions (Continued)

Signal Name	663	664	Description
CPU Bus Interface			
CPU_ADDR[0:31]	—	I/O see App C	CPU address bus. Represents the physical address of the current transaction. Is valid from the bus cycle in which TS# is asserted through the bus clock in which AACK# is asserted. CPU_ADDR is an input to the 664 on transactions initiated by a CPU bus master. The the CPU bus target responds with AACK# when the address is no longer required, ending the address tenure. CPU_ADDR is an output from the 664 on system memory transactions initiated by a PCI bus master device. The 664 initiates an address tenure to snoop the address requested by the PCI bus master.
CPU_BUS_CLAIM#	—	I 132	CPU bus claim. This signal is asserted by a CPU bus target to claim a CPU bus memory transaction. It inhibits the 664 from driving AACK, TA#, TEA#, and the CPU data bus lines. This signal is sampled by the 664 on the second CPU_CLK after TS# is sampled active. CPU bus targets can only map to system memory space (0 to 2G) and only to memory space that is not cached by the L2. The L2 caches as much of the space from 0 to 2G as is populated by DRAM. So, if 8M is installed starting at 0, the CPU_BUS_CLAIM# can be asserted from 8M to 2G. If the internal L2 is disabled, then the entire 0 to 2G memory space can be claimed by CPU_BUS_CLAIM#.
CPU_CLK	I 157	I 121	CPU bus clock. The 660 Bridge supports up to a 66Mhz CPU bus clock frequency. The CPU_CLK frequency must be an integer multiple (1x, 2x, or 3x) of PCI_CLK.
CPU_DATA[0:63]	I/O	—	The 64-bit 60X CPU data bus. CPU_DATA[0] is the most-significant-bit. 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]. Connect a 33Ω series resister to each CPU_DATA net between the 663 and all other CPU bus agents. Place these resisters as close to the 663 as possible.
	see App C		
CPU_DPAR[0:7]	I/O	—	60X CPU data parity bus. The most-significant-bit is CPU_DPAR[0], the least-significant-bit is CPU_DPAR[7].
	see App C		
CPU_GNT1#	—	O 134	CPU bus grant. CPU_GNT1# is the grant line for the CPU_REQ1# request line.
CPU_GNT2#	—	O 135	CPU bus grant. CPU_GNT2# is the grant line for CPU_REQ2#.
CPU_REQ1#	—	I 127	CPU bus request. CPU_REQ1# is the request line for the primary CPU on the CPU bus.
CPU_REQ2#	—	I 128	CPU bus request. CPU_REQ2# is the request line for a secondary CPU bus master, such as a secondary CPU or other device.
DBG# (see Section 1.5)	—	O 140	CPU data bus grant. The 660 Bridge asserts DBG# (while ARTRY# is inactive) to signal that the requesting CPU may take ownership of the CPU data bus. This signal must be connected to the CPU only in the case of a 604 using fast L2 data streaming mode. Else this signal may be either connected to the CPU or a no–connect at the 660 Bridge and pulled low at the CPU. Do not use no-DRTRY# mode with multiprocessors or with PID6-603e at >40MHz bus.

Table 2-1. 660 Bridge Signal Descriptions (Continued)

Signal Name	663	664	Description
CPU Bus Interface			
DPE#	—	I 133	Data parity error from CPU. The processor checks data parity and reports any data beat with bad parity two CPU bus clocks after the TA# for that data beat. CPU bus parity is one odd parity bit for every byte—eight data parity bits total.
GBL#	—	O t/s 120	Global. This signal is asserted during snoop operations to indicate that CPU bus masters must snoop the transaction. The 660 Bridge does not monitor GBL# (see Section 6.2).
INT_CPU#	—	O 139	CPU interrupt. The 664 asserts INT_CPU# to signal the processor to run an interrupt cycle. The software is expected to run a read PCI interrupt acknowledge transaction in response to INT_CPU# being asserted. The 660 Bridge can assert INT_CPU# in response to an INT_REQ input. In 601 error reporting mode, the 660 can assert INT_CPU# in response to various error conditions.
MCP#	—	O o/d 138	Machine check pin. When a 603 or 604 CPU is used with the 664. this pin is driven to notify the CPU of a system error from a source not affiliated with the current transaction. The 601 does not implement the MCP# pin, therefore the 664 drives an INT_CPU# to the CPU, and terminates the resulting PCI interrupt acknowledge request with TEA#. The 664 asserts MCP# in the event of a catastrophic or unrecoverable system error. This signal is asserted for two CPU bus cycles.
SHD#	—	O t/s 141	Shared. The function of this pin is to restore the SHD# net to a high state after it has been asserted (the same as ARTRY# restore). The restore is enabled by means of bit 4 of the memory controller timing programming register (8000 0821h).
TA#	—	I/O 111	CPU bus transfer acknowledge. TA# signals that a data transfer has occurred. For every CPU clock that TA# is asserted, a data beat completes. For a single-beat cycle, TA# is only one clock. For a four–beat burst, the data tenure is complete on the fourth cycle in which TA# is asserted.
TBST#	—	I/O 144	Transfer burst. TBST# indicates a burst transfer of four 64-bit double-words on the 60X CPU bus. The 664 does not assert TBST# during PCI to memory snoop cycles.
TEA#	—	O t/s 137	CPU bus transfer error acknowledge. Assertion of TEA# causes a machine check exception in the CPU. Assertion of TEA# terminates the current data tenure. The 664 asserts TEA# in the event of a catastrophic or unrecoverable system error. TEA# can be masked by setting the mask TEA# bit in the bridge control registers. If XATS# is asserted for a PIO cycle, the 664 asserts TEA# regardless of the condition of MASK_TEA#.
TS#	—	I/O 143	CPU bus transfer start. TS# is asserted low for one CPU bus clock to signal a valid address on the CPU_ADDR lines to start a transaction. TS# is an input to the 664 when a CPU bus master initiates a CPU bus transaction. TS# is an output of the 664 when it initiates a snoop cycle on behalf of a PCI bus master accessing system memory.

Table 2-1. 660 Bridge Signal Descriptions (Continued)

Signal Name	663	664	Description
CPU Bus Interface			
TSIZ[0:2]	—	I/O 145 146 147	CPU bus transfer size—number of bytes. The TSIZ lines are valid with the CPU_ADDR lines. The 660 ignores TSIZ[0:2] when TBST# is asserted for 32-byte bursts.
TT[0:4]	—	I/O See App C	Transfer type. Indicates the type of transaction currently in progress. The TT lines are valid with the CPU_ADDR lines.
XATS#	—	I 129	Extended address transfer start. When asserted, this signal indicates that the 60X CPU is performing I/O controller interface (PIO) operations. PIO operations are not supported by the 660 Bridge. If XATS# is asserted, the 664 generates a TEA# error to the 60X CPU (regardless of the setting of MASK_TEA#).
PCI Bus Interface			
PCI_AD[31:0]	I/O See App. C	I/O	PCI address/data bus. Address and data are multiplexed on the same pins. A PCI bus transaction consists of an address tenure followed by one or more data tenures. The address tenure is defined as one PCI bus clock in duration and is coincident with the clock in which PCI_FRAME# is first asserted. After the first clock, the PCI_AD pins carry data. The PCI_AD lines are driven by the initiation during the address phase, and by the originator of the data during the data phases.
PCI_C/BE[3:0]#	—	I/O t/s 3 4 5 6	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 active low byte enables—one bit for each of the four bytes on the PCI bus. During a PCI data tenure, PCI_C/BE[3]# applies to PCI_AD[31:24]—data byte three. If no bus transaction is in progress, then the current PCI bus master must drive the PCI_C/BE[3:0]# pins. C/BE[3:0]# are always driven by the PCI bus master.
PCI_CLK	—	I 123	PCI bus clock. The PCI bus clock is required to be within PCI bus specifications. The frequency of the CPU bus clock must be either 1x or 2x the PCI bus clock frequency. The 660 Bridge determines the CPU to PCI bus frequency ratio following the rising edge of RESET#.
PCI_DEVSEL#	—	I/O s/t/s 204	PCI device select. PCI_DEVSEL# is driven by a PCI target that has decoded an address and control bit encoding and claims the transaction. When another PCI bus master initiates a PCI memory transaction, the 664 drives PCI_DEVSEL# as the target of the transaction whenever the address is from 0 to 2G and within the range of physical memory (the installed DRAM space).
PCI_FRAME#	—	I/O s/t/s 200	PCI Frame. The current PCI bus master drives PCI_FRAME#. PCI_FRAME# signals the beginning of the address tenure on the PCI bus and the duration of the data tenure. PCI_FRAME# is deasserted to signal the final data phase of the transaction.
PCI_GNT#	—	I 54	PCI bus grant. The PCI bus arbiter asserts PCI_GNT# in response to the 664 asserting PCI_REQ# to request the PCI bus.

Table 2-1. 660 Bridge Signal Descriptions (Continued)

Signal Name	663	664	Description
PCI Bus Interface			
PCI_IRDY#	I 167	I/O s/t/s 201	PCI initiator ready. PCI_IRDY# is driven by the current PCI bus master. Assertion of PCI_IRDY# indicates that the PCI initiator is ready to complete this data phase.
PCI_LOCK#	—	I 53	PCI lock. This signal is used to allow PCI masters to establish a resource lock of one cache line of system memory. The 660 Bridge is never a locking master, but it enforces PCI locks of system memory.
PCI_PAR	—	I/O t/s 7	PCI parity. Even parity across PCI_AD[31:0] and the PCI_C/BE[3:0]# lines. PCI_PAR is valid one PCI bus clock after either an address or data tenure. The PCI device that drove the PCI_AD lines is responsible for driving the PCI_PAR line on the next PCI bus clock. The 664 checks and drives PCI parity.
PCI_PERR#	—	I/O s/t/s 10	PCI parity error. PCI_PERR# is asserted by the PCI device receiving the data. PCI_PERR# is sampled on the second PCI clock after the PCI_AD lines are sampled.
PCI_REQ#	—	O 58	PCI bus request. The 664 asserts PCI_REQ# to the PCI bus arbiter to request the PCI bus for a transaction initiated by a CPU bus master.
PCI_SERR#	—	O s/o/ d 71	PCI system error. PCI_SERR# is asserted for one PCI clock when a catastrophic failure is detected when the 660 Bridge is a PCI target. PCI_SERR# is not monitored by the 660 Bridge. The 660 asserts PCI_SERR# for certain PCI bus errors. Note: The signal is not s/t/s.
PCI_STOP#	—	I/O s/t/s 203	PCI stop. The target of the current PCI transaction can assert PCI_STOP# to indicate that the PCI target wants to end the current transaction. Data transfer can still take place if the target also asserts PCI_TRDY#, but that is the final data tenure.
PCI_TRDY#	I 168	I/O s/t/s 202	PCI target ready. The target of the current PCI transaction drives PCI_TRDY# to indicate that the PCI target is ready. Data transfer occurs when both PCI_TRDY# and PCI_IRDY# are asserted.
DRAM Interface			
CAS[7:0]#	—	O	Column address selects. CAS[0]# selects memory byte lane 0 (MEM_DATA[7:0]).
MA[11:0]	—	O	Memory address. MA[11] is the most significant bit.
MEM_CHECK[7:0]	I/O See App C	— 	Memory ECC/parity bus. MEM_CHECK[0] is always the memory check bit for memory byte lane 0 (MEM_DATA[7:0]). ECC or odd parity is generated and written on memory write cycles. ECC or even parity across eight bytes is checked on memory read cycles when enabled.

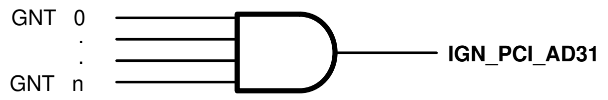
Table 2-1. 660 Bridge Signal Descriptions (Continued)

<table>
<tr><th>Signal Name</th><th>663</th><th>664</th><th>Description</th></tr>
<tr><td colspan="4">DRAM Interface</td></tr>
<tr><td rowspan="2">MEM_DATA[63:0]</td><td>I/O</td><td>—</td><td rowspan="2">Memory data bus. MEM_DATA[63:56] is always called memory byte lane 0 and is accessed using CAS[7]#. MEM_DATA[7,15,...63] are always the most significant bit in their memory byte lane.
In BE mode, MEM_DATA[63:56] is steered to/from CPU_DATA[56:63]. In LE mode, MEM_DATA[63:56] is steered to/from CPU_DATA[0:7].
<table>
<tr><th>MEM DATA</th><th>Most Signif. Bit</th><th>Mem Byte Lane</th><th>CAS#</th><th>Corresponding CPU Data [ ] in: BE mode</th><th>LE mode</th></tr>
<tr><td>63:56</td><td>63</td><td>7</td><td>7</td><td>56:63</td><td>0:7</td></tr>
<tr><td>55:48</td><td>55</td><td>6</td><td>6</td><td>48:55</td><td>8:15</td></tr>
<tr><td>47:40</td><td>47</td><td>5</td><td>5</td><td>40:47</td><td>16:23</td></tr>
<tr><td>39:32</td><td>39</td><td>4</td><td>4</td><td>32:39</td><td>24:31</td></tr>
<tr><td>31:24</td><td>31</td><td>3</td><td>3</td><td>24:31</td><td>32:39</td></tr>
<tr><td>23:16</td><td>23</td><td>2</td><td>2</td><td>16:23</td><td>40:47</td></tr>
<tr><td>15:8</td><td>15</td><td>1</td><td>1</td><td>8:15</td><td>48:55</td></tr>
<tr><td>7:0</td><td>7</td><td>0</td><td>0</td><td>0:7</td><td>56:63</td></tr>
</table></td></tr>
<tr><td colspan="2">See App C</td></tr>
<tr><td>RAS[7:0]#</td><td>—</td><td>O
See App C</td><td>Row address selects.</td></tr>
<tr><td>WE[1:0]#</td><td>—</td><td>O
175
176</td><td>DRAM write enables. WE[1]# and WE[0]# are the same and are used to avoid the need for external buffers.</td></tr>
<tr><td colspan="4">SRAM Interface</td></tr>
<tr><td>SRAM_ADS#/
ADDR0</td><td>—</td><td>O
124</td><td>SRAM address strobe/address 0. Enables latching of new address for the SRAMs when burst SRAMs are used. Least significant address bit when asynchronous SRAMs are used.</td></tr>
<tr><td>SRAM_ALE</td><td>—</td><td>O
119</td><td>SRAM address latch enable. Enables latching of address for the SRAMs to support address pipelining when asynchronous SRAMs are used. This signal is always high when burst SRAMs are used.</td></tr>
<tr><td>SRAM_CNT_EN#/
ADDR1</td><td>—</td><td>O
125</td><td>SRAM count enable/address 1. Enables incrementing of burst address if burst SRAMs are used. Next to least significant address bit when asynchronous SRAMs are used.</td></tr>
<tr><td>SRAM_OE#</td><td>—</td><td>O
117</td><td>SRAM output enable.</td></tr>
<tr><td>SRAM_WE#</td><td>—</td><td>O
118</td><td>SRAM write enable.</td></tr>
<tr><td colspan="4">TagRAM Interface</td></tr>
<tr><td>TAG_CLR#</td><td>—</td><td>O
116</td><td>Tag RAM clear. When asserted, all tags are forced to the invalid state. See the L2 Invalidate BCR.</td></tr>
<tr><td>TAG_MATCH</td><td>—</td><td>I
142</td><td>Tag RAM match indication (cache hit). This signal is asserted for a cache hit. It is usually an active high, open drain output of the tag RAM(s).</td></tr>
</table>

Table 2-1. 660 Bridge Signal Descriptions (Continued)

Signal Name	663	664	Description
TagRAM Interface			
TAG_VALID	—	O 115	Tag valid bit. The 660 asserts TAG_VALID to mark the current block valid in the tag. It is negated during tag writes in response to PCI write hits, etc. Connect to the "valid" input of the tags.
TAG_WE#	—	O 114	Tag RAM write enable. Asserted to write to the Tag RAM.
Miscellaneous			
IGN_PCI_AD31	—	I 57	Ignore PCI_AD[31]. This signal is asserted when the Intel™ SIO is the PCI master. IGN_PCI_AD31 is used to allow ISA bus masters to access system memory when the SIO is used as the ISA bridge. The 664 expects the memory address to appear in the range of 0 to 16M (it actually works over the entire 0-2G range) during the address phase when IGN_PCI_AD31 is asserted and then maps the access to system memory at 0 to 16M. It is usually generated by ANDing all of the active PCI bus grants (see note 1). IGN_PCI_AD31 must be valid on the PCI clock before FRAME# is sampled active.
INT_REQ	—	I 55	Interrupt request. This signal from the interrupt controller is synchronized with the CPU bus clock and passed through to the CPU as an interrupt on INT_CPU#.
MIO_TEST	I 156	I 154	Chip level test. Deassert low for normal operation. Do not casually assert this signal.
NMI_REQ	—	I 56	Non–maskable interrupt request. When detected active (normally from the ISA bridge), an error is reported to the CPU.
RESET#	—	I 156	Power–On–Reset. When this pin is low, all latches in the 664 enter a pre–defined state. Clocking mode is re-sampled, and ROM write lockout is cleared.
ROM_OE#	—	O 47	ROM output enable. ROM_OE# enables direct-attached ROM. This signal is always high during remote ROM operation.
ROM_WE#	—	O 60	ROM write enable. Write enable for flash ROM for direct-attach ROM. This signal is always high during remote ROM operation.
STOP_CLK_EN#	—	I 151	Prepares the 660 for stopping the CPU_CLK during power management. Tie this pin inactive (high).
TEST#	I 155	I 155	Test mode. Pull to logic high during normal operation. Do not casually assert this signal.

Note 1



2

Table 2-1. 660 Bridge Signal Descriptions (Continued)

Signal Name	663	664	Description
Miscellaneous — Signals Supported for Intra-Chipset Use Only			
AOS_RR_MMRS (see Section 8.3)	I 166	O 69	All Ones Select/ROM Remote/Mask MEM_RD_SMPL. This signal is used to force the data bus to 64 one-bits when the CPU reads memory or PCI space that is unoccupied. To force all ones, this signal must be asserted on the CPU_CLK that the CPU read latch samples data. While ROM_LOAD is asserted, this signal is used to determine the location of the ROM. Since ROM_REMOTE is deasserted, it indicates that the ROM is locally on the PCI bus. In this case, ROM data always arrives on PCI_AD[31:24]. While ROM_REMOTE is asserted, the ROM is assumed to be on a tertiary bus (such as the ISA bus). In this case, ROM data arrives like all other one-byte PCI targets—first byte on PCI_AD[7:0], second byte on PCI_AD[15:8], etc. When the PCI is burst reading memory, MASK_MEM_RD_SMPL is asserted after the first MEM_RD_SMPL. It then stays asserted until the PCI–to–MEM read latch is empty. Note: On PCI burst reads of memory, the PCI–to–MEM read latch keeps getting refilled as long as data from the memory is available before the PCI uses it up.
C2P_WRL_OPEN	I 154	O 61	CPU–to–PCI Write Latch Open. When asserted, the CPU–to–PCI write latch accepts new data on each CPU_CLK. When deasserted, the CPU–to–PCI write latch holds its current contents. This signal is asserted when data is to be sampled from the CPU. It is held deasserted until the data is written to the PCI.
CPU_DATA_OE#	I 146	O 197	CPU Data Output Enable. When asserted, the 663 drives the CPU_DATA bus on the next CPU_CLK. The 663 automatically drives the OE for CPU_DATA with the same value as it had on the previous CPU_CLK when an ECC correction occurs.
CPU_PAR_ERR#	O 174	I 192	CPU Data Bus Parity Error. When asserted, this signal indicates a parity error on the CPU data bus during a write cycle. This signal is only valid one CPU_CLK after the CPU data bus is valid (one clock delay).
CPU_RDL_OPEN	I 148	O 50	CPU Read Latch Open. When asserted, the CPU read latch accepts new data on each CPU_CLK. When deasserted, the CPU read latch holds its current contents. This signal is asserted when data is to be sampled from memory or the PCI. When sampling data from memory, this signal is also active on the following CPU_CLK to allow ECC corrections to occur if necessary. If no ECC corrections occur, the same data is provided by the MEM read ECC correction logic. See Section 2.2, CPU_RDL_OPEN Resistor.

Table 2-1. 660 Bridge Signal Descriptions (Continued)

Signal Name	663	664	Description
Miscellaneous — Signals Supported for Intra-Chipset Use Only			
CRS_C2PWXS	I 151	O 65	CPU Read Select/CPU-to-PCI Write Crossover Select. When the CPU read latch is sampling data, this signal controls the CPU read multiplexer. When asserted, the memory data bus is routed to the CPU read latch. When deasserted, PCI data is routed to the CPU read latch. This signal must be valid on the CPU_CLK that the CPU read latch samples data. When the CPU-to-PCI write latch is sampling data, this signal controls the CPU-to-PCI write crossover. When asserted, the most-significant 32 bits are driven to the CPU-to-PCI write latch. When deasserted, the least-significant 32 bits are driven to the CPU-to-PCI write latch. This signal must be valid on the CPU_CLK that the CPU-to-PCI write latch samples data.
DUAL_CTRL_REF	I 170	O 205	Control Signal Multiplexer Select. For 663 inputs that have two functions (MEM_BEs and ECC_LE_SEL) this signal indicates which function is currently active. This signal is generated by dividing CPU_CLK by two.
ECC_LE_SEL	I 149	O 2	ECC Select/Little-Endian Select. This signal indicates use of ECC or byte parity when DUAL_CTRL_REF is high and indicates use of little-endian or big-endian mode when DUAL_CTRL_REF is low. Asserting ECC_LE_SEL enables the generation and checking of ECC to memory. Deassertion enables generation and checking of byte parity. Asserting ECC_LE_SEL indicates little-endian mode. Deassertion indicates big–endian mode.
MEM_BE[3:0]	I 164 163 162 161	O 1 208 207 206	Memory Byte Enables. The eight BEs for read-modify-write memory cycles are multiplexed on MEM_BE[3:0]. The first four BEs are asserted on MEM_BE[3:0] when DUAL_CTRL_REF is high, and the second four are asserted on MEM_BE[3:0] when DUAL_CTRL_REF is low. When combined, they form MEM_RMW_BE[7:0]#. An active byte enable indicates valid write data from the CPU or PCI. The MEM_RMW_BE[7:0]# signals are sampled on the same clock that read-data is sampled by the MEM write latch; therefore, MEM_BE[3:0] must be valid on the first and second clock before read data is sampled. When not running a read-mod-write cycle MEM_BE[3:0] should indicate all data lanes are enabled—MEM_RMW_BE[7:0]# all asserted low.
MEM_DATA_OE#	I 145	O 196	Memory Data Output Enable. When asserted, the 663 drives the MEM_DATA bus on the next CPU_CLK.
MEM_ERR#	O 171	I 194	Memory Error. When asserted, indicates an uncorrectable multi-bit or parity error has occurred during a memory read. This signal is only valid on the third CPU_CLK after MEM_RD_SMPL is asserted.
MEM_RD_SMPL	I 147	O 49	Memory Read Sample. This signal is asserted on the CPU_CLK that data from the memory is sampled during a CPU or PCI read. MEM_RD_SMPL is used by the ECC logic to determine when to sample ECC results. This signal is also used by the PCI read extension latch and the PCI-to-MEM read latch to load new data. See PCI_OL_OPEN for more details.

2

Table 2-1. 660 Bridge Signal Descriptions (Continued)

Signal Name	663	664	Description
Miscellaneous — Signals Supported for Intra-Chipset Use Only			
MEM_WRL_OPEN	I 150	O 51	Memory Write Latch Open. When asserted, the MEM write latch accepts new data on each CPU_CLK. When deasserted, the MEM write latch holds its current contents. This signal is asserted when data is to be sampled from the CPU or the PCI. It is held deasserted until the data is written to memory. When this signal is asserted on read-modify-write cycles, data is sampled from the CPU and memory or from the PCI and memory. This signal is also active on the following CPU_CLK to allow ECC corrections to occur if necessary. If no ECC corrections occur, the same data is provided by the MEM read ECC correction logic.
MWS_P2MRXS	I 152	O 66	Memory write select/PCI-to-memory read crossover select. When the memory write latch is sampling data, this signal controls the memory write multiplexer. When MWS_P2MRXS is asserted, the CPU data bus is routed to the memory write latch. When it is deasserted, the PCI data is routed to the memory write latch. This signal must be valid on the CPU_CLK that the memory write latch samples data. When the PCI-to-MEM read latch is sampling data, this signal controls the PCI-to-MEM read crossover. When the signal is asserted, the most-significant 32 bits are driven to the PCI-to-MEM read latch. When it is deasserted, the least-significant 32 bits are driven to the PCI-to-MEM read latch. This signal must be valid on the CPU_CLK that the PCI-to-MEM read latch samples data.
PCI_AD_OE#	I 144	O 195	PCI Data Output Enable. When asserted, the 663 drives the PCI_AD bus. Note: This is an asynchronous input to the buffer chip—it is not clocked.
PCI_EXT_SEL	I 153	O 67	PCI Read Extension Select/PCI Write Extension Select. When the PCI is reading from memory, this signal controls the PCI read extension multiplexer. When the signal is asserted, data from the PCI read extension latch is routed to the PCI-to-MEM read latch. When it is deasserted, data from the memory is routed to the PCI-to-MEM read latch. This signal must be valid when the PCI-to-MEM read latch samples data. When the PCI is writing to memory, this signal controls the PCI write extension multiplexer. When the signal is asserted, data from the PCI write extension latch is routed to the MEM write latch. When it is deasserted, PCI data is routed to the MEM write latch. This signal must be valid when the MEM write latch samples data.

2

Table 2-1. 660 Bridge Signal Descriptions (Continued)

Signal Name	663	664	Description
Miscellaneous — Signals Supported for Intra-Chipset Use Only			
PCI_OL_OPEN	I 165	O 64	PCI Other Latches Open. This signal controls the latch enables for the following three latches—the PCI-to-MEM read latch, the PCI read extension latch, and the PCI write extension latch. During PCI reads to memory, the PCI read extension latch and the PCI-to-MEM read latch accept new data when PCI_OL_OPEN is asserted (and when PCI_IRDY# is asserted) or when MEM_RD_SMPL is asserted (one CPU_CLK after MEM_RD_SMPL when ECC is enabled). During the period when enabling the latches is dependent on PCI_IRDY#, this signal must only be active on CPU_CLKs when PCI_CLK rises, to guarantee sampling PCI_IRDY# only on rising PCI_CLK. Essentially, PCI_OL_OPEN on PCI reads to memory if !PCI_TRDY# && !PCI_CLK. This allows the 663 to advance to the next data as soon as the current data is accepted by the PCI master. This signal is also asserted during PCI writes to memory when data is sampled from the PCI. It is held deasserted until the data is passed to the MEM write latch.
PCI_OUT_SEL	I 169	O 68	PCI Output Select. When asserted the memory data is routed to the PCI output bus. When deasserted, CPU data is routed to the PCI output bus. This must be valid the entire time that data is driven to the PCI bus. This signal is asynchronous—not referenced to clock.
ROM_LOAD	I 160	O 70	ROM Load. This signal is used to load data from a ROM one byte at a time until eight bytes are received, then pass the eight bytes to the CPU. The ROM data buffer receives the one-byte transfers and assembles them into the eight-byte result. When ROM_LOAD is deasserted for more than one CPU_CLK, the ROM data buffer is reset. When ROM_LOAD is then asserted, it receives data into its first byte. Each time ROM_LOAD is deasserted, the first byte is passed to the second, the second is passed to the third, etc. ROM_LOAD should not be deasserted for more than one CPU_CLK while assembling the eight bytes to prevent resetting the buffer. Also, the ROM data buffer is routed toward the CPU data read latch when ROM_LOAD is asserted. It is expected that the CPU data read latch will grab eight bytes of ROM Data on the eighth ROM_LOAD.
SBE#	O 175	I 193	Single-Bit Error. When asserted, indicates a correctable single-bit error has occurred on the memory data bus. This signal is valid only on the CPU_CLK following the assertion of MEM_RD_SMPL. If the memory is not in ECC mode, this signal is undefined.

2.2 CPU_RDL_OPEN Resistor

Add a 200Ω series resister to the CPU_RDL_OPEN net between the 664 and the 663.

During a CPU to memory read, if (at the 663) CPU_RDL_OPEN goes low before MEM_RD_SMPL goes low, then the 663 may provide incorrect data to the CPU. The Table shows the minimum required interval between the falling edge of MEM_RD_SMPL and the falling edge of CPU_RDL_OPEN.

Table 2-2. Minimum Required Interval Between Low MEM_RDL_OPEN and Low CPU_RDL_OPEN

Case	Conditions	663 Requires	664 Provides
1	Worst Case Process, Temperature, & V_{DD}	> 1.8ns	1.3ns
2	Best Case Process, Worst Case Temp & V_{DD}	> 0.2ns	0.5ns
3	Best Case Process, Temperature, & V_{DD}	> 0.1ns	0.3ns

The worst practical case occurs while the 664 is at Case 2 (provides .5ns difference) and the 663 is at Case 1 (requires 1.8ns difference). This requires that a minimum delay of 1.3ns be added to the CPU_RDL_OPEN signal. A delay of 2.4ns is recommended to allow a conservative margin of error. (Delay = RC = 200Ω * 12pf = 2.4ns). Note that this assumes the CPU_RDL_OPEN and MEM_RD_SMPL nets are both about three inches long and that the resister is close to the 664. A different resister value or an R-C combination may be required if the length or capacitance of the two nets are significantly different, or if the resister placement differs significantly.

Section 3
CPU Bus

The CPU bus connects the CPU(s) with the 660 Bridge, the L2 cache SRAM and tagRAM, and possibly other CPU bus agents. All access to the rest of the system is provided to the CPU by the 660 Bridge. The 660 supports CPU bus frequencies up to 66MHz.

3.1 Transfer Type Decoding

Table 3-1 shows the 660 decoding of CPU bus transfer types. Based on TT[0:3], the 660 responds to CPU bus master cycles by generating a read transaction, a write transaction, or an address-only response. (The 660 ignores TT[4] when it evaluates the transfer type.)

The bridge decodes the target of the transaction based on the address range of the transfer as shown in Table 3-2. The transfer type decoding shown in Table 3-1 combines with the target decoding to produce the following:

- System memory reads and writes
- PCI I/O reads and writes
- PCI configuration reads and writes
- PCI interrupt acknowledge reads
- PCI memory reads and writes
- System ROM reads and writes
- Various bridge control register (BCR) reads and writes.

Within Table 3-1, SBR means single-beat read, and SBW means single-beat write.

Transfer types in Table 3-1 that have the same response are handled identically by the bridge. For example, if the address is the same, the bridge generates the same memory read transaction for transfer types 0101, 0111, 1101, and 1111.

References in the remainder of this document to a CPU read, assume one of the transfer types that produce the read response from the 660. Likewise, references to a CPU write refer to those transfer types that produce the write response.

The 660 does not generate PCI or system memory transactions in response to address only transfers. The bridge does drive all-ones onto the CPU bus and signals TA# during an eciwx if no other CPU bus agent claims the transfer.

In general, the 660 supports PowerPC 740 and 750 as members of the 603 family and PID9q-, PID9v-, PID9t-, and PID10q-604e as members of the 604e family.

Table 3-1. TT[0:3] (Transfer Type) Decoding by 660 Bridge

TT[0:3]	CPU Operation	CPU Bus Transaction	660 Operation For CPU to Memory Transfers (CPU Bus Addr 0 to 2G)	660 Operation For CPU to PCI Transactions (CPU Bus Addr over 2G)
0000	Clean block or lwarx	Address only	Asserts AACK#. No other response. No PCI transaction.	
0001	Write with flush	SBW or burst	Memory write operation.	PCI write transaction.
0010	Flush block or stwcx	Address only	Asserts AACK#. No other response. No PCI transaction.	
0011	Write with kill	SBW or burst	Memory write operation. L2 invalidates addressed block.	PCI write transaction.
0100	sync or tlbsync	Address only	Asserts AACK#. No other response. No PCI transaction.	
0101	Read or read with no intent to cache	SBR or burst	Memory read operation.	PCI read transaction.
0110	Kill block or icbi	Address only	Asserts AACK#. L2 invalidates addressed block.	Asserts AACK#. No other response.
0111	Read with intent to modify	Burst	Memory read operation.	PCI read transaction.
1000	eieio	Address only	Asserts AACK#. No other response. No PCI transaction.	
1001	Write with flush atomic, stwcx	SBW	Memory write operation.	PCI write transaction.
1010	ecowx	SBW	Asserts AACK# and TA# if the transaction is not claimed by another 60X bus device. No PCI transaction. No other response.	
1011	Reserved		Asserts AACK#. No other response. No PCI transaction.	
1100	TLB invalidate	Address only	Asserts AACK#. No other response. No PCI transaction.	
1101	Read atomic, lwarx	SBR or burst	Memory read operation.	PCI read transaction.
1110	External control in, eciwx	Address only	660 asserts all ones on the CPU data bus. Asserts AACK#, and TA# if the transaction is not claimed by another 60X bus device. No PCI transaction. No other response.	
1111	Read with intent to modify atomic, stwcx	Burst	Memory read operation.	PCI read transaction.

3.2 CPU Bus Address Mapping

The bridge decodes the target of the transaction based on the address range of the transfer as shown in Table 3-2.

Table 3-2. 660 Bridge Address Mapping of CPU Bus Transactions

CPU Bus Address	Other Conditions	Target Transaction	Target Bus Address	Notes
0 to 2G 0000 0000h to 7FFF FFFFh		System Memory	0 to 2G (DRAM) 0000 0000h to 7FFF FFFFh	(1)(2)
2G to 2G + 8M 8000 0000h to 807F FFFFh	Contiguous Mode	PCI I/O Transaction, BCR Transaction, or PCI Configuration (Type 1) Transaction	0 to 8M (PCI I/O) 0000 0000h to 007F FFFFh	(3)
	Non-Contigu-ous Mode		0 to 64K (PCI I/O) 0000 0000h to 0000 FFFFh	(4)
2G + 8M to 2G + 16M 8080 0000h to 80FF FFFFh		PCI Configuration (Type 0) Transaction	PCI Configuration Space 0080 0000h to 00FF FFFFh	
2G + 16M to 3G – 8M 8100 0000h to BF7F FFFFh		PCI I/O Transaction	16M to 1G – 8M (PCI I/O) 0100 0000h to 3F7F FFFFh	
3G – 8M to 3G BF80 0000h to BFFF FFFFh		BCR Transactions and PCI Interrupt Ack. Transactions	1G – 8M to 1G 3F80 0000h – 3FFF FFFFh	(3)(6)
3G to 4G – 2M C000 0000h to FF7F FFFFh		PCI Memory Transaction	0 to 1G – 2M (PCI Mem) 0000 0000h to 3FDF FFFFh	
4G – 2M to 4G FFE0 0000h to FFFF FFFFh	Direct Attach ROM Read, Write, or Write Lockout	BCR Transaction	0 to 2M (PCI ROM) 0000 0000h to 001F FFFFh (ROM Address Space)	(2)
	Remote ROM	PCI Memory Transac-tion to I/O Bus Bridge	4G – 2M to 4G (ISA ROM) FFE0 0000h to FFFF FFFFh	(2)

Notes for Table 3-2:
1) System memory can be cached. Addresses from 2G to 4G are not cacheable.
2) Memory does not occupy the entire address space.
3) Registers do not occupy the entire address space.
4) Each 4K page in this 8M CPU bus address range maps to 32 bytes in PCI I/O space.
5) Registers and memory do not occupy the entire address space. Accesses to unoccupied addresses result in all one-bits on reads and no-ops on writes.
6) A memory read of BFFF FFF0h generates an interrupt acknowledge transaction on the PCI bus.

3

3.3 CPU to Memory Transfers

The system memory address space is from 0 to 2G. Physical memory does not occupy the entire address space. When the CPU reads an unpopulated location, the Bridge returns all-ones and completes the transfer normally. When the CPU writes to an unpopulated location, the Bridge signals normal transfer completion to the CPU but does not write the data to memory. The memory select error bit in the error status 1 register (bit 5 in index C1h) is set in both cases.

All CPU to memory writes are posted and can be pipelined. The 660 supports all CPU to memory bursts, and all single-beat transfer sizes and alignments that do not cross an 8-byte boundary, which includes all memory transfers initiated by the 604 CPU.

The bridge supports all transfer sizes and alignments that the CPU can create in LE mode; however, all loads or stores must be at natural alignments in LE mode (or the PowerPC 604 will take an alignment exception). Also, load/store multiple word and load/store string word instructions may not be supported in the CPU in LE mode.

3.4 CPU to PCI Transactions

CPU accesses to addresses within the 2G to 4G range produce the various PCI transactions. When the 660 decodes a CPU access as targeted for the PCI, the 660 requests the PCI bus. Once the PCI bus arbiter grants the PCI bus to the 660, the bridge initiates the PCI cycle and (in general) releases the CPU bus. All addresses from 2G to 4G (including ROM space) must be marked non-cacheable. See the *PowerPC Reference Platform Specification*.

The 660 supports all CPU to PCI transfer sizes that do not cross a 4-byte boundary, and, to support the 604 store multiple instruction, the bridge supports 8-byte CPU to PCI writes that are aligned on an 8-byte boundary. The 660 does not support CPU bursts to the PCI bus.

In compliance with the PCI specification, the 660 master aborts all PCI I/O transactions that are not claimed by a PCI agent.

Since all CPU to PCI transactions are CPU memory mapped, software must, in general, utilize the EIEIO instruction which enforces in-order execution, particularly on PCI I/O and configuration transactions. Some PCI memory operations can also be sensitive to order of access also.

The first beat of CPU to PCI transactions can complete as soon as the clock after frame is deasserted (see Figure 3-1).

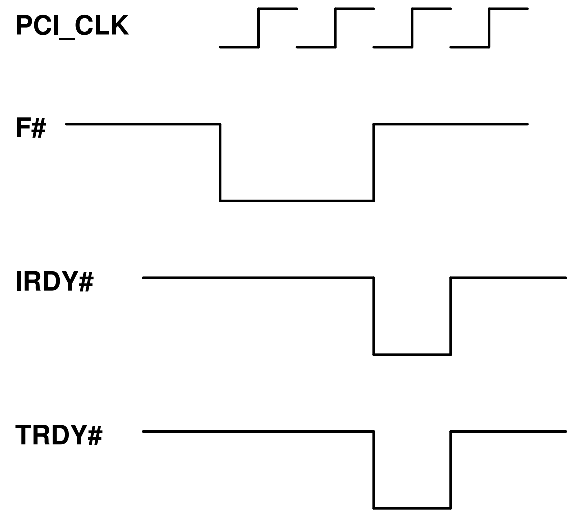


Figure 3-1. CPU to PCI Transactions

3.4.1 CPU to PCI Read

If the CPU to PCI cycle is a read, a PCI read cycle is run. If the PCI read cycle completes, the data is passed to the CPU and the CPU cycle is ended. If the PCI cycle is retried, the CPU cycle is retried. If a PCI master access to system memory is detected before the PCI read cycle is run then the CPU cycle is retried (and no PCI cycle is generated).

Minimum read time from the PCI is two CPU_CLKs plus three PCI_CLKs plus one CPU_CLK. If the PCI is run at 33MHz and the CPU bus is at 66MHz, the minimum PCI read cycle is nine CPU clocks. This assumes that the clock period that TS# goes active is the first clock counted and the clock period that TA# goes active is the last clock counted. The maximum throughput is four bytes of data every 10 CPU clocks (every five PCI clocks).

3.4.2 CPU to PCI Write

If the CPU to PCI cycle is a write, a PCI write cycle is run. CPU to PCI (one byte to four byte) I/O writes are not posted, as per the *PCI Local Bus Specification* version 2.1. If the PCI transaction is retried, the Bridge retries the CPU.

Four byte CPU to PCI memory writes are posted, so the CPU write cycle is ended as soon as the data is latched. Eight byte memory writes are posted as soon as the first PCI data phase completes. If the PCI cycle is retried, the Bridge retries the cycle (on the PCI bus) until it completes. Another PCI busmaster can initiate a PCI transaction between bridge retries.

Minimum write time to the PCI is six CPU clocks. When the PCI write cycle occurs, it takes a minimum of three PCI clocks. The maximum throughput is four bytes of data every eight CPU clocks (every four PCI clocks).

3.4.2.1 Eight-Byte Writes to the PCI (Memory and I/O)

The 660 supports 1-byte, 2-byte, 3-byte, and 4-byte transfers to and from the PCI. The 660 also supports 8-byte memory and I/O writes (writes only, not reads) to the PCI bus. This enables the use of the 604 store multiple instruction to PCI devices.

When an 8-byte write to the PCI is detected, it is not posted initially. Instead the CPU waits until the first 4-byte write occurs, then the second 4-byte write is posted. If the PCI retries on the first four byte transfer or a PCI master access to system memory is detected before the first 4-byte transfer then the CPU is retried. If the PCI retries on the second 4-byte transfer then the 660 retries the PCI write.

Minimum write time is eight bytes for every 12 CPU clocks (six PCI clocks).

Eight-byte transactions are not supported to PCI configuration space.

3.4.3 PCI Retry

CPU to PCI transactions that the PCI target retries, cause the 660 to deassert its PCI_REQ# (the Bridge follows the PCI retry protocol). The Bridge stays off of the PCI bus for two PCI clocks before reasserting PCI_REQ# (or FRAME#, if the PCI bus is idle and the PCI_GNT# to the Bridge is active). A PCI busmaster can initiate a PCI transaction between bridge retries.

3.4.4 PCI FRAME#

During CPU to PCI transfers, the 664 typically asserts FRAME# for 1 or 2 PCI clocks before asserting IRDY#; however, under certain conditions, the 664 may assert FRAME# for up to 8 PCI clocks. IRDY# is asserted in the correct relationship to FRAME#.

3.4.5 CPU to PCI Configuration

The 660 allows access to PCI configuration space via two different mechanisms.

CPU accesses to addresses from 2G + 8M to 2G + 16M or from 8000 0CFCh to 8000 0CFFh (the configuration data register) when the configuration address indicates a device number greater than zero generate a PCI type 0 configuration cycle. (See Section 10.2 and 10.3.)

Type 1 configuration transactions can also be initiated via the PCI/BCR configuration address and data BCRs.

3.4.5.1 PCI Type 0 Configuration Transaction (650 Compatible)

CPU accesses to the address range 2G+8M to 2G+16M (see Table 3-2) cause the 660 to arbitrate for the PCI bus and then to run a type 0 PCI configuration transaction as described in the PowerPC Reference Platform Specification and implemented by the IBM27–82650 PowerPC to PCI Bridge. Eight-byte PCI configuration transactions are not supported. Using this method with addresses other than the ones shown in Table 3-3 is not supported, and may lead to hardware damage due to having more than one IDSEL asserted at the same time.

Table 3-3. PCI Configuration Addresses

Bridge Control Register	CPU Bus Address	R/W	Bytes	Cycle and IDSEL
PCI Slot 0 Configuration Space	8080 08xx	R/W	4	PCI Config, IDSEL = PCI_AD[11]
PCI Slot 1 Configuration Space	8080 10xx	R/W	4	PCI Config, IDSEL = PCI_AD[12]
PCI Slot 2 Configuration Space	8080 20xx	R/W	4	PCI Config, IDSEL = PCI_AD[13]
PCI Slot 3 Configuration Space	8080 40xx	R/W	4	PCI Config, IDSEL = PCI_AD[14]
PCI Slot 4 Configuration Space	8080 80xx	R/W	4	PCI Config, IDSEL = PCI_AD[15]
PCI Slot 5 Configuration Space	8081 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[16]
PCI Slot 6 Configuration Space	8082 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[17]
PCI Slot 7 Configuration Space	8084 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[18]
PCI Slot 8 Configuration Space	8088 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[19]
PCI Slot 9 Configuration Space	8090 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[20]
PCI Slot 10 Configuration Space	80A0 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[21]
PCI Slot 11 Configuration Space	80C0 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[22]

When the PCI bus is acquired, the address is driven for one PCI clock before PCI_FRAME# is asserted. This allows the IDSELs to be resistively connected to the PCI_AD[31:0] bus at the system level. This method of accessing PCI configuration space does not allow access to the PCI configuration registers in the bridge chip and it should not be used unless required to maintain 650 compatibility.

3.4.5.2 PCI Type 0 Configuration Transaction (CFC/CF8)

PCI configuration space can be accessed via the CFC/CF8 register pair as described in section 10.3. Eight-byte PCI configuration transactions are not supported.

3.4.6 CPU to PCI Interrupt Acknowledge Transaction

When the CPU initiates a single-byte read of the interrupt acknowledge address (BFFF FFF0h), the bridge arbitrates for the PCI bus and then executes a standard PCI interrupt acknowledge transaction. The system interrupt controller claims the transaction and supplies the 1-byte interrupt vector. There is no physical interrupt vector BCR in the bridge. Other PCI bus masters can initiate interrupt acknowledge transactions. This operation is unaffected by the Endian mode of the system; in both little- and big-endian modes, BE[0]# is asserted.

3.5 CPU to ROM

The system ROM address space is from 4G – 2M to 4G. If the size of the installed ROM is less than 2M, it is mirrored throughout the ROM space. When the CPU writes to any ROM location while the ROM is locked out, the bridge signals normal transfer completion to the CPU but does not write the data to the ROM. The CPU bus write to the locked flash bit in the error status 2 register (bit 0 in index C5h) is set.

3.6 CPU to BCR Transfers

The bridge control registers (BCRs) do not occupy the entire address space assigned to them in the memory map. If the transaction is not claimed by the Bridge, other PCI agents can claim it. If the transaction is not claimed (including subtractively), the bridge master aborts the transaction.

Do not configure any PCI agents to have I/O locations that overlap the BCR locations.

3.7 CPU to ISA I/O

CPU accesses in the 2G to 2G + 8M range are mapped to the PCI bus as I/O transactions in the 0 to 8M range (contiguous I/O) or 0 to 64K range (non-contiguous I/O). These transactions are to be claimed by the ISA bus bridge and forwarded to the ISA bus as I/O cycles.

3.7.1 Contiguous I/O Mode Address Mapping

In contiguous I/O mode (CONTIG_IO asserted), CPU addresses from 2G to 2G + 8M generate a PCI I/O cycle on the PCI bus with PCI_AD[29:00] unchanged. The low 64K of PCI I/O addresses are forwarded to the ISA bus unless claimed by a PCI agent.

Memory page protection attributes can only be assigned by 4K groups of ports, rather than by 32-port groups as in the non-contiguous mode. This is the power-on default mode.

Figure 3-2 shows an example of the mapping from the CPU bus to the ISA I/O address in contiguous I/O mode.

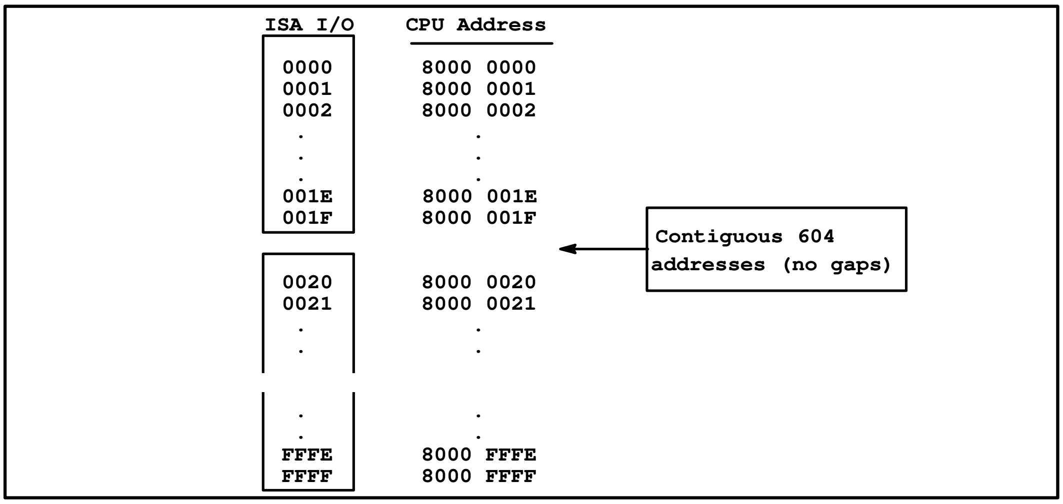


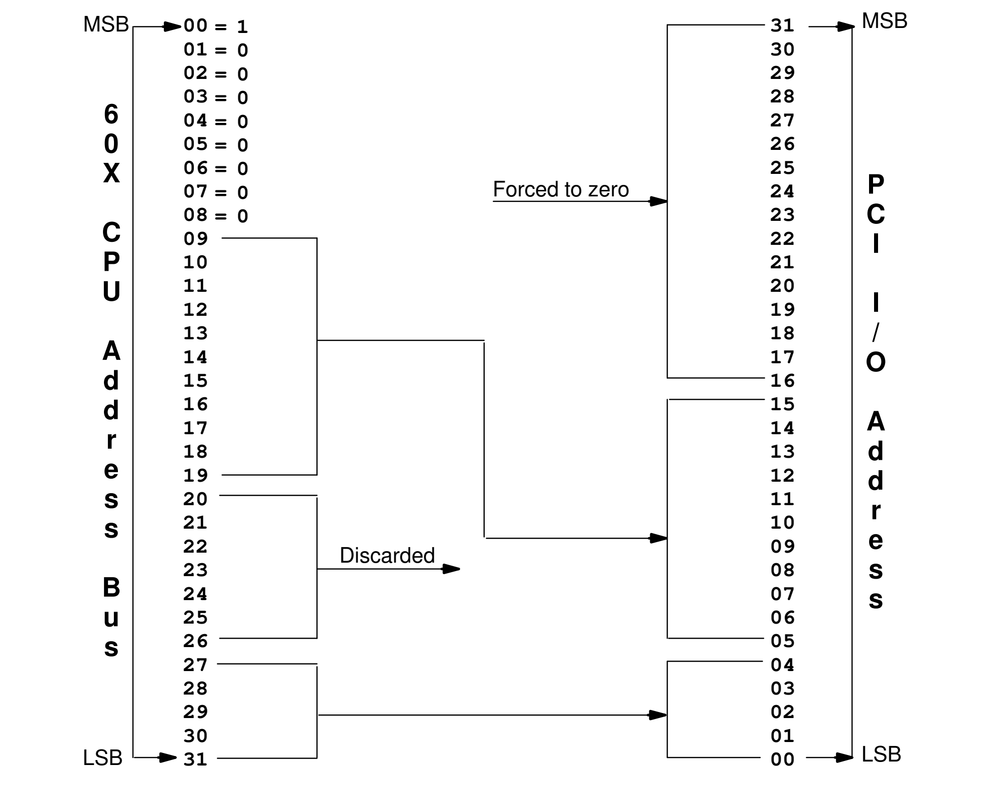
Figure 3-2. Contiguous PCI I/O Address Translation

3.7.2 Non-Contiguous I/O Mode Address Mapping

Figure 3-3 shows the address mapping that the 660 performs in non-contiguous mode. The I/O map type register (address 8000 0850h) and the bridge chip set options 1 register (index BAh) control the selection of contiguous and non-contiguous I/O. In non-contiguous mode, the 8M address space of the bus is compressed into 64K of PCI address space, and the CPU cannot create PCI I/O addresses from 64K to 8M.

If little-endian mode is selected, CPU_ADDR[29:31] are unmunged before they reach PCI_AD[2:0]. In little-endian mode, CPU_ADDR[29:31] are munged by the CPU before the address is driven to the CPU bus. The 660 unmunges the three low-order address bits as part of the process of handling little-endian memory formatting. See the *PowerPC 601 User's Manual*.

In non-contiguous I/O mode, the 660 partitions the address space so that each 4K page is remapped into a 32-byte section of the 0 to 64K ISA port address space, so that CPU protection attributes can be assigned to any of the 4K pages. This provides a flexible mechanism to lock the I/O address space from change by user-state code. This partitioning spreads the ISA I/O address locations over 8M of CPU address space.



The three least-significant address bits are unmunged during the transformation if little-endian mode is selected.

Figure 3-3. Non-Contiguous PCI I/O Address Transformation

In non-contiguous mode, the first 32 bytes of a 4K page are mapped to a 32-byte space in the PCI address space. The remainder of the addresses in the 4K page are mirrors into the the same 32-byte PCI space. Each of the 32 contiguous port addresses in each 4K page has the same protection attributes in the CPU.

For example, in Figure 3-4 CPU, addresses 8000 0000h to 8000 001Fh are converted to PCI I/O port 0000h through 001Fh. PCI I/O port 0020h starts in the next 4K page at CPU address 8000 1000h.

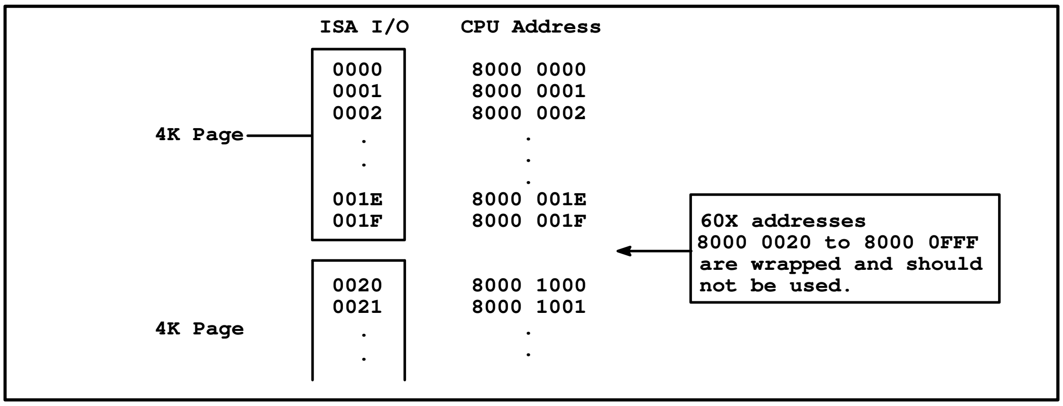


Figure 3-4. Non-Contiguous PCI I/O Address Translation

3.7.3 Final I/O Address Formation

The 660 maps CPU bus addresses from 2G to 4G as PCI transactions, error address register reads, or ROM reads and writes. The 660 manipulates CPU 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 little-endian mode is selected.
- After unmunging, PCI_AD[1:0] are set to 00b except for PCI I/O cycles.

3.8 CPU Bus Masters

Typical system configurations using the 660 have two masters on the CPU bus—the CPU and the 660. The 660 is the master on the CPU bus when it runs snoop cycles in response to PCI master access to system memory. The CPU is the master at all other times.

The 660 can support a second external master on the CPU bus, which is usually an external L2 cache or a second CPU. To support a second external master, the 660 includes the following:

- A second set of bus request and bus grant signals.
- Sampling of ARTRY# on the third CPU_CLK (fourth clock for the 603 in 1:1 mode) following the falling edge of TS# for all accesses to system memory. This allows cache coherency protocols to function properly. Note that the L2 cache controller may already be asserting TA# on this cycle.
- The First_Read# bit, BCR(8000 081C) bit 0, which can be read by the CPUs at Power-on to resolve who is CPU0.

Each bus master must provide both of the following functions:

- Support at least one level of address pipelining, so that address tenure(n) can end and address tenure(n+1) can begin before data tenure n ends. Note that this requirement must be met even if there is only a single bus master. The CPUs meet this requirement.
- Support the data bus busy protocol, driving DBB# while it is the data bus master, and monitoring DBB# to correctly determine the end of a data tenure mastered by another CPU bus agent. The CPUs meet this requirement except in no–DRTRY# mode; therefore, when a second CPU bus master is installed, any CPUs on the bus must have DRTRY# enabled.

This is the extent to which the 660 supports multiple processors. This may be sufficient for many special purpose multi-processor systems, but general-purpose symmetric-multi-processor systems require additional support such as multi-interrupt controllers and error handlers (see the *Open Programmable Interrupt Controller (PIC) Register Interface Specification*, Revision 1.2).

Do not use no-DRTRY# mode with PID6-603e with CPU bus speeds greater than 40 MHz.

3.8.1 External L2 as a CPU Bus Master

An external L2 cache that is look-aside with write-back capability can act as the second CPU bus master. To attach an external L2 controlled by the L2 invalidate register (8000 0814h), the system control register (8000 081Ch), and the I/O map-type register (8000 0850h), use the following setup:

4. Disable the internal L2 by setting bit 1 of the cache status register (index B1) to zero.
5. Enable external register support mode by setting bit 5 of the bridge chip set options 3 register (BCR(D4) bit 5) to one.

3.9 CPU Bus Targets

The 660 supports target devices attached to the CPU bus. By default, the bridge acts as the target for all CPU bus cycles; however, a mechanism has been provided for other CPU bus agents to claim a cycle and respond as the target of that cycle.

CPU bus targets can only claim memory cycles initiated by a CPU bus master other than the 660. For example, they cannot claim snoop cycles or PIO cycles.

CPU bus targets can only claim cycles from the 0 to 2G system memory address space. These cycles must not be cached by the internal L2 cache controller. But the internal L2 caches all memory space which is occupied by DRAM. For example, if the L2 is enabled and 1G of memory is installed, then only the space from 1G to 2G can be claimed. However, if the internal L2 is disabled, the entire address space from 0 to 2G can be claimed regardless of the installed memory.

A CPU bus target claims a cycle by asserting CPU_BUS_CLAIM# on the CPU_CLK after it samples TS# active. The 660 samples this signal two clocks after it samples TS# active.

If a CPU bus target claims a cycle, it is responsible for terminating both the address and data tenures associated with the cycle, including driving AACK#, TA#, and TEA#. The claiming CPU bus agent must drive AACK# even if the bus operation is retried (ARTRY# asserted). CPU bus targets may not assert DRTRY#. The CPU bus target is also responsible for detecting and reporting any errors that occur during the claimed cycle.

CPU bus targets must not drive AACK#, TA#, TEA#, or CPU_DATA until the listed number of clocks following the assertion of TS#:

1. First clock, if the internal L2 is disabled (second clock for 603 in 1:1 mode).
2. Third clock, if the internal L2 is enabled (fourth clock for 603 in 1:1 mode). This is necessary because the internal L2 drives these signals until it determines that the cycle is not a cache hit.
3. In compliance with the CPU bus spec, TA#, TEA#, and CPU_DATA also must not be driven until one clock after the previous data bus tenure has ended. These signals must not be driven until the data bus tenure for the claimed cycle has begun.

The CPU bus target must restore and tristate AACK#, TA#, and TEA# following their assertion at the end of the associated bus tenure.

The CPU bus target must have some means (e.g., DBG#) for determining when it may begin a data tenure.

The CPU bus target must not assert AACK# until the previous data bus tenure has ended. This is necessary to meet the 660 requirement of one level of pipelining.

3.10 CPU Bus Parity

The 660 can generate parity and check for parity errors on the CPU data bus (CPU_DATA[0:63]). Whenever a CPU bus master performs a write cycle, the 660 checks the parity. Whenever a CPU bus master performs a read cycle, the 660 drives the data parity bits. If desired, the data SRAMs for the L2 cache can store and read the parity data to provide an extra measure of error checking.

The 660 does not generate or check CPU data bus parity on cycles claimed by other CPU bus targets (see Section 8, Exceptions).

3.11 CPU Bus Arbitration

The 660 arbiter coordinates the activities of the three CPU bus agents, CPU1, CPU2, and the snoop engine. The snoop engine of the bridge is a conceptual set of logic inside the Bridge which broadcasts snoop cycles to the CPU bus in response to PCI to memory transactions (see section 4.5).

Each CPU bus agent has an address bus request line and an address bus grant line. CPU1 is the default for the CPU. CPU may be an additional CPU, an external (to the Bridge) L2, or other well behaved CPU bus agent. The request and grant lines of the snoop engine are internal to the Bridge.

The 660 supports pipelined split-bus transactions, which feature an address tenure and a subsequent data tenure (see the *PowerPC 604 User's Manual*, especially section 8.2). The initial business of the Bridge arbiter is with the address tenure, which may or may not be followed by a data tenure.

3.11.1 Arbiter Rules

The following rules describe the operation of the arbiter:

1. There are three CPU bus masters that can generate CPU address bus requests: CPU1, CPU2, and the snoop engine. The priority of the agents is
   - CPU1 (the highest priority)
   - CPU2
   - Snoop engine (the lowest priority).

   If more than one bus request is sampled active at the same time, the address bus is granted to the highest priority requesting BM.
2. When no bus master is requesting the CPU address bus, the Bridge parks the bus on the CPU that most recently owned it. This behavior supports multiprocessing applications.
3. The Bridge supports CPU bus masters which conform to the bus arbitration protocol described in Chapter 8 of the *PowerPC 604 User's Manual*. In general, the CPU bus master will deassert BR# for one CPU clock cycle following the assertion of TS#.
4. The Bridge arbiter recognizes an active CPU address bus request from the snoop engine from the time that the Bridge decodes a PCI bus transaction to system memory (FRAME# initially sampled active), until the termination of the PCI bus transaction. The bus request is considered to be negated during PCI bus idle (turn-around) cycles. Note that this bus request always goes inactive between consecutive PCI transactions.

5. The system arbiter (external to and independently of the Bridge) selects the current PCI bus master, who is in general allowed to initiate PCI transactions independently of the state of the CPU bus. Thus a PCI bus master can begin a PCI to memory transaction while a CPU has control of the CPU address bus. In this case:

   5.1 The current CPU bus master is allowed to complete the current transfer:

      5.1.1 The address tenure completes, followed by two CPU clocks during which the CPU address bus is rearbited. Then the snoop is broadcast onto the CPU address bus.

      5.1.2 The data tenure of the CPU transfer completes normally.

   5.2 The Bridge asserts DEVSEL# as usual. It is not delayed by the current CPU address or data bus activity.

   5.3 At the end of the CPU to memory data tenure, there is an idle CPU clock, and then the memory controller begins the PCI to memory transaction.

      5.3.1 On reads, TRDY# is delayed until the first data phase is valid on the PCI bus. In general, pipelined PCI read bursts require 2 fewer PCI clocks to deliver the first beat than are required for bursts that start from a CPU bus idle (non–pipelined bursts).

      5.3.2 On writes, TRDY# is asserted 1 or 2 PCI clocks after the completion of the data phase of the CPU to memory transfer. The posted write buffers in the Bridge will accept the first four 4–byte data phases as 1/2–1–1–1. Subsequent data phases are accepted one at a time as the buffer writes each 4–byte to memory. Write times are affected by various factors, as discussed in section 4.5.

   5.4 If the snoop cycle (see step 5.1.1) is retried by either CPU, the Bridge retries the PCI bus master and grants the CPU bus to the CPU bus agent that retried the cycle (assuming that it is requesting the bus). If both CPUs retry the cycle, the bus is granted first to CPU1.

6. Fairness is enforced by the Bridge arbiter, which will not re–grant the CPU bus to a previous owner before granting it to a new requester. This assures that each of the three agents gets a fair chance to own the bus. No agent is held off of the bus for more cycles than it takes for each of the other two agents to run one transaction (worst case).

   6.1 An apparent exception to this (which does not violate fairness) is that if CPU1, CPU2, or the snoop engine is retried during a cycle, the arbiter will next grant the CPU bus to the bus agent that retried the cycle (assuming that it is requesting the bus). If both of the other agents retry the cycle, the CPU bus is granted first to the one with the highest priority, and then to the other agent. Then the bus is returned to the original owner. The worst case hold-off in this case is three CPU bus transactions.

7. When a CPU begins a CPU to PCI transaction (single-beat only):

   7.1 If the Bridge has the PCI_GNT# (and the PCI bus is idle), the Bridge initiates the PCI transaction.

   7.2 If the Bridge does not have the PCI_GNT#, then:

      7.2.1 If there is currently a PCI to system memory transaction in progress, the Bridge will retry the CPU.

7.2.2 If there is not a PCI to memory transaction in progress (there is a PCI to PCI transaction in progress or the PCI bus is idle and not parked on the Bridge), then the Bridge arbitrates for the PCI bus and begins the transaction.

7.2.2.1 On CPU writes, the first transaction is posted even before the PCI bus is granted to the bridge. Completion of any second transaction depends on completion of the first transaction.

7.2.2.2 On CPU reads, the CPU transfer will not complete until data is returned from the PCI bus.

8. When a CPU begins a CPU to memory transfer, there is no collision with a PCI broadcast snoop cycle, because the CPU bus is granted to the snoop engine throughout the PCI to memory transaction.

3.12 Broadcast Snoop Details

PCI to memory transactions begin with the Bridge decoding and recognizing the transaction, arbiting for the CPU bus, and in the case of writes, completing a broadcast snoop cycle to the CPU bus. The memory-controller-active part of a PCI to memory transaction is delayed by the time it takes to arbitrate for the CPU address bus on writes (waiting for the snoop cycle to complete), if the transaction starts from a CPU bus idle condition. Pipelined PCI to memory transactions (which start while a previous CPU bus transaction is still running), do not suffer this initial delay, but they do have to wait for the completion of the data tenure of the previous transfer in order to begin. Pipelined transactions begin as soon as the memory controller is available: they do not have to wait for the snoop to complete, because it has already completed.

Another CPU address tenure is allowed to begin several CPU clocks before the end of the PCI transaction, as soon as the arbiter knows that it will not be broadcasting another snoop immediately (e.g., on the last beat of a PCI burst).

Table 3-4 shows the TT[0:3] (transfer type) of snoop operations generated to the CPU bus when a PCI master accesses system memory.

Table 3-4. Types of Snoop Cycles for PCI to Memory Operations

PCI Bus Cycle	CPU Bus Snoop – 604, 604e, 760		CPU Bus Snoop – 603, 603e, 740, 750	
	Operation	TT[0:4][1]	Operation	TT[0:4][2]
Memory Read	Clean	00000	Single-Beat Read	01010
Memory Write	Flush Sector	00100	Single-Beat Write with Flush	00010
Initiate Lock (Read)	Single-Beat Write with Flush	00010	Single-Beat Write with Flush	00010

Notes:

1. These settings only work for the 604 family.
2. These settings work for the 604, 604e, 760, 603, 603e, 740, and 750.

The following discussion includes three figures (Figure 3-5, Figure 3-6, and Figure 3-7) that show PCI to memory transactions and the associated broadcast snoop operations. These figures show the following:

- CPU and PCI clocks that are for reference only
- CPU address bus tenures from TS# to AACK# (no retries considered)
- CPU data bus tenures (these are approximate only – their duration is not to scale nor to clock)
- PCI bus tenures (same caveats).

3.12.1 Snoop Cycles From CPU Bus Idle

Figure 3-5 shows a CPU to memory transfer, followed by at least one CPU bus idle cycle, followed by a PCI bus master transaction to memory, and followed by another CPU bus address tenure. Since the PCI transaction begins during a CPU bus idle, as soon as the 660 recognizes the PCI transaction, it grants the CPU bus to itself (the snoop engine), which requires at most 1 PCI clock, and then it broadcasts a snoop cycle on the CPU bus. This snoop cycle is very similar to the address tenures of non-pipelined CPU to memory transfers.

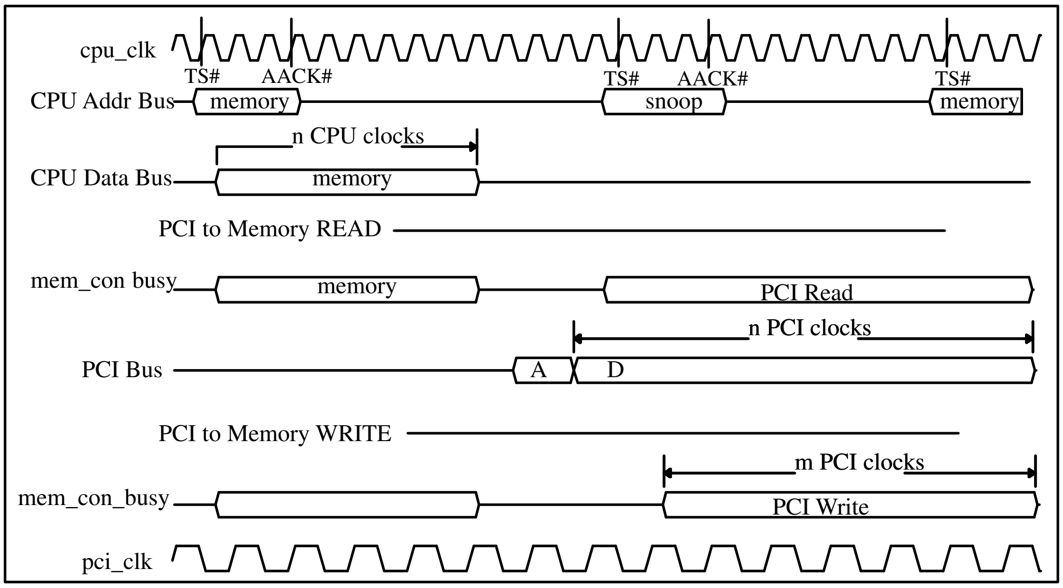


Figure 3-5. PCI Bus Snoops From CPU Bus Idle

If the PCI transaction is a read, the memory controller begins the memory read immediately (if there is a CPU bus retry, the memory controller can stop a read without corrupting the memory). The snoop operation requires 3 CPU clocks to complete, which is less time than is required for the memory controller to access the first 8 bytes of DRAM, so no time is lost to the PCI transaction because of the snoop. From the assertion of FRAME#, it will take t to 9 PCI clocks until the bridge places the first 4–byte on the PCI bus and the PCI bus master can sample TRDY# active. Subsequent data phases of the read are serviced at basically -1-1-1, so the first 4 beats take 8-1-1-1 to complete. Also see Section 4.5.

If the PCI transaction is a write, the memory controller is delayed 2 PCI clocks (3 CPU clocks) for the snoop to complete before beginning the memory access; however, some overhead functions are occurring during this time, so the net effective snoop delay is usually 1 PCI clock. Following successful completion of the snoop, the PCI to memory write data phases are posted, and once the 1 to 2 PCI clock snoop delay is over, the subsequent 3 PCI data beats are accepted at -1-1-1. Thus the write pattern is 5-1-1-1.

3

In both the read and write cases, another CPU address tenure is allowed to begin several CPU clocks before the end of the PCI transaction (as soon as the arbiter knows that it will not be broadcasting another snoop immediately—e.g., on the last beat of a PCI burst).

3.12.2 Pipelined Snoop Cycle Following a CPU Bus Transfer

Figure 3-6 shows a PCI to memory transaction pipelined after a CPU to memory transfer. In this case, the 660 recognizes the initiation of a PCI to memory transaction during the address tenure of a CPU to memory transfer. At the close of the CPU to memory address tenure, up to 2 CPU clocks of CPU address bus are idle for arbitration and internal overhead. Then the bridge broadcasts the PCI snoop on the CPU address bus. The snoop cycle is completed before the end of the CPU to memory transfer, so it imposes no delay on either the CPU transfer or the PCI transaction.

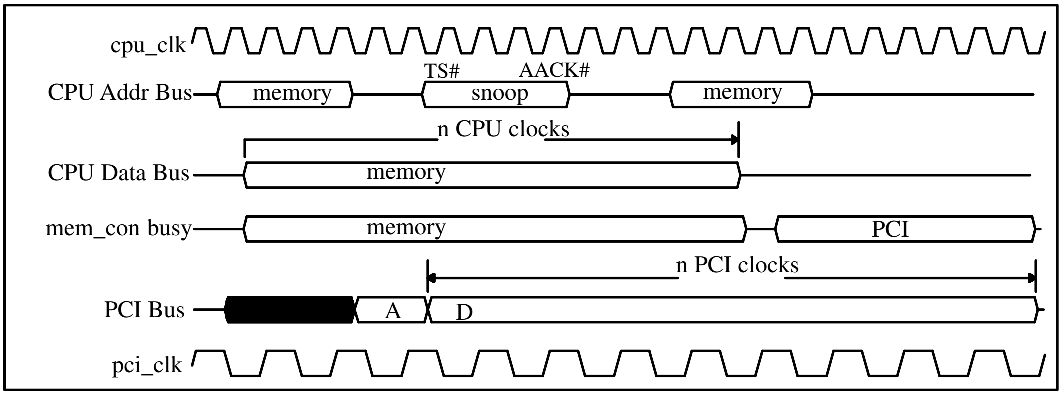


Figure 3-6. Pipelined PCI Bus Snoop Following a CPU Bus Transfer

3.12.3 Pipelined Snoop Cycle Following a Pipelined CPU Bus Transfer

Figure 3-7 shows an initial CPU to memory transfer, followed by a pipelined CPU to memory transfer, and followed by a PCI to memory transfer. The pipelined PCI transaction executes as it did in Figure 3-6. Note that snoop cycles continue to have little to no effect on CPU bus performance during pipelined transactions. In this case, the Bridge recognizes a PCI to memory transaction initiation sometime during the address phase of the second CPU to memory transfer. Note that the Bridge will not complete the CPU address tenure for transfer 2 until the data tenure for tenure 2 begins (1 clock after the end of the data tenure of transfer 1). The third transaction (the pipelined PCI snoop) has no effect on the timing of the two CPU transfers.

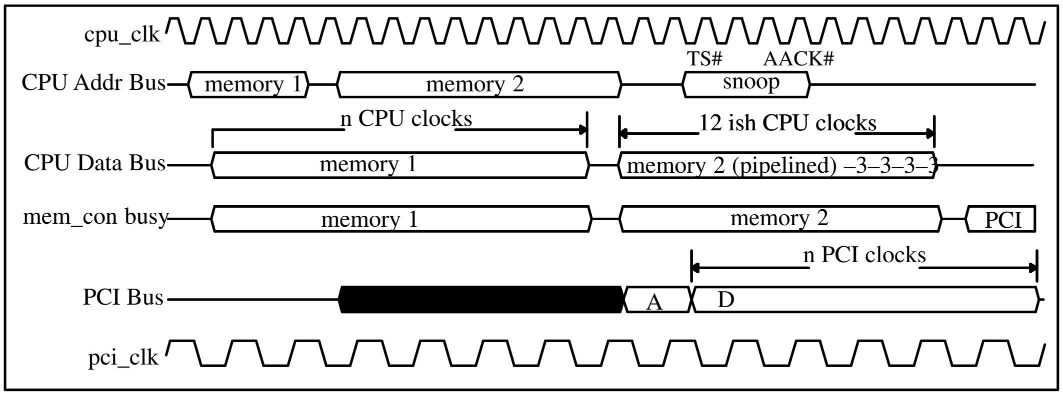


Figure 3-7. Pipelined PCI Bus Snoop Following a Pipelined CPU Bus Transfer

3.12.4 PCI Bus Snoop On Block Boundary During a PCI Burst Read

Figure 3-8 shows a PCI snoop that is incurred by a PCI burst crossing over a cache block 32-byte boundary during a PCI to memory read. See section 4.5.4.1 for a discussion of timing variable **M**.

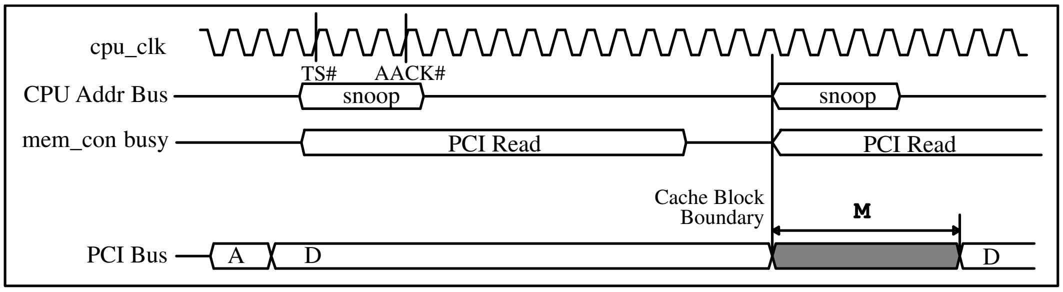


Figure 3-8. PCI Bus Snoop On Block Boundary During a PCI Burst Read

3.12.5 PCI Bus Snoop On Block Boundary During a PCI Burst Write

Figure 3-9 shows a PCI snoop incurred by a PCI burst crossing over a cache block boundary during a PCI to memory write. See section 4.5.4.2 for a discussion of timing variable **X**.

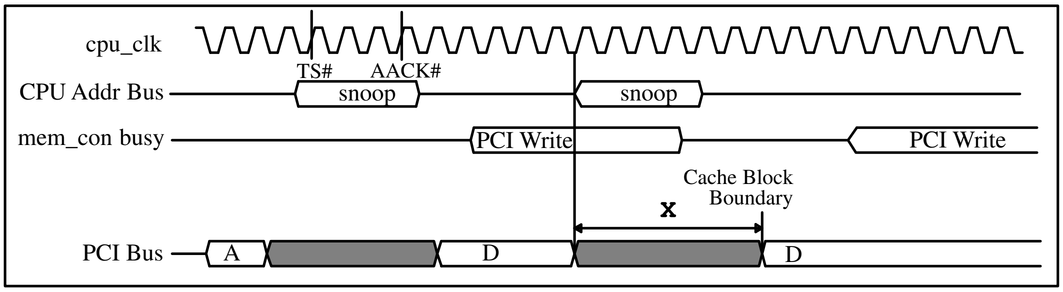


Figure 3-9. PCI Bus Snoop On Block Boundary During a PCI Burst Write

3.13 Related Bridge Control Registers

Bridge Control Register	Index	R/W	Bytes	See
Bridge Options 1	Index BA	R/W	1	10.3.35
Bridge Options 2	Index BB	R/W	1	10.3.36
CPU Bus Error Status	Index C3	R	1	10.3.39
Error Enable 2	Index C4	R/W	1	10.3.40
Error Status 2	Index C5	R/W	1	10.3.41
PCI Bus Error Status	Index C7	R/W	1	10.3.42
CPU/PCI Error Address	Index C8 – CB	R/W	4	10.3.43
Bridge Chip Set Options 3	Index D4	R/W	1	10.3.46
Memory Controller Misc	8000 0821	R/W	1	10.2.2.4
I/O Map Type	8000 0850	R/W	1	10.2.2.9
PCI/BCR Configuration Address	8000 0CF8	R/W	4	10.3.1.1
PCI/BCR Configuration Data	8000 0CFC	R/W	4	10.3.1.2
PCI Type 0 Configuration Addresses IBM27–82650 Compatible	8080 08xx 8080 10xx 8080 20xx 8080 40xx 8080 80xx 8081 00xx 8082 00xx 8084 00xx 8088 00xx 8090 00xx 80A0 00xx 80C0 00xx	R/W	4	3.4.5.1

3

Section 4
PCI Bus

The 660 Bridge supports a 32-bit PCI expansion bus at frequencies up to 33MHz. The PCI bus is compatible with the PCI Specification, revisions 2.0 and 2.1.

The PCI bus can be run at one-half or the same frequency as the CPU bus. CPU:PCI bus operation at 3:1 is not supported. For 1:1 operation, see section 4.6. The 664 automatically detects the frequency ratio.

4.1 PCI Arbitration

The 660 requires an external PCI arbiter such as may be supplied in the ISA bridge. The 660 sends a CPU/660 bus request to the PCI bus arbiter to request ownership of the PCI. The 660 receives a PCI bus grant from the PCI bus arbiter. The 660 follows the PCI specification for host bridges. The PCI arbiter typically parks the PCI bus on the 660, but see section 1.11.8 for exceptions.

4.2 PCI Lock

The 660 does not set PCI locks, but does honor them. Also see Section 5.6.

4.2.1 PCI Busmaster Locks

The PCI_LOCK# signal is an input-only to the 660. The 660 provides resource locking of one 32-byte cache sector (block) of system memory. Once a PCI busmaster sets a lock on a 32-byte block of system memory, the 660 saves the block address. Subsequent accesses to that block from other PCI bus masters or from the CPU bus are retried until the lock is released.

The bridge generates a write-with-flush snoop cycle on the CPU bus when a PCI bus master sets the PCI lock. The write-with-flush snoop cycle causes the L1 and L2 caches to invalidate the locked block, which prevents cache hits on accesses to locked blocks. If the L1 contains modified data (as indicated by a CPU retry), the PCI cycle is retried and the modified data is pushed out to memory.

4.2.2 CPU Bus Locking

The 660 does not set PCI locks when acting as the PCI master.

The 60X processors do not have bus-locking functions. Instead, they use the *load reserve* and *store conditional* instructions (lwarx and stwcx) to implement exclusive access by setting a reservation on a memory block. To work with the lwarx and stwcx instructions, the 660 snoops all PCI accesses to system memory, which allows the CPU that is holding the reservation to detect a violation of the reservation. In addition, the 660 generates a write-with-flush operation on the CPU bus in response to the PCI read that begins a PCI lock.

4.3 660 (Target) Response by PCI Bus Command

Table 4-1 shows the response of the 660 (and the allowed response of other PCI agents) to various PCI bus transactions initiated by a PCI busmaster. The 660 ignores (No response) all PCI bus transactions except PCI memory read and write transactions, which it decodes as possible system memory accesses.

Table 4-1. 660 Bridge Responses to PCI_C/BE[3:0] Bus Commands

C [3:0]	PCI Bus Command	Can a PCI Bus Master Initiate this Transaction?	660 Response to the Transaction	Can Another PCI Target Claim the Transaction?
0000	Interrupt Acknowledge	No. Only the 660 is allowed to initiate.	No response	Yes. The ISA bridge is intended to be the target.
0001	Special Cycle	Yes	No response	Yes
0010	I/O Read	Yes	No response	Yes
0011	I/O Write	Yes	No response	Yes
0100	Reserved	No. Reserved	No response	n/a
0101	Reserved	No. Reserved	No response	n/a
0110	Memory Read	Yes	System memory read	Yes, if no address conflict.
0111	Memory Write	Yes	System memory write	Yes, if no address conflict.
1000	Reserved	No. Reserved	No response	n/a
1001	Reserved	No. Reserved	No response	n/a
1010	Configuration Read	No. Only the 660.	No response	Yes
1011	Configuration Write	No. Only the 660.	No response	Yes
1100	Memory Read Multiple	Yes	System memory read	Yes, if no address conflict.
1101	Dual Address Cycle	Yes	No response	Yes
1110	Memory Read Line	Yes	System memory read	Yes, if no address conflict.
1111	Memory Write and Invalidate	Yes	System memory write	Yes, if no address conflict.

4.4 660 (Target) Response by PCI Memory Address

Table 4-2 shows the mapping of PCI memory accesses to system memory. This is the response of the 660 bridge as a PCI target when a PCI busmaster initiates a PCI memory transaction.

Table 4-2. 660 Bridge Mapping of PCI Memory Space

PCI Bus Address	Other Conditions	Target Cycle Decoded	Target Address	Notes
0 to 2G	IGN_PCI_AD31 Deasserted	Not Decoded	N/A	Ignored by 660.
	IGN_PCI_AD31 Asserted	System Memory *	0 to 2G	Snooped by caches. The 660 broadcasts a snoop onto the CPU bus.
2G to 4G		System Memory *	0 to 2G	Snooped by caches**

Notes:
*Memory does not occupy this entire address space. Accesses to unoccupied space are not decoded.
**In remote ROM mode, the 660 ignores PCI busmaster transactions from 4G–2M to 4G.

For PCI memory transactions from 0 to 2G:

- While IGN_PCI_AD31 is negated, PCI busmaster memory accesses in the 0 to 2G address range are ignored by the 660.
- While IGN_PCI_AD31 is asserted, the 660 maps PCI busmaster memory accesses from 0 to 16M directly to system memory at 0 to 16M in order to support ISA busmaster accesses to system memory. (The 660 actually maps the entire 0 to 2G range to system memory while IGN_PCI_AD31 is asserted.)

The 660 maps PCI busmaster memory accesses from 2G to 4G as system memory from 0 to 2G. All PCI master accesses to system memory are snooped by the caches.

Note that the CPU can generate PCI memory accesses with PCI addresses from 0 to 1G – 2M. However, the range of addresses in which to locate PCI memory so that it can be addressed by both PCI busmaster and the CPU, is from 0 to 1G – 2M.

4.5 PCI Access to System Memory

4.5.1 Memory Access Range and Limitations

PCI memory reads and writes by PCI bus masters are decoded by the 660 to determine if they access system memory. PCI memory reads and writes to addresses from 2G to 4G on the PCI bus are mapped by the 660 as system memory reads and writes from 0G to 2G. These PCI to memory transactions are checked against the top_of_memory variable to determine is a given access is to a populated bank. The logic of the 660 does not recognize unpopulated holes in the memory banks. PCI accesses to unpopulated locations below the top_of_memory are undefined.

PCI accesses to system memory are not limited to 32 bytes. PCI burst-mode accesses are limited only by the size of memory, PCI bus latency restrictions, and the PCI disconnect counter.

4.5.2 ISA Master Transactions

The 660 samples IGN_PCI_AD31 during PCI busmaster memory transactions from 0 to 2G. If IGN_PCI_AD31 is negated, the 660 ignores the transaction, and if IGN_PCI_AD31 is asserted, the 660 forwards the transaction to system memory. In theory, the IGN_PCI_AD31 signal can be used by any PCI agent for this purpose, but to ensure PR–P compliance, this signal should be asserted only while the ISA bridge is initiating a PCI to memory transaction on behalf of an ISA master. One way to generate IGN_PCI_AD31 is to AND together the PCI_GNT# signals of all of the PCI agents except the ISA bridge and the 660. This will assert IGN_PCI_AD31 (during a PCI transaction) only while either the 660 or the ISA bridge is the initiator (and the 660 knows when it is the initiator).

The required connectivity of IGN_PCI_AD31 prevents the ISA bridge from initiating peer to peer PCI memory transactions in the 0 to 2G range. The ISA bridge is allowed to initiate PCI memory transactions from 2G to 4G, and other PCI transaction types (I/O &etc.).

4.5.3 Memory Access Sequence

When a PCI access is decoded as a system memory read or write, the memory and CPU bus are requested and, when granted, a snoop cycle to the CPU bus and a memory cycle to system memory are generated. If the processor indicates a snoop hit in the L1 cache (ARTRY# asserted), then the memory cycle is abandoned and the PCI cycle is retried. The CPU then does a snoop push. The L2 cache does not need to do a snoop push because it is write-through, and, therefore, system memory always contains the result of all write cycles.

4.5.3.1 Snooping

Section 3.12 contains a detailed description of the snoop process.

4.5.3.2 Writes

If the PCI access is a memory burst write access, the 660 PCI interface performs data gathering before initiating the cycle to the memory controller. The data gathering involves combining two PCI write cycles into one memory write cycle if the address of the first write cycle is even.

Minimum initial write access time to 70ns DRAM when the CPU bus is 66MHz and the PCI bus is 33MHz is 5-1-1-1 -3-1-1-1 PCI clocks for 4-4-4-4 -4-4-4-4 bytes of data (14 PCI clocks for 32 bytes of data). Subsequent data phases of the same burst are generally serviced at -3-1-1-1 -3-1-1-1 (12 PCI clocks for 32 bytes of data). See Section 4.5.4.2.

4.5.3.3 Reads

If the PCI access is a memory burst read access, the 664 PCI interface performs memory pre-fetching when it initiates cycles to the memory controller. The pre-fetching involves loading or pre-loading 32 bytes from the memory for eight 4-byte PCI read cycles. Pre-fetching is only done within the same cache line.

Minimum initial read access time from 70ns DRAM when the CPU bus is 66MHz and the PCI bus is 33MHz, is 8-1-1-1 -1-1-1-1 PCI clocks for 4-4-4-4 -4-4-4-4 bytes of data (15 PCI clocks for 32 bytes of data). Subsequent data phases of the same burst are generally serviced at -7-1-1-1 -1-1-1-1 (14 PCI clocks for 32 bytes of data). See Section 4.5.4.1.

4.5.4 PCI to Memory Burst Transfers

PCI to memory burst transfers continue to normal completion unless one of the following occurs:

1. The initiating PCI bus master disconnects. The 660 handles all master disconnects correctly.
2. The 660 target disconnects on a 1 M boundary. The Bridge disconnects on all 1M boundaries.
3. The 660 target disconnects because the PCI disconnect timer has timed out. See section 10.3.21.
4. The CPU retries the snoop cycle that the 660 broadcast on the CPU bus. In this case, the 660 target retries the PCI bus master. (Note that L2 hits do not affect the PCI to memory transaction. Read hits have no effect, and write hits cause the internal L2 to invalidate the block.)
5. The 660 will target disconnect the PCI bus master if the refresh timer (see section 10.3.45) times out. In this case, the 660 will disconnect at the end of the current data phase for writes, or at the end of the current cache block, for reads.

4.5.4.1 Detailed Read Burst Sequence Timing

The basic PCI to memory read sequence with 70ns DRAM is 8-1-1-1 -1-1-1-1 -7-1-1-1 -1-1-1-1 -7..., giving a peak burst read rate of 32 bytes in 14 PCI clocks, or about 73MBps with a 33MHz PCI clock. This scenario holds while the RAS# timer (10us typical) does not time out and no refresh is requested (15us typ).

The actual detailed read sequence is affected by several factors, such as the speed of the DRAM, refresh requests, memory arbitration delays, page and/or bank misses, and cache boundary alignment. Table 4-3 shows the details of the various sequences that a PCI to memory burst read will experience, depending on the address (relative to a cache block boundary) of the first data phase of the transaction. The starting address of the numbering sequence shown on the top row was arbitrarily chosen as xx00, and could be any 32–byte aligned boundary. The times shown in Table 4-3 are in PCI clock cycles, and do not include any cycles that the PCI master spends acquiring the PCI bus from the PCI bus arbiter. The initial data phase is timed from the assertion of FRAME# to the PCI clock at which the PCI master samples TRDY# active. Subsequent data phase times are from the PCI clock at which the previous TRDY# was sampled active to the PCI clock at which the current TRDY# is sampled active.

All the numbers shown in Table 4-3 are for parity (or none) operation. These numbers are also correct for ECC mode operation.

Table 4-3. PCI to Memory Read Burst Sequence Timing

S								S									
00	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C	40	...
N	1	1	1	1	1	1	1	M	1	1	1	1	1	1	1	M	...
	N	1	1	1	1	1	1	M	1	1	1	1	1	1	1	M	...
		N	1	1	1	1	1	M	1	1	1	1	1	1	1	M	...
			N	1	1	1	1	M	1	1	1	1	1	1	1	M	...
				N	1	1	1	M	1	1	1	1	1	1	1	M	...
					N	1	1	M	1	1	1	1	1	1	1	M	...
						N	1	M	1	1	1	1	1	1	1	M	...
							N	M	1	1	1	1	1	1	1	M	...
								N	1	1	1	1	1	1	1	M	...

S indicates a cache block boundary at 0 mod 32. Snoops are broadcast to the CPU bus when a PCI burst crosses one of these boundaries.

N is the number of PCI clocks required from the assertion of FRAME# until the master samples the first TRDY# (from the 660) active, and is a function of snoop and memory arbitration delays. If the CPU is accessing memory when the PCI agent begins the memory read burst, the 660 waits until the CPU completes the current CPU access before allowing the PCI to memory read to proceed. If the RAS# watchdog timer has timed out, the memory controller will precharge the RAS# lines, and if the refresh timer has timed out, the memory controller will do a refresh operation.

N (min)	= 5	This occurs when the memory controller is idle and no refresh or RAS# timeout occurs, and the access produces a page hit.
N (typ)	= 8 or 9	This occurs if the memory controller is in the middle (beat 3 of 4) of serving a CPU burst transfer when the PCI burst starts, and no refresh or RAS# timeout occurs.
N (max)	= 26	This occurs when CPU1 is just starting a burst transfer to memory, followed by CPU2 starting a burst transfer to memory, after which a refresh happens to be required.

M is a function of a 2-clock snoop delay and other delays caused by bridge overhead functions. Whenever the memory access crosses a cache block boundary, the Bridge broadcasts a snoop cycle on the CPU bus.

M (typ)	= 6 or 7	Unless a refresh or RAS# timeout occurs.
M (typ)	= 7 or 8	This occurs for a refresh or RAS# timeout.

The memory controller, running at its own speed, requests up to 4, 8-byte memory reads (into 8, 4-byte buffers in the 663) while the PCI target engine of the 660 is servicing the memory read transaction. Under worst case conditions (slow memory, etc.), the memory controller just keeps up with the PCI bus, and N goes up. Under better conditions, the memory controller gets ahead of the PCI read process, and N decreases.

4.5.4.2 Detailed Write Burst Sequence Timing

The basic PCI to memory write sequence to 70ns DRAM is 5-1-1-1 -3-1-1-1 -3-1-1-1 -3-1-1-1 -3..., giving a peak burst write rate of 32 bytes in 12 PCI clocks, or about 85MBps with a 33MHz PCI clock. This scenario holds while the RAS# timer (10us typical) does not time out, the burst remains within the same 4K memory page, and no refresh is requested (15us typ).

The actual detailed write sequence is affected by several factors, such as the speed of the DRAM, refresh requests, memory arbitration delays, page and/or bank misses, and cache boundary alignment. Table 4-4 shows the details of the various sequences that a PCI to memory burst write will experience, depending on the address (relative to a cache block boundary) of the first data phase of the transaction. The starting address of the numbering sequence shown on the top row was arbitrarily chosen as xx00, and could be any 32–byte aligned boundary. The times shown in Table 4-4 are in PCI clock cycles, and do not include any cycles that the PCI master spends acquiring the PCI bus from the PCI bus arbiter. The initial data phase is timed from the assertion of FRAME# to the PCI clock at which the PCI master samples TRDY# active. Subsequent data phase times are from the PCI clock at which the previous TRDY# was sampled active to the PCI clock at which the current TRDY# is sampled active.

All the numbers shown in Table 4-4 are for parity (or none) operation. The numbers are also correct for ECC mode operation as long as all the writes are gather-store pairs. Incurring a RMW operation costs 3 PCI_CLKs.

Table 4-4. PCI to Memory Write Burst Sequence Timing

S								S									
00	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C	40	...
W	1	1	1	Y	1	Z	1	X	1	Z	1	Z	1	Z	1	X	...
	W	1	1	Y	1	Z	1	X	1	Z	1	Z	1	Z	1	X	...
		W	1	1	1	Y	1	X	1	Z	1	Z	1	Z	1	X	...
			W	1	1	Y	1	X	1	Z	1	Z	1	Z	1	X	...
				W	1	1	1	G	1	Z	1	Z	1	Z	1	X	...
					W	1	1	G	1	Z	1	Z	1	Z	1	X	...
						W	1	G	1	Z	1	Z	1	Z	1	X	...
							W	G	1	Z	1	Z	1	Z	1	X	...
								W	1	1	1	Y	1	Z	1	X	...

S indicates a cache block boundary at 0 mod 32. Snoops are broadcast to the CPU bus when a PCI burst crosses one of these boundaries.

W is a function of a 1.5 PCI clock snoop delay and memory arbitration delays. If the CPU is accessing memory when the PCI agent begins the memory write burst, the 660 waits until the CPU completes the current CPU access before allowing the PCI to memory write to proceed. If the RAS# watchdog timer has timed out, the memory controller will precharge the RAS# lines, and if the refresh timer has timed out, the memory controller will do a refresh operation.

W (min) = 5 This occurs when the memory controller is idle and no refresh or RAS# timeout occurs.
W (typ) = 6 or 7 This occurs if the memory controller is in the middle (beat 3 of 4) of serving a CPU burst transfer when the PCI burst starts, and no refresh or RAS# timeout occurs.
W (max) = 23 This occurs when CPU1 is just starting a burst transfer to memory, followed by CPU2 starting a burst transfer to memory, after which a refresh happens to be required.

X is a function of snoop delays only. Whenever the memory access crosses a cache block boundary, the Bridge broadcasts a snoop cycle on the CPU bus. (Due to the posted write buffer structure, delays incurred by crossing a page boundary here do not show up until later in the sequence.)

X= 3 Always. (The only benefit to disabling PCI snooping or enabling pre–snooping is to reduce this delay to 1. Otherwise neither function increases performance.)

Y is a function of memory latency. This page and/or bank miss delay can only be incurred at a page boundary, but shows up here due to the posted write buffer structure. The Bridge has a 4 x 4 posted PCI write buffer, which allows it to accept data phases from the PCI bus while the memory controller is busy servicing page misses. This minimizes the transfer delays caused by these memory overhead functions.

Y (typ) = 1 This occurs for a page hit with no refresh. This is also the minimum.
Y (mid) = 2 This occurs for a page miss with no refresh.
Y (max) = 4 This occurs for a refresh (which also forces a page miss).

Z is a function of a subset of the **W** factors (RAS# timeouts and refresh operations). This delay is only incurred due to a RAS# timeout or refresh request that has occurred since the last **W**, **Y**, **Z**, or **G**.

Z (typ) = 2 to 3 This occurs for no refresh and no RAS# timeout.
Z (max) = 3 to 4 This occurs for either a RAS# timeout or a refresh operation.

G is the combination of **X** and **Y**, and is equal to the longer of **X** and **Y**.

4.6 1:1 CPU:PCI Bus Ratio Operation

When using the 660 with a CPU:PCI bus ratio of 1:1, the PCI arbiter must be programmed to not park the PCI bus on the 660.

Operation of the 660 with a CPU:PCI bus ratio of 1:1 is supported only with the use of external logic. The logic deasserts PCI_GNT# to the 664 from the beginning of the address phase of a CPU to PCI access, until the CPU bus is no longer pipelined (DBB# deasserts for a clock).

4.6.1 1:1 PAL Connectivity

This section shows one possible implementation of the 1:1 bus ratio external logic in a 16V8 PAL. See Figure 4-1.

1. PAL clock is CPU_CLK or PCI_CLK, since they will be at the same speed.
2. PAL inputs (just add this connection to the current nets): TS#, CPU_A[0], and DBB#.
3. Cut PCI_GNT# between the 664 and the PCI arbiter. Connect PCI_GNT# from the arbiter to the PCI_GNT_IN# input of the PAL. Connect the PCI_GNT_OUT# output of the PAL to the PCI_GNT# input of the 664.

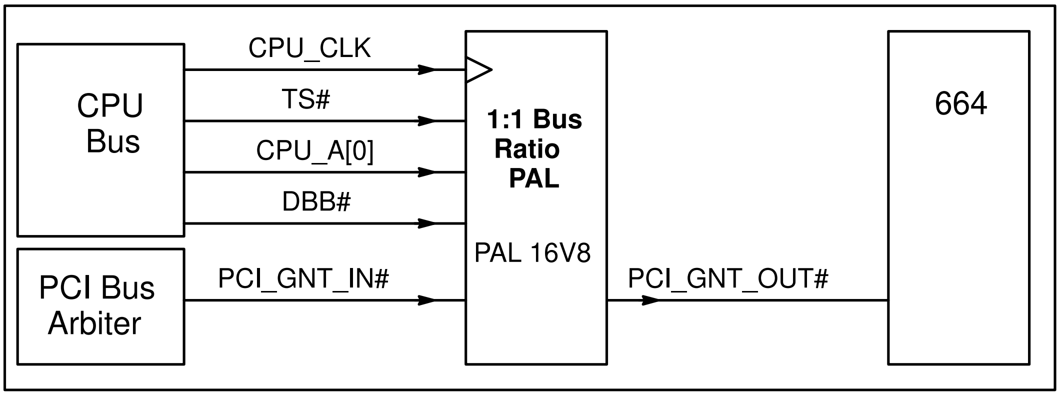


Figure 4-1. PAL

4.6.2 1:1 PAL Equations

```
TITLE    1_to_1.pds
PATTERN  none
REVISION 1.0
COMPANY  IBM
DATE     02/14/96
CHIP   _1_to_1 PALCE16V8

;------------------------- PIN Declarations --------------------
;
;  Predefined
;
;PIN  1   CPU_CLK          ; CLOCK
;PIN 10   GND              ; GROUND
;PIN 11   REG_OE#          ; OUTPUT ENABLE FOR REGISTERED OUTPUTS
;PIN 20   VCC              ; VCC
;
;  Inputs
;

PIN  2    TS_
PIN  3    A0
PIN  4    DBB_
PIN  5    PCI_GNT_IN_
;PIN  6
;PIN  7
;PIN  8
;PIN  9
;
;  outputs
;
;
;  Registered outputs
;
;PIN 12                      REG
;PIN 13                      REG
;PIN 14                      REG
;PIN 15                      REG
;PIN 16                      REG
;PIN 17                      REG
PIN 18    MASK_PCI_GNT       REG
PIN 19    PCI_GNT_OUT_      COMB


;
```

```
;------------------ Boolean Equation Segment -----------------
;
EQUATIONS


MASK_PCI_GNT = /TS_ * A0 * /DBB_    ;START WHEN TS* IS LOW AND
             + MASK_PCI_GNT * /DBB_ ;THE CPU BUS IS PIPELINED
                                    ;HOLD WHILE DBB* IS LOW

PCI_GNT_OUT_ = PCI_GNT_IN_
             + MASK_PCI_GNT


;---------------------- END OF FILE ---------------------------
```

4.7 Related Bridge Control Registers

Bridge Control Register	Index	R/W	Bytes	See
Memory Controller Misc	8000 0821	R/W	1	10.2.2.4
PCI/BCR Configuration Address	8000 0CF8	R/W	4	10.3.1.1
PCI/BCR Configuration Data	8000 0CFC	R/W	4	10.3.1.2
PCI Type 0 Configuration Addresses IBM27–82650 Compatible	8080 08xx thru 80C0 00xx	R/W	4	3.4.5.1
PCI Vendor ID	Index 00 – 01	R	2	10.3.3
PCI Device ID	Index 02 – 03	R	2	10.3.4
PCI Command	Index 04 – 05	R/W	2	10.3.5
PCI Device Status	Index 06 – 07	R/W	2	10.3.6
Revision ID	Index 08	R	1	10.3.7
PCI Standard Programming Interface	Index 09	R	1	10.3.8
PCI Subclass Code	Index 0A	R	1	10.3.9
PCI Class Code	Index 0B	R	1	10.3.10
PCI Cache Line Size	Index 0C	R	1	10.3.11
PCI Latency Timer	Index 0D	R	1	10.3.12
PCI Header Type	Index 0E	R	1	10.3.13
PCI Built-in Self-Test (BIST) Control	Index 0F	R	1	10.3.14
PCI Interrupt Line	Index 3C	R	1	10.3.15
PCI Interrupt Pin	Index 3D	R	1	10.3.16
PCI MIN_GNT	Index 3E	R	1	10.3.17
PCI MAX_LAT	Index 3F	R	1	10.3.18
PCI Bus Number	Index 40	R	1	10.3.19
PCI Subordinate Bus Number	Index 41	R	1	10.3.20
PCI Disconnect Counter	Index 42	R/W	1	10.3.21
PCI Special Cycle Address BCR	Index 44 –45	R	2	10.3.22
Error Enable 1	Index C0	R/W	1	10.3.37
Error Status 1	Index C1	R/W	1	10.3.38
Error Enable 2	Index C4	R/W	1	10.3.40
Error Status 2	Index C5	R/W	1	10.3.41
PCI Bus Error Status	Index C7	R/W	1	10.3.42
CPU/PCI Error Address	Index C8 – CB	R/W	4	10.3.43

Section 5
DRAM

The memory controller in the 660 controls the system memory DRAM. The system memory can be accessed from both the CPU bus and the PCI bus.

5.1 Features and Supported Devices

- Supports memory operations for the PowerPC Architecture™
- Data bus path 72 bits wide—64 data bits and eight bits of optional ECC or parity data
- Eight SIMM sockets supported, with empty SIMM sockets allowed at any position
- Eight RAS# outputs, eight CAS# outputs, and two write-enable outputs
- Supports industry-standard 8-byte (168-pin) SIMMs of 8M, 16M, 32M, 64M, and 128M that can be individually installed for a minimum of 8M and a maximum of 1G
- Supports industry-standard 4-byte (72-pin) SIMMs of 4M, 8M, 16M, 32M, 64M, and 128M that must be installed in pairs for a minimum of 8M and a maximum of 1G
- Mixed use of different size SIMMs, including mixed 4-byte and 8-byte SIMMs
- Full refresh support, including refresh address counter and programmable DRAM refresh timer with low-power mode
- Burst-mode memory address generation logic
  - 32-byte CPU bursts to memory
  - Variable length PCI burst to memory
- Little-endian and big-endian addressing and byte swapping modes
- Provides row and column address multiplexing for DRAM SIMMs requiring the following addressing:

SIMM type	SIMM size	Addressing
72-pin	4 Meg	10 x 10
	8 Meg	10 x 10
	16 Meg	11 x 11
	32 Meg	11 x 11
	64 Meg	12 x 12
168-pin	8 Meg	10 x 10
	16 Meg	11 x 10
	32 Meg	12 x 10 or 11 x 11
	64 Meg	12 x 11
	128 Meg	12 x 12

5.1.1 SIMM Nomenclature

The term SIMM is used extensively to mean DRAM memory module, without implying the physical implementation of the module, which can be a SIMM, DIMM, or other package.

5.1.2 DRAM Timing

The memory controller timing parameters are programmable to allow optimization of timings based on speed of DRAM, clock frequency, and layout topology. Timing must be programmed based on the slowest DRAM installed.

- Support for fast page-mode DRAMs
- Support for extended-data-out (EDO) DRAM (hyper-page mode)
- If 70ns DRAM is used in a system with the CPU bus at 66MHz, the minimum access times for initial (not pipelined) CPU to memory transfers with page mode and EDO DRAM are as follows:

Transfer	EDO DRAM	Page Mode DRAM	Note
Initial Read Burst	10–3–3–3	11–4–4–4	CPU clocks for 32 bytes
Initial Write Burst	5–3–3–3	5–4–4–4	CPU clocks for 32 bytes

- In the same system, the times for a pipelined burst following a read are as follows:

Transfer	EDO DRAM	Page Mode DRAM	Note
Page Hit Read	(152MBPS) –5–3–3–3	–5–4–4–4	CPU clocks for 32 bytes
Page Hit Write	–3–3–3–3	–3–3–4–4	CPU clocks for 32 bytes

- In the same system, the times for a pipelined burst following a write are as follows:

Transfer	EDO DRAM	Page Mode DRAM	Note
Page Hit Read	–9–3–3–3	–11–4–4–4	CPU clocks for 32 bytes
Page Hit Write	(152MBPS) –5–3–3–3	–6–3–4–4	CPU clocks for 32 bytes

- Other minimum memory timings are as follows:
  - PCI to memory read at 66MHz CPU and 33MHz PCI
    8-1-1-1 -1-1-1-1 7-1-1-1 -1-1-1-1 7-1-1-1 -1-1-1-1 ... 7-1-1-1 -1-1-1-1
  - PCI to memory write at 66MHz CPU and 33MHz PCI
    5-1-1-1 -3-1-1-1 3-1-1-1 -3-1-1-1 3-1-1-1 -3-1-1-1 ... 3-1-1-1 -3-1-1-1

5.1.3 DRAM Error Checking

The 660 supports either no parity or one bit per byte parity DRAM SIMMs, in which one parity bit is associated and accessed with each byte. The 660 is BCR programmable to support either no parity, odd parity, or ECC data error detection and correction. ECC is implemented using standard parity SIMMs. All installed SIMMs must support the selected error checking protocol.

Systems without error checking cost the least. Parity checking mode allows a standard level of error protection with no performance impact. ECC mode allows detection and correction of all single-bit errors and detection of all two-bit errors. ECC mode adds one CPU clock to the latency of CPU to memory reads, and does not effect the timing of 8-byte and 32- byte writes

5.2 DRAM Performance

5.2.1 Memory Timing Parameters

Most memory controller timing parameters can be adjusted to maximize the performance of the system with the available resources. This adjustment is done by programming various memory controller BCRs. Figure 5-1 shows the various programmable memory timing variables. These variables control the number of CPU_CLKs between various events. The actual amount of time between the events shown will also be affected by various other factors such as clock to output delays. The CPU_CLK signal shown is not meant to be contiguous, as the number of clocks between various events is programmable.

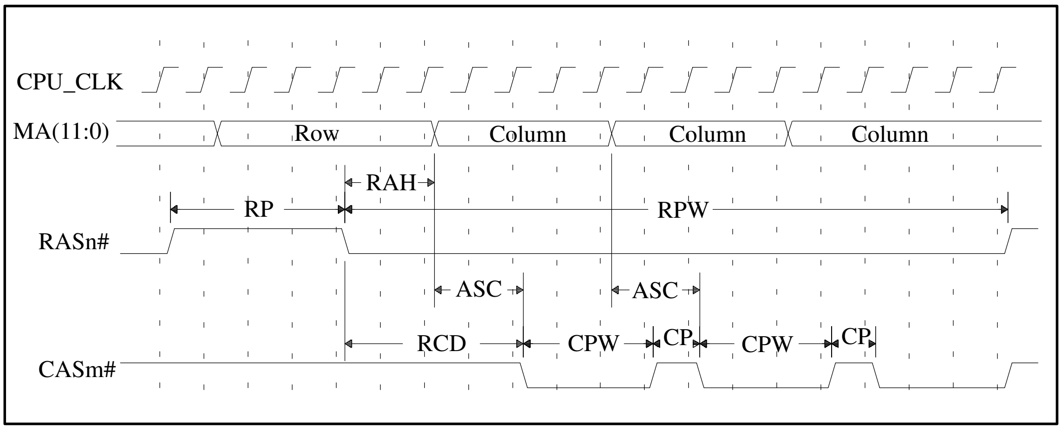


Figure 5-1. CPU to Memory Transfer Timing Parameters

Table 5-1 shows the function, location, and section references for the variables shown in Figure 5-1.

Table 5-1. Memory Timing Parameters

Variable	Function	BCR	Section
ASC	Column Address Setup (min)	Memory Timing Register 2	5.2.1.2
CP	CAS# Precharge	Memory Timing Register 2	5.2.1.2
CPW	CAS# Pulse Width (Read & Write)	Memory Timing Register 2	5.2.1.2
RAH	Row Address Hold (min)	Memory Timing Register 1	5.2.1.1
RCD	RAS# to CAS# Delay (min)	Memory Timing Register 2	5.2.1.2
RP	RAS# Precharge	Memory Timing Register 1	5.2.1.1
RPW	RAS# Pulse Width	Memory Timing Register 1	5.2.1.1

Note that ASC, RAH, and RCD are minimums. if RAH + ASC does not equal RCD, then the larger value will be used such that:

- If RCD < RAH + ASC, then the actual RCD will be stretched to equal RAH + ASC.
- If RCD > RAH + ASC, then the actual RAH will be stretched to equal RCD – ASC.

5.2.1.1 Memory Timing Register 1

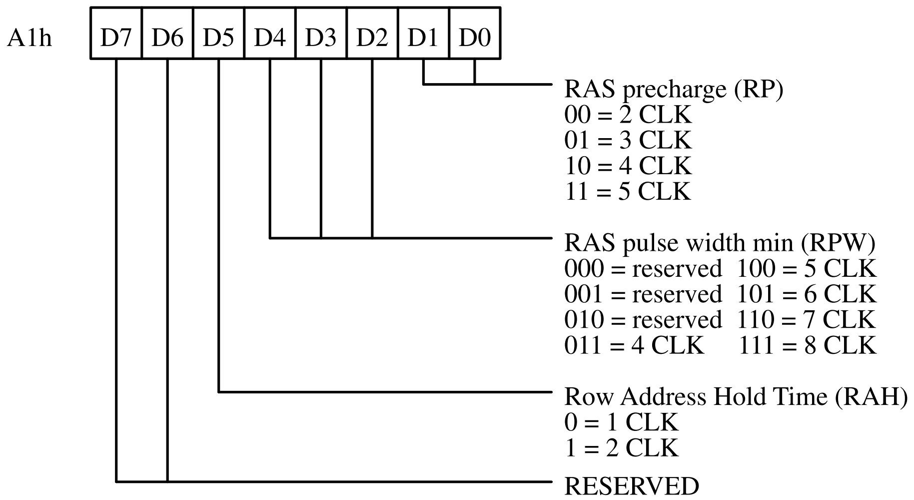
Index A1	Read/Write	Reset to 3Fh

This BCR determines the timing of RAS# signal assertion for memory cycles. RAS# timing must support the worst-case timing for the slowest DRAM installed in the system. See Section 5.2.1.

Bits 1:0 These bits control the number of CPU clocks for RAS# precharge.

Bits 4:2 These bits control the minimum allowed RAS# pulse width except on refresh. For refresh, the RAS# pulse width is hard-coded to three PCI clocks.

Bit 5 This bit controls the number of CPU clocks that the row address is held following the assertion of RAS#.

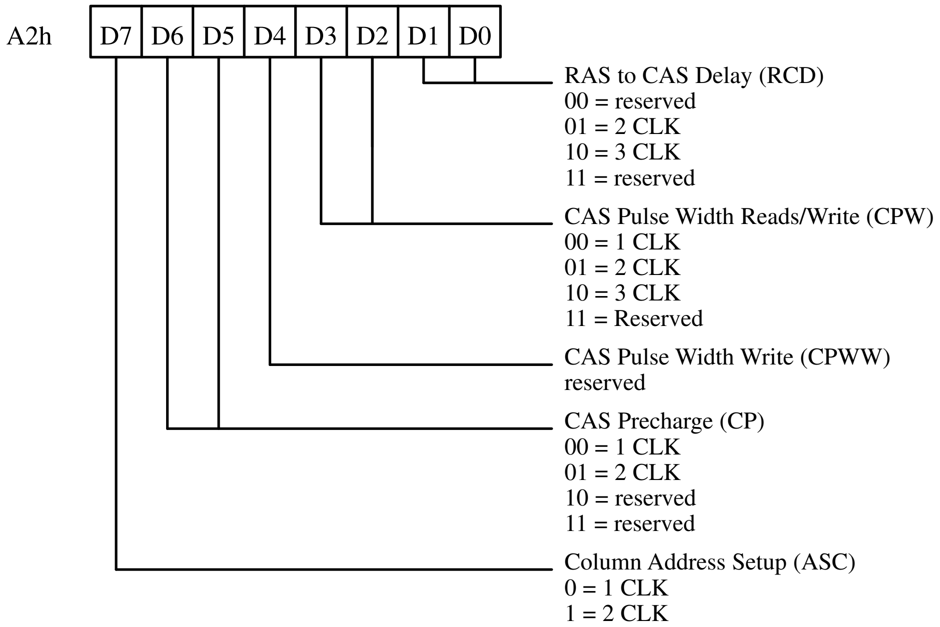


5.2.1.2 Memory Timing Register 2

Index A2	Read/Write	Reset to AEh

This BCR determines the timing of CAS# signal assertion for memory cycles. CAS# timing must support the worst-case timing for the slowest DRAM installed in the system. See Section 5.2.1.

When using the 660 with a 603 or 604 family CPU, and the CPU:PCI bus frequency ratio is 1:1, then CP plus CPW must be set less than or equal to 3 clocks total (not the number of settings. e.g., when cp=01 and cpw=00, that is equal to 3 clocks).



5.2.1.3 RAS# Watchdog Timer BCR

Index B6	Read/Write	Reset to 53h

This BCR limits the maximum RAS# active pulse width. The value of this BCR represents the maximum amount of time that any RAS# can remain active in units of eight CPU bus clocks. The timer (down-counter) associated with this BCR is reloaded on the assertion of any RAS# line. On expiration of the timer, the 660 drops out of page mode to deassert the RAS# lines.

In response to the RESET# signal, this register is reset to 53h. This value results in a maximum RAS# active time of just under 10us at 66MHz. This is the value required by most 4-byte and 8-byte SIMMs. The value of the BCR must be reprogrammed if the CPU bus frequency is not 66MHz or a different RAS# pulse width is required.

5

5.2.2 General Case DRAM Timing Calculations

The memory controller of the 660 features programmable DRAM access timing. DRAM timing is programmed into Memory Timing Register 1 (MTR1) and Memory Timing Register 2 (MTR2). See section 5.2.1 for an explanation of the format of these BCRs. All memory controller outputs are switched on the rising edge of the CPU clock, with the exception of signal assertion during refresh operations, which is timed from the PCI clock. The RAS# Watchdog Timer Register, the Refresh Timer Divisor Register and the Bridge Chipset Options 3 (BCO3) BCRs also have an effect on DRAM timing.

The values programmed into these registers are a function of the clock frequencies, the timing requirements of the memory, the amount of memory installed (capacitive loading), the mode of operation (EDO vs. standard & special vs. general case), the timing requirements of the 660, the type and arrangement of buffering for the MA (memory address) signals, the clock skew between the 663 and the 664, and the net lengths of the signals to/from the memory (flight time). The calculations below ignore the factors of clock skew and flight time.

This section discusses the timing calculations that are appropriate to 660 memory controller design. Except as noted in section 5.2.4, the timing recommendations in this section apply to all 660 configurations.

Each of the nine equations below lists a register or register bits that governs a memory timing parameter. An equation is then provided for calculating the required value based on the timing requirements of the memory and the 660. MTR1[1:0] refers to Memory Timing Register 1 bits 1:0. See Figure 5-1 and Table 5-1.

1. MTR1[1:0] – RAS# precharge (RP). The critical path that determines the RAS# precharge requirement is RAS# rising to RAS# falling. The minimum RAS# precharge time supplied by the 660 must exceed the minimum precharge time required by the DRAM. Make:

   RAS# precharge (RP) > Trp min . . . . . . * DRAM min RAS# precharge

2. MTR1[4:2] – RAS# pulse width (RPW). The critical path that determines the RAS# pulse width requirement is RAS# falling to RAS# rising. The minimum RAS# pulse width supplied by the 660 must exceed the minimum RAS# pulse width required by the DRAM plus 5ns. Make:

   RAS# pulse width (RPW) > Tras min . . . . * DRAM min RAS# pulse width
   \+ 5ns . . . . * pulse width shrinks (note 1)

3. MTR2[6:5] – CAS# precharge (CP). The critical path that determines the CAS# precharge requirement is CAS# rising to CAS# falling. The minimum CAS# precharge time supplied by the 660 must exceed the minimum precharge time required by the DRAM. Make:

   CAS# precharge (CP) > Tcp min . . . . * DRAM min CAS# precharge

5

4. MTR2[3:2] – CAS# pulse width (CPW). The critical path that determines the CAS# pulse width requirement is the data access time from CAS# plus the setup time into the 663. Thus the minimum CAS# pulse width provided by the 660 must exceed the minimum CAS# pulse width required by the DRAM, plus these factors. Note that the 663 samples memory data on the clock that CAS# is deasserted for standard DRAM and on the clock after CAS# is deasserted for EDO DRAM. Make:

   ```
   CAS# pulse width (CPW) > T CAS# fall max . . .  * CAS# active out if 664
   (+ 1 CLK if EDO)         + Tcac  . . . . . . . . . . .  * DRAM data access from CAS#
                            + MD setup max . . .  * MEM_DATA setup into 663
   ```

   The factor (+ 1 CLK if EDO) is included in the equation only if EDO DRAM is used. Note that CPW must be set to 3 or fewer clocks.

5. MTR2[7] – Column Address Setup (ASC). There are two critical paths that determine the Col addr setup requirement. The minimum column address setup time supplied by the 660 must exceed both constraints. The first is Tasc of the memory. Make:

   ```
   a) Col Addr Setup (ASC) >   Tasc min . . . . .  * DRAM min col addr setup time
                               + T MA max . .  * MA[11:0] valid out of 664
                               + T 244 max . .  * Buffer delay
                               – T CAS# max   * CAS# active out of 664
   ```

   The second critical path is the data access time from MA plus the setup time into the 663. Make:

   ```
   b) Col addr setup (ASC)
      + CAS# pulse width (CPW) >  T MA max . . . . .  * MA[11:0] valid out of  664
      (+ 1 CLK if EDO)            + T 244 max . . .  * Buffer delay
                                  + Taa min  . . . . .  * DRAM data valid from col addr
                                                          valid
                                  + MD setup max   * MEM_DATA setup into 663
   ```

6. MTR2[1:0] – RAS# to CAS# delay. The minimum RAS# to CAS# delay provided by the 660 must exceed the timing of the critical path that determines the RAS# to CAS# delay, which is the data access time from RAS# plus the setup time into the 663. Make:

   ```
   RAS# to CAS# delay (RCD)
   + CAS# pulse width (CPW) >  T RAS# fall max .  * max 660 RAS# fall time (note 1)
   (+ 1 CLK if EDO)            + Trac min . . . . . .  * DRAM data access from RAS#
                               + MD setup max/   * MEM_DATA setup into 663+
   ```

7. MTR1[5] – Row address hold time. The row address hold time must be set to the RAS#–to–CAS# delay minus the Column address setup. The timing for the row address hold time can also be calculated as follows. Make:

   Row addr hold (RAH) > Trah min . . * DRAM min row addr hold
   + T244 min . . . . * Buffer delay
   + T MA max . . . . . . * MA[11:0] valid out of 664
   – T RAS# fall max . * max 660 RAS# fall time (note 1)

   Note 1: The 664 drivers that drive the RAS# and CAS# signals are slower falling than rising. This causes active high pulse widths to grow by 0 to 5ns and active low pulse widths to shrink by 0 to 5ns.

5

8. Refresh Timer Divisor. The refresh timing divisor is clocked by the PCI clock. The required value is calculated as follows:

   Refresh rate = period(Tref) / period(PCI clock)

9. RAS# watchdog timer. The RAS# watchdog timer must be set to limit the max RAS# pulse width:

   RAS# watchdog timer = Tras max / [period(CPU clock) * 8].

5.2.3 General Case DRAM Timing Examples

This section presents example DRAM timing calculations based on the equations found in Section 5.2.2. Except as noted in Section 5.2.4, the timing recommendations in this section apply to all 660 configurations.

In the equations below, first the capactive loads are calculated based on the quantity and types of SIMMs and the buffers used. Next, the timing characteristics are calculated based on the capacitive loads. Finally, the timing requirements and register values are caculated.

5.2.3.1 70ns DRAM Calculations

Ex 1: Assume 70ns standard DRAM memory, four 72–pin DRAM SIMMs, a CPU bus cycle time of 15ns (66.7Mhz), and MA[11:0] buffered by an FCT244 (four SIMMs per driver).

Capacitive loads:

RAS# = 2*62pf + 30pf = 154pf
CAS# = 2*62pf + 30pf = 154pf
MA (to buffer) = 30pf
MA (to memory) = 4*161pf + 40pf = 684pf

Timing Characteristics:

RAS# = 13.2ns + .025*(154pf–50pf) = 16ns
CAS# = 13.3ns + .025*(154pf–50pf) = 16ns
MA to buffer = 13.6ns + .025*(30pf–50pf) = 13ns
MA to memory = 4.6ns + .007(684pf–50pf) = 9ns
663 memory data input setup = 6ns

664 Output timings are at 50pf. For loads greater than 50pf, 0.025ns/pF are added.

Timing Requirements and register value calculations:

1. 4 CLK * 15ns > 50ns
   60ns > 50ns .... MTR1[1:0]=10

2. 5 CLK * 15ns > 70ns + 5ns
   75ns > 75ns .... MTR1[4:2]=100

3. 1 CLK * 15ns > 10ns
   15ns > 10ns .... MTR2[6:5]=00

4. 3 CLK * 15ns > 16ns + 20ns + 6ns
   45ns > 42ns .... MTR2[3:2]=10

5. a. 1 CLK * 15ns > 0ns + 13ns + 9ns – 16ns
   15ns > 6ns

   b. (1 + 3)CLK * 15ns > 13ns + 9ns + 35ns + 6ns
   60ns > 63ns .... MTR2[7]=0 (see Note 2)

6. (3 + 3)CLK * 15ns > 16ns + 70ns + 6ns
   90ns > 92ns .... MTR2[1:0]=10 (see Note 2)

7. 2 CLK * 15ns > 10ns + 4ns + 13ns – 16ns
   30ns > 11ns .... MTR1[5]=1

   Results: Memory Timing Register 1 (index A1h) = 32h
   Memory Timing Register 2 (index A2h) = 0Ah

   Note 2: The timing analysis above includes two timing violations (path #5b is violated by 5% and path #6 is violated by 2%). More conservate system designers may wish to use the value MTR1=32h to ensure all timing requirements are met under all worst–case conditions.

8. Refresh rate = 15.6us/30ns = 520d = 208h

   Refresh Timer Divisor (index D1h,D0h) = 0208h

9. RAS# watchdog timer = 10,000ns / (15ns*8) = 83d = 53h

   RAS# watchdog timer register (index B6h) = 53h (default value).

5

5.2.3.2 60ns DRAM Calculations

Ex 2: Same assumptions as above, but with 60ns DRAM

1. 3 CLK * 15ns > 40ns
   45ns > 40ns .... MTR1[1:0]=01
2. 5 CLK * 15ns > 60ns + 5ns
   75ns > 65ns .... MTR1[4:2]=100
3. 1 CLK * 15ns > 10ns
   15ns > 10ns .... MTR2[6:5]=00
4. 3 CLK * 15ns > 16ns + 15ns + 6ns
   45ns > 37ns . ... MTR2[3:2]=10
5. a. 1 CLK * 15ns > 0ns + 13ns + 9ns – 16ns
   15ns > 6ns

   b. (1 + 3)CLK * 15ns > 13ns + 9ns + 30ns + 6ns
   60ns > 58ns .... MTR2[7]=0
6. (2 + 3)CLK * 15ns > 16ns + 60ns + 6ns
   75ns > 82ns .... MTR2[1:0]=01 (see Note 3)
7. 1 CLK * 15ns > 10ns + 4ns + 13ns – 16ns
   15ns > 11ns .... MTR1[5]=0

   Results: Memory Timing Register 1 (index A1h) = 11h
   Memory Timing Register 2 (index A2h) = 09h

   Note 3: The timing analysis above includes one timing violation (path #6 is violated by 9%). More conservate system designers may wish to use the values MTR1=31h MTR2=0Ah to ensure all timing requirements are met under complete worst–case conditions.
8. Refresh rate = 15.6us/30ns = 520d = 208h

   Refresh Timer Divisor (index D1h,D0h) = 0208h
9. RAS# watchdog timer = 10,000ns / (15ns*8) = 83d = 53h

   RAS# watchdog timer register (index B6h) = 53h (default value).

5.2.3.3 50ns DRAM Calculations

Ex 3: Same assumptions as above, but with 50ns DRAM

1. 2 CLK * 15ns > 30ns
   30ns > 30ns .... MTR1[1:0]=00
2. 4 CLK * 15ns > 50ns + 5ns
   60ns > 55ns .... MTR1[4:2]=011
3. 1 CLK * 15ns > 10ns
   15ns > 10ns .... MTR2[6:5]=00
4. 2 CLK * 15ns > 16ns + 13ns + 6ns
   30ns > 35ns .... MTR2[3:2]=01 (see note 4)
5. a. 1 CLK * 15ns > 0ns + 13ns + 9ns – 16ns
   15ns > 6ns

   b. (1 + 2)CLK * 15ns > 13ns + 9ns + 25ns + 6ns
   45ns > 53ns .... MTR2[7]=0 (see note 4)
6. (3 + 2)CLK * 15ns > 16ns + 50ns + 6ns
   75ns > 72ns .... MTR2[1:0]=10
7. 2 CLK * 15ns > 10ns + 4ns + 13ns – 16ns
   30ns > 11ns .... MTR1[5]=1

   Results: Memory Timing Register 1 (index A1h) = 2Ch
   Memory Timing Register 2 (index A2h) = 06h

   Note 4: The timing analysis above includes two timing violations (path #4 is violated by 14% and path #5b is violated by 15%). More conservate system designers may wish to use the values MTR1=0Ch, MTR2=09h to ensure all timing requirements are met under complete worst–case conditions.

8. Refresh rate = 15.6us/30ns = 520d = 208h

   Refresh Timer Divisor (index D1h,D0h) = 0208h
9. RAS# watchdog timer = 10,000ns / (15ns*8) = 83d = 53h

   RAS# watchdog timer register (index B6h) = 53h (default value).

5.2.3.4 60ns EDO DRAM Calculations

Ex 4: Same assumptions as above, except using 60ns EDO DRAM

1. 3 CLK * 15ns > 40ns
   45ns > 40ns .... MTR1[1:0]=01
2. 5 CLK * 15ns > 60ns + 5ns
   75ns > 65ns .... MTR1[4:2]=100
3. 1 CLK * 15ns > 10ns
   15ns > 10ns .... MTR2[6:5]=00
4. (2 + 1)CLK * 15ns > 16ns + 15ns + 6ns
   45ns > 37ns .... MTR2[3:2]=01
5. a. 1 CLK * 15ns > 0ns + 13ns + 9ns – 16ns
   15ns > 6ns

   b. (1 + 2 + 1)CLK * 15ns > 13ns + 9ns + 30ns + 6ns
   60ns > 58ns ...... MTR2[7]=0
6. (2 + 2 + 1)CLK * 15ns > 16ns + 60ns + 6ns
   75ns > 82ns .... MTR2[1:0]=01 (see Note 5)
7. 1 CLK * 15ns > 10ns + 4ns + 13ns – 16ns
   15ns > 11ns .... MTR1[5]=0

   Results: Memory Timing Register 1 (index A1h) = 11h
   Memory Timing Register 2 (index A2h) = 05h

   Note 5: The timing analysis above includes one timing violation (path #6 is violated by 9%). More conservate system designers may wish to use the values MTR1=31h, MTR2=06h to ensure all timing requirements are met under complete worst–case conditions.
8. Refresh rate = 15.6us/30ns = 520d = 208h

   Refresh Timer Divisor (index D1h,D0h) = 0208h
9. RAS# watchdog timer = 10,000ns / (15ns*8) = 83d = 53h

   RAS# watchdog timer register (index B6h) = 53h (default value).

5.2.3.5 Aggressive Timing Summary

Table 5-2 contains a summary of recommended general case aggressive DRAM timing. The table also shows the resulting performance of the memory controller. Aggressive timings may generate slight violations of certain worst case timing constraints. In many cases, these violations are of only theoretical interest, since the conditions required to produce the violations are of such low probability.

The first section of Table 5-2 shows the settings of the memory controller BCRs. The second section of the table shows access times from the memory controller idle state, which it enters when it is not servicing a read or write request. The other two sections of the table show access times during back to back burst transfers. The middle section of the table shows access times for the second of any pair of back to back transfers where the first transfer is a read. The lower section of the table shows access times for the second of any pair of back to back transfers where the first transfer is a write.

Table 5-2. General Case Aggressive DRAM Timing Summary

Transfer	70ns Aggressive	60ns Aggressive	50ns Aggressive	60ns EDO Aggressive	Note
Memory Timing Register 1 (MTR1)	32	11	2C	11	
Memory Timing Register 2 (MTR2)	0A	09	06	05	
Bridge Chipset Options 3 (BCO3)	08	08	08	0C	
Initial Read Burst	11-4-4-4	10-4-4-4	10-3-3-3	10-3-3-3	(3)
Initial Write Burst	5-4-4-4	5-3-4-4	5-4-3-3	5-3-3-3	
For a pipelined burst transfer immediately following a read:					
Page Hit Read	-4-4-4-4	-4-4-4-4	-4-3-3-3	-5-3-3-3	(3)
Page Hit Write	-3-3-4-4	-3-3-4-4	-3-3-3-3	-3-3-3-3	
Page Miss and Bank Miss Read	-8-4-4-4	-7-4-4-4	-7-3-3-3	-7-3-3-3	(1,3)
Page Miss and Bank Hit Read	-10-4-4-4	-8-4-4-4	-7-3-3-3	-8-3-3-3	(2,3)
Page Miss and Bank Miss Write	-3-3-4-4	-3-3-4-4	-3-3-3-3	-3-3-3-3	(1)
Page Miss and Bank Hit Write	-3-5-4-4	-3-3-4-4	-3-3-3-3	-3-3-3-3	(2)
For a pipelined burst transfer immediately following a write:					
Page Hit Read	-11-4-4-4	-10-4-4-4	-8-3-3-3	-9-3-3-3	(3)
Page Hit Write	-6-3-4-4	-5-3-4-4	-4-3-3-3	-5-3-3-3	
Page Miss and Bank Miss Read	-14-4-4-4	-13-4-4-4	-11-3-3-3	-11-3-3-3	(1,3)
Page Miss and Bank Hit Read	-16-4-4-4	-14-4-4-4	-11-3-3-3	-12-3-3-3	(2,3)
Page Miss and Bank Miss Write	-6-6-4-4	-6-5-4-4	-5-5-3-3	-5-4-3-3	(1)
Page Miss and Bank Hit Write	-6-8-4-4	-6-6-4-4	-6-5-3-3	-5-5-3-3	(2)

1) The RAS# of the new bank is high and has been high (precharging) for the minimum RAS# high time. The bridge places the address on the address lines and asserts RAS#.
2) The access is a page miss, but within the same bank, so the RAS# line must be sent high for at least the minimum RAS# high (precharge) time. The bridge also places the new address on the address lines and asserts RAS#.
3) If asynchronous SRAMs are used with the internal L2 controller, an additional clock cycle is added to the fourth beat of any CPU to memory read burst that causes a cache miss. For example, following a read—a pipelined page hit, cache miss, burst read with EDO DRAM, requires –3–3–3–3 CPU clocks when the L2 uses burst SRAMs and –3–3–3–4 CPU clocks when the L2 uses asynchronous SRAMs. This extra beat is caused by delaying the final TA# by one CPU_CLK to allow the asynchronous SRAM sufficient data hold time for the fourth beat.
4) Refresh Rate set to 0208h. RAS# watchdog timer BCR set to 53h.

5

5.2.3.6 Conservative Timing Summary

Table 5-2 contains a summary of recommended general case conservative DRAM timing. The table also shows the resulting performance of the memory controller. These conservative timings may be too conservative for many applications. These timings meet all of the worst case timing constraints for the systems described in the examples sections above.

Table 5-3. General Case Conservative DRAM Timing Summary

Transfer	70ns Conservative	60ns Conservative	50ns Conservative	60ns EDO Conservative	Note
Memory Timing Register 1 (MTR1)	32	31	0C	31	
Memory Timing Register 2 (MTR2)	0E or 0A	0A	09	06	
Bridge Chipset Options 3 (BCO3)	08	08	08	0C	
Initial Read Burst	12-5-5-5	11-4-4-4	10-4-4-4		(3)
Initial Write Burst	5-4-5-5	5-4-4-4	5-3-4-4		
For a pipelined burst transfer immediately following a read:					
Page Hit Read	-5-5-5-5	-4-4-4-4	-4-4-4-4		(3)
Page Hit Write	-3-3-5-5	-3-3-4-4	-3-3-4-4		
Page Miss and Bank Miss Read	-9-5-5-5	-8-4-4-4	-7-4-4-4		(1,3)
Page Miss and Bank Hit Read	-11-5-5-5	-9-4-4-4	-7-4-4-4		(2,3)
Page Miss and Bank Miss Write	-3-3-5-5	-3-3-4-4	-3-3-4-4		(1)
Page Miss and Bank Hit Write	-3-5-5-5	-3-4-4-4	-3-3-4-4		(2)
For a pipelined burst transfer immediately following a write:					
Page Hit Read	-15-5-5-5	-11-4-4-4	-11-4-4-4		(3)
Page Hit Write	-6-3-5-5	-6-3-4-4	-5-3-4-4		
Page Miss and Bank Miss Read	-17-5-5-5	-14-4-4-4	-13-4-4-4		(1,3)
Page Miss and Bank Hit Read	-19-5-5-5	-15-4-4-4	-13-4-4-4		(2,3)
Page Miss and Bank Miss Write	-6-6-5-5	-6-6-4-4	-6-5-4-4		(1)
Page Miss and Bank Hit Write	-6-9-5-5	-6-7-4-4	-6-5-4-4		(2)

See Table 5-2 for notes 1, 2, and 3. Refresh Rate set to 0208h. RAS# watchdog timer BCR set to 53h.

5.2.4 Special Case Memory Controller Operation

The memory controller special case occurs for a specific set of 660 configurations. Memory controller special case operation is generally slower than general case operation. There are several recommended options for avoiding special case operation.

The special case applies to a specific hardware configuration that can lead to the 660 executing a series of operations that can under certain conditions require the 660 to switch all of the CPU data bus outputs on the clock before the CPU samples the data. This operation can in rare cases cause a simultaneous switching fault (incorrect data returned to the CPU bus) unless the memory timing register settings are slowed to compensate.

The user has the choice of either avoiding the special case (see section 5.2.4.2) or exercising one of the options for special case operation (slowing memory controller operation with faster DRAM, disabling page hits, or using performance enhancement logic).

5.2.4.1 Required Conditions for Special Case Operation

Special case memory controller operation occurs only while all of the following conditions are true (if any of the conditions are not true, the memory controller operates in the general case):

1. Memory is set to parity (not ECC) error checking. Note that disabling memory error checking does not avoid special case operation.
2. Memory is set to standard (not EDO) DRAM.
3. The 660 L2 is present/enabled (and set to sync).
4. The L2 cache type bit is set to sync SRAM (not sync).
5. CAS# pulse–width + Column address setup time is set to 4 or less clocks.
6. The CPU bus clock frequency is above 50Mhz.
7. The CPU runs two back–to–back memory reads.
8. Both memory reads are in the same memory page (page–hit).
9. The second memory read starts exactly two clocks before the first read's data tenure ends (2nd read's TS# is two clocks before 1st read's last TA#).

5.2.4.2 Avoiding The Special Case

Special case operation can be avoided by changing the system configuration in at least one of the following ways:

1. Change memory error checking from parity (or none) to ECC using Bridge Chip Set Options 3 BCR (index D4h) bit 1. This avoids condition 1 by increasing the memory access timing from that used in parity to that used in ECC mode. Note that the timing changes even if ECC error checking is turned off.
2. Change DRAM from page-mode to EDO, and set Bridge Chip Set Options 3 BCR (index D4h) bit 0 to 1. This avoids condition 2.
3. Disable and/or remove the 660 onboard L2 (and set SRAM type to async). This avoids conditions 3 and 4.
4. Change from asynchronous SRAM to synchronous SRAM (leave the 660 onboard L2 enabled) using Bridge Chip Set Options 3 BCR (index D4h) bit 3. This avoids condition 4.
5. Limit the CPU bus frequency to 50MHz or less. This avoids condition 6 by moving the sampling window to a clear region.

5.2.4.3 Special Case Option 1 – Disable Page Mode

Special case operation can be avoided by changing the setting of the Timer Register (index B6h) to zero. This prevents the memory controller from running page hits, which avoids condition 8. This option may provide the best memory performance for software which causes a low percentage of page hits.

5.2.4.4 Special Case Option 2 – Change the DRAM Timing

Special case operation can be avoided by changing the memory timing registers (see Table 5-4) to avoid condition 5.

Table 5-4. Special Case DRAM Timing Summary

Transfer	70ns Special Conser vative	70ns Special Aggressive	60ns Special	50ns Special	Note
Memory Timing Register 1 (MTR1)	32	12	11	0C	
Memory Timing Register 2 (MTR2)	0E or 0A	8A	8A	8A	
Bridge Chipset Options 3 (BCO3)	08	08	08	08	
Initial Read Burst	12-5-5-5	11-4-4-4	11-4-4-4	11-4-4-4	(3)
Initial Write Burst	5-4-5-5	5-4-4-4	5-4-4-4	5-4-4-4	
For a pipelined burst transfer immediately following a read:					
Page Hit Read	-5-5-5-5	-5-4-4-4	-5-4-4-4	-5-4-4-4	(3)
Page Hit Write	-3-3-5-5	-3-3-4-4	-3-3-4-4	-3-3-4-4	
Page Miss and Bank Miss Read	-9-5-5-5	-8-4-4-4	-8-4-4-4	-8-4-4-4	(1,3)
Page Miss and Bank Hit Read	-11-5-5-5	-10-4-4-4	-9-4-4-4	-8-4-4-4	(2,3)
Page Miss and Bank Miss Write	-3-3-5-5	-3-3-4-4	-3-3-4-4	-3-3-4-4	(1)
Page Miss and Bank Hit Write	-3-5-5-5	-3-5-4-4	-3-4-4-4	-3-3-4-4	(2)
For a pipelined burst transfer immediately following a write:					
Page Hit Read	-15-5-5-5	-12-4-4-4	-12-4-4-4	-12-4-4-4	(3)
Page Hit Write	-6-3-5-5	-6-3-4-4	-6-3-4-4	-6-3-4-4	
Page Miss and Bank Miss Read	-17-5-5-5	-14-4-4-4	-14-4-4-4	-14-4-4-4	(1,3)
Page Miss and Bank Hit Read	-19-5-5-5	-16-4-4-4	-15-4-4-4	-14-4-4-4	(2,3)
Page Miss and Bank Miss Write	-6-6-5-5	-6-6-4-4	-6-6-4-4	-6-6-4-4	(1)
Page Miss and Bank Hit Write	-6-9-5-5	-6-8-4-4	-6-7-4-4	-6-6-4-4	(2)

See Table 5-2 for notes 1 , 2, and 3. Refresh Rate set to 0208h. RAS# watchdog timer BCR set to 53h.

5.2.4.5 Special Case Option 3 – Performance Enhancement PAL

If special case operation is unavoidable, and disabling page hits or slowing the faster DRAM access timing (as shown in Table 5-4) is undesirable, then a performance enhancement PAL can be installed. The PAL moves the 660 out of special case operation by addressing condition 9 (Deassert BG1# (and BG2#) on the third clock before the current data tenure ends):

The PAL forces BG1# (and BG2#) deasserted:

1. From the beginning of each data tenure until the first TA# (this covers single beat reads), and
2. On the third clock after the second TA# of a burst (this covers burst reads with two clock CAS# pulse width), and
3. On the clock after the third TA# of a burst (this covers burst reads with three clock CAS# pulse width).

This option may effect the maximum operating frequency of the CPU bus due to the additional delay through the PAL in the BG1# (and BG2#) path. Clock to BG# out is nine ns.

- The memory timing effect of the performance enhancement PAL is to speed up special case DRAM accesses to the same speed as general case accesses.

5.2.4.6 Performance Enhancement PAL Design

This implementation of the PAL equations uses a single 16V8. It was developed for a Release 3.0 IBM PowerPC 604 SMP Reference Design, and it requires the addition of one wire to the board, as well as populating and unpopulating some configuration resistors.

1. PAL clock is CPU_CLK.
2. PAL inputs (no trace cuts): DBB#, SRAM_OE#, TA#.
3. Cut BG1 (and BG2# if MP) between the 664 and the CPU, and run them through the PAL as shown in Figure 5-2.

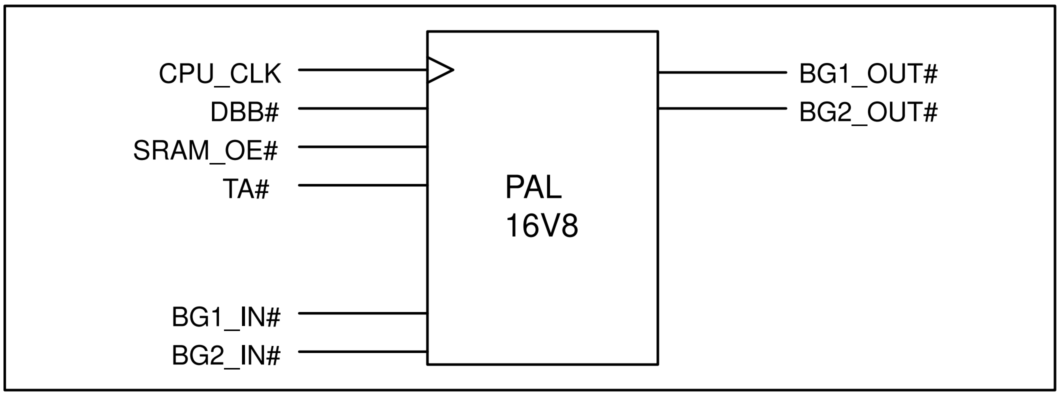


Figure 5-2. PAL

5

PAL Equations

```
;————————————————————————————————————————————————————————————————
;
; File Name:      K12_FIX1.PDS
; For:            Kauai 1.2
; PAL Type:       PAL16V8–5, 20–pin
; PAL Equation Format: PALASM
;
;———————————————————————— Declaration Segment ————————————————————
TITLE           K12_FIX1.pds
PATTERN         none
REVISION        1.0
COMPANY         IBM
DATE            01/12/96

CHIP            _K12_FIX1  PALCE16V8

;————————————————————————— PIN Declarations ——————————————————————
;
;  Predefined
;
;PIN  1   CPU_CLK       ; CLOCK
;PIN 10   GND           ; GROUND
;PIN 11   REG_OE#       ; OUTPUT ENABLE FOR REGISTERED OUTPUTS
;PIN 20   VCC           ; VCC

;
;  Inputs
;

PIN  2   DBB_
PIN  3   TA_
PIN  4   SRAM_OE_
PIN  5   BG1_IN_
PIN  6   BG2_IN_
;PIN  7
;PIN  8
;PIN  9

;
;  outputs
;
PIN 18   BG2_OUT_      COMB
```

```
PIN 19   BG1_OUT_      COMB
;
;  Registered outputs
;
PIN 12   TA_CNT1              REG
PIN 13   TA_CNT2              REG
PIN 14   TA_CNT3              REG
PIN 15   D_TA_CNT2            REG
PIN 16   DD_TA_CNT2           REG
PIN 17   MASK_BG              REG

;
;------------------------- Boolean Equation Segment -------------------------
;
EQUATIONS
;
; Output enables for Comb logic (Reg OE's controllered by pin 11)
;

; NONE – always enabled

;
;   Equations for registers
;


TA_CNT1 = /TA_ * /TA_CNT1 * /TA_CNT2 * /TA_CNT3 + TA_CNT1 * TA_ * /DBB_

TA_CNT2 = /TA_ * TA_CNT1 * /TA_CNT2 * /TA_CNT3 + TA_CNT2 * TA_ * /DBB_

TA_CNT3 = /TA_ * /TA_CNT1 *  TA_CNT2 * /TA_CNT3 + TA_CNT3 * TA_ * /DBB_


D_TA_CNT2 = TA_CNT2

DD_TA_CNT2 = D_TA_CNT2


;*************************************************************************************************
;
;  MASK_BG comments:
; 1st term – deassert BG from beginning of data tenure til 1st TA#.
;           This covers single–beat memory accesses.
; 2nd term – deassert BG on 2nd CLK after 2nd TA#. This covers
;           burst accesses with two clk CAS# pulse width
; 3rd term – deassert BG on 1st CLK after 3nd TA#. This covers
;           burst accesses with three clk CAS# pulse width
```

5

```
;*****************************************************************************************************

MASK_BG = /TA_CNT1 * /TA_CNT2 * /TA_CNT3 * TA_ * /DBB_ * SRAM_OE_ +
D_TA_CNT2 * /DD_TA_CNT2 * SRAM_OE_ + TA_CNT2 * /TA_ * /MASK_BG * SRAM_OE_


BG1_OUT_ = BG1_IN_ + MASK_BG

BG2_OUT_ = BG2_IN_ + MASK_BG

;—————————————————————— END OF FILE ——————————————————————
```

5.2.5 Page Hit and Page Miss

PowerPC CPU bus memory transfers have the following characteristic behavior. When a CPU issues a memory access followed immediately by another memory access, the second access is typically from the same page of memory. On the other hand, if the second memory access does not immediately follow the first one (so that the CPU bus goes idle) then the second memory access is typically a page miss. Thus the majority of memory accesses following a bus idle condition are page misses. 660 memory performance is optimized by assuming that a CPU to memory transfer from bus idle will be a page miss.

When neither the CPU or the PCI is accessing memory, the memory controller goes to the idle state, and all RAS# lines are precharged (deasserted). Deasserting the RAS# lines at idle begins to satisfy the minimum RAS# precharge time requirement. Assuming that the first access out of idle will be a page miss, this technique allows the memory controller to reduce the time required for the initial beat of the burst DRAM read or write access by three CPU clocks. If the initial access is a page hit, this technique results in an increase in access time of two CPU clocks. A net gain is realized whenever the system is experiencing more page misses from bus idle than page hits from bus idle.

For the first beat of pipelined transactions, the memory controller checks the MA[11:0] memory address for a page hit. If the address is within the same 8K memory page as the previous memory access it is a page hit, the row address currently latched into the DRAM is considered valid, and the bridge accesses the DRAM using CAS# cycles. On page misses, the bridge latches the new row address into the DRAMs before it accesses the DRAM using CAS# cycles.

On CPU to memory bursts, only the address of the first beat of the burst is checked for page hits, because the following three beats are always within the same memory page.

On PCI to memory bursts, the address of each data phase of the burst is checked for page hits.

5.2.6 CPU to Memory Access Pipelining

CPU to memory accesses are pipelined, with the result that during a series of back-to-back CPU to memory accesses, all transfers following the initial transfer are faster. The information from the address tenure of the subsequent transfers is processed by the bridge while the data tenure of the preceding transfer is still active.

Considering a series of CPU to memory read transfers using 60ns EDO DRAM, the initial burst requires 10-3-3-3 CPU clocks. If this transfer is followed immediately (back-to-back) by another CPU to memory transfer, the required cycle time is -5-3-3-3. As long as the transfers are back-to-back, they are pipelined, and can be retired at this pipelined rate.

5.2.7 Extended Data Out (EDO) DRAM

The 660 is designed to support hyper-page-mode DRAM, sometimes called extended data out (EDO) DRAM. Information about the operation of the 660 with EDO DRAM is distributed throughout this section.

5.3 System Memory Addressing

5.3.1 DRAM Logical Organization

The DRAM system implemented by the 660 is logically arranged as shown in Figure 5-3. Each block shown in Figure 5-3 is a 9 bit DRAM composed of 8 data bits and 1 parity (or check) bit that is accessed whenever the 8 data bits are accessed. The RAS# lines strobe in the row address. The CAS# lines strobe in the column address. For a block to activate (from idle) for either a read or a write, both the RAS# and CAS# line to it must be activated in the proper sequence. After the initial access, the device can deliver data in fast page mode with only CAS# strobes.

The CAS# lines can be thought of as byte enables, and the RAS# lines as bank enables.

The WE# signal goes to all (each of) the devices in the memory array. The OE# of each DRAM device is tied active. This signal is not required to be deasserted at any time, since the DRAMs only enable their output drivers when so instructed by the RAS#/CAS# protocol.

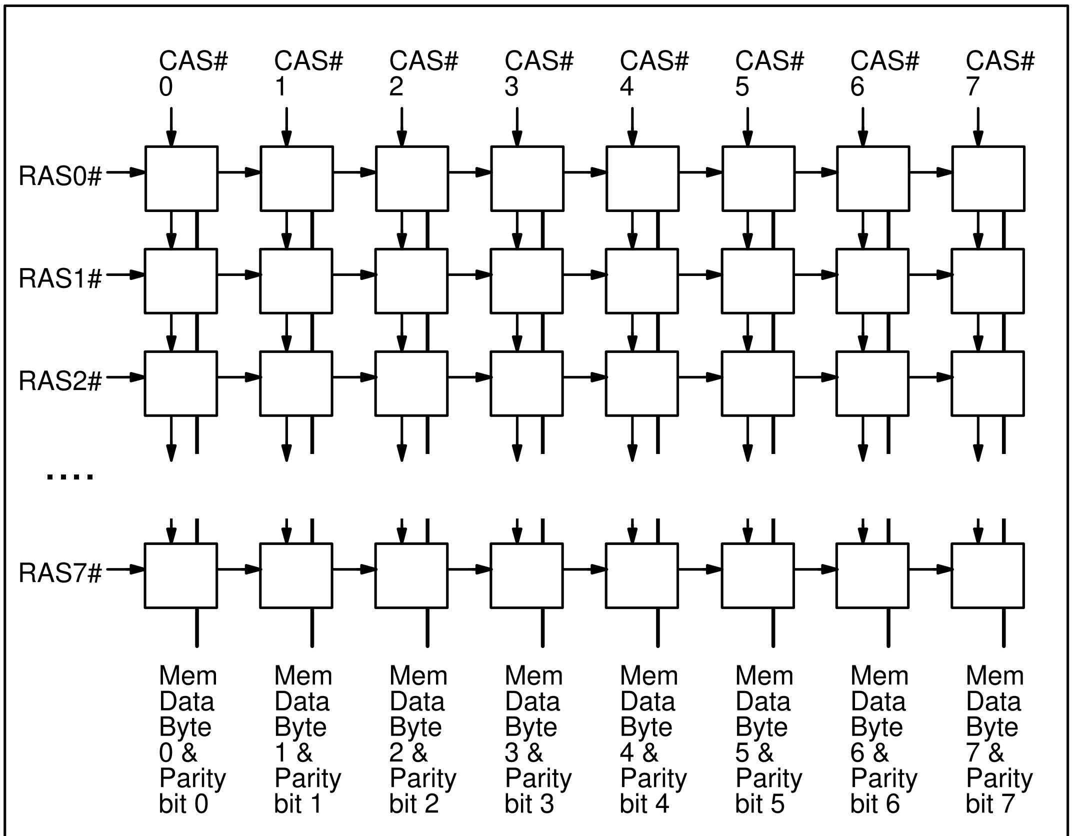


Figure 5-3. DRAM Logical Implementation

5.3.2 SIMM Topologies

Table 5-5 shows the various memory module (SIMM) topologies that the 660 supports. Each memory bank can be populated with any supported SIMM.

Table 5-5. Supported SIMM Topologies

SIMM type	Size	Depth	Width	Banks	Addressing	Addressing Mode (1)
4-Byte Wide (72-pin)	4 Meg	1M	4 bytes	1 + 1 empty	10 x 10	2
	8 Meg	1M	4 bytes	2	10 x 10	2
	16 Meg	4M	4 bytes	1 + 1 empty	11 x 11	2
	32 Meg	4M	4 bytes	2	11 x 11	2
	64 Meg	16M	4 bytes	1 + 1 empty	12 x 12	3
8-Byte Wide (168-pin)	8 Meg	1M	8 bytes	1	10 x 10	2
	16 Meg	2M	8 bytes	1	11 x 10	2
	32 Meg	4M	8 bytes	1	11 x 11 or 12 x 10	2
	64 Meg	8M	8 bytes	1	12 x 11	3
	128 Meg	16M	8 bytes	1	12 x 12	3

Note for Table 5-5:

(1) See BCR(A4 to A7) in Section 5.3.7.

The 660 supports the 168-pin 8-byte SIMMs shown in Table 5-5, which are each arranged as a single bank of 8-byte-wide DRAM. Each SIMM requires a single RAS# line. These SIMMs do not have to be installed in pairs.

The bridge also supports the 72-pin 4-byte SIMMs shown in Table 5-5, which are each arranged as two banks of 4-byte-wide DRAM, only one bank of which may be accessed at a given time. Each bank requires a RAS# line, and each bank is addressed by the same address lines. These SIMMs must be installed in pairs (of identical devices), since it is necessary to use two (72-pin) 4-byte SIMMs to construct an 8-byte-wide memory array. Since each 72-pin 4-byte SIMM consists of two banks, this pair of SIMMs also requires two RAS# lines. The 660 addresses a given SIMM based on the value of the associated memory bank addressing mode BCR.

5.3.3 Row and Column Address Generation

The 660 formats the row and column addresses presented to the DRAM based on the organization of the DRAM. In memory bank addressing mode 2, the bridge is configured to address devices that require 12x10, 11x11, 11x10, or 10x10 bit (row x column) addressing. In memory bank addressing mode 3, the bridge is configured to address devices that require 12x12 or 12x11 bit (row x column) addressing (no other addressing modes are currently available).

Table 5-6 and Table 5-7 show which CPU address bits are driven onto the memory address bus during CPU to memory transfers. Table 5-8 and Table 5-9 show which PCI_AD address bits are driven onto the memory address bus during PCI to memory transfers. These address line assignments are not affected by the endian mode of the system. The addressing mode is selected using the memory bank addressing mode BCRs (see Section 5.3.7). The addressing mode of each bank of memory is individually configurable.

5

Table 5-6. Row Addressing (CPU Addressing)

Memory Bank Addressing Mode	Addressing Mode BCR	MA 11	MA 10	MA 9	MA 8	MA 7	MA 6	MA 5	MA 4	MA 3	MA 2	MA 1	MA 0
12x10, 11x10, 10x10, 11x11	010 (Mode 2)	A 7	A 8	A 9	A 10	A 11	A 12	A 13	A 14	A 15	A 16	A 17	A 18
12x12, 12x11	011 (Mode 3)	A 7	A 8	A 9	A 10	A 11	A 12	A 13	A 14	A 15	A 16	A 17	A 18

Table 5-7. Column Addressing (CPU Addressing)

Memory Bank Addressing Mode	Addressing Mode BCR	MA 11	MA 10	MA 9	MA 8	MA 7	MA 6	MA 5	MA 4	MA 3	MA 2	MA 1	MA 0
12x10, 11x10, 10x10, 11x11	010	A 5	A 7	A 19	A 20	A 21	A 22	A 23	A 24	A 25	A 26	A 27	A 28
12x12, 12x11	011	A 5	A 6	A 19	A 20	A 21	A 22	A 23	A 24	A 25	A 26	A 27	A 28

Table 5-8. Row Addressing (PCI Addressing)

Memory Bank Addressing Mode	Addressing Mode BCR	MA 11	MA 10	MA 9	MA 8	MA 7	MA 6	MA 5	MA 4	MA 3	MA 2	MA 1	MA 0
12x10, 11x10, 10x10, 11x11	010	AD 24	AD 23	AD 22	AD 21	AD 20	AD 19	AD 18	AD 17	AD 16	AD 15	AD 14	AD 13
12x12, 12x11	011	AD 24	AD 23	AD 22	AD 21	AD 20	AD 19	AD 18	AD 17	AD 16	AD 15	AD 14	AD 13

Table 5-9. Column Addressing (PCI Addressing)

Memory Bank Addressing Mode	Addressing Mode BCR	MA 11	MA 10	MA 9	MA 8	MA 7	MA 6	MA 5	MA 4	MA 3	MA 2	MA 1	MA 0
12x10, 11x10, 10x10, 11x11	010	AD 26	AD 24	AD 12	AD 11	AD 10	AD 9	AD 8	AD 7	AD 6	AD 5	AD 4	AD 3
12x12, 12x11	011	AD 26	AD 25	AD 12	AD 11	AD 10	AD 9	AD 8	AD 7	AD 6	AD 5	AD 4	AD 3

In the case of 10x10 addressing, MA[9:0] are connected to the DRAM modules. In the case of 11x10 or 11x11, connect MA[10:0], and in the case of 12x11 or 12x12, connect MA[11:0] to the DRAM modules.

5.3.4 DRAM Pages

The 660 uses an 8K page size for DRAM page-mode determination.

5.3.5 Supported Transfer Sizes and Alignments

The 660 supports all CPU to memory transfer sizes and alignments that do not cross an 8-byte boundary.

5.3.6 Unpopulated Memory Locations

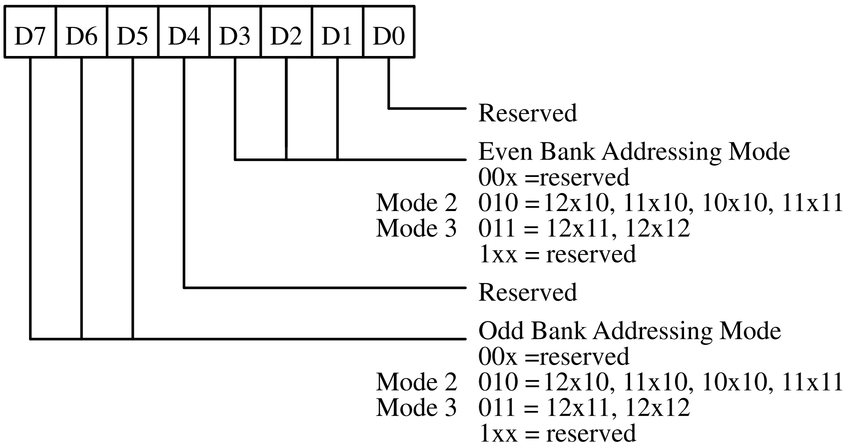
Physical memory does not occupy the entire address space assigned to system memory in the memory map. When the CPU reads an unpopulated location while memory select error is disabled, the bridge returns all-ones and completes the transfer normally. When the CPU writes to an unpopulated location, the bridge signals normal transfer completion to the CPU, but does not write the data to memory. The memory select error bit in the error status 1 register (index C1h) is set in both cases. Gaps are not allowed in the DRAM memory space, but empty (size=0) memory banks are allowed.

While memory select error enable = 0, reads and writes to unpopulated memory locations are not flagged as errors. The 660 completes them normally. Note that reading unpopulated memory locations can produce ECC errors while ECC is enabled. This can also happen during less than 8-bite writes to unpopulated memory locations in ECC mode.

5.3.7 Memory Bank Addressing Mode BCRs

Index A4 to A7	Read/Write	Reset to 44h (each BCR)

This array of four 8-bit, read/write BCRs defines the format of the row and column addressing of each DRAM memory bank.



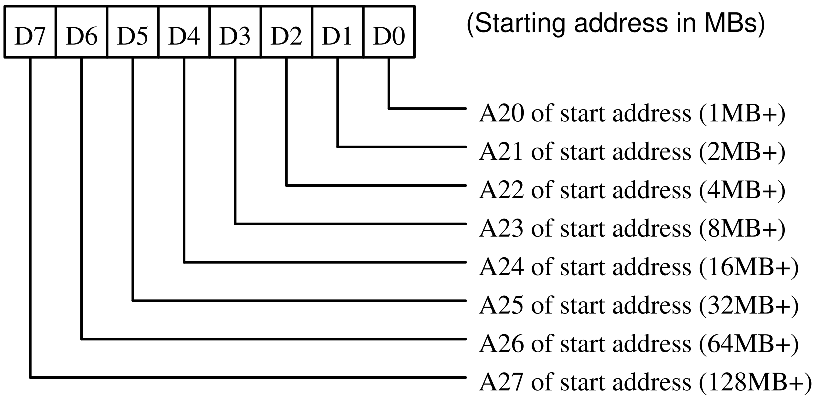
Register	Bits	Memory Bank	Bits	Memory Bank
A4h	3:0	0	7:4	1
A5h	3:0	2	7:4	3
A6h	3:0	4	7:4	5
A7h	3:0	6	7:4	7

5.3.8 Memory Bank Starting Address BCRs

Index 80 to 87h	Read/Write	Reset to 00h (each BCR)

This array of eight BCRs (along with the eight extended starting address registers) contains the starting address for each memory bank. Each pair of registers maps to the corresponding RAS# decode. For example, RAS[4]# corresponds to the BCRs at index 84h and 8Ch. The eight least-significant bits of the bank starting address are contained in the starting address register, and the most-significant bits come from the corresponding extended starting address register. The starting address of the bank is entered with the least significant 20 bits truncated. These BCRs must be programmed in conjunction with the ending address and extended ending address registers.

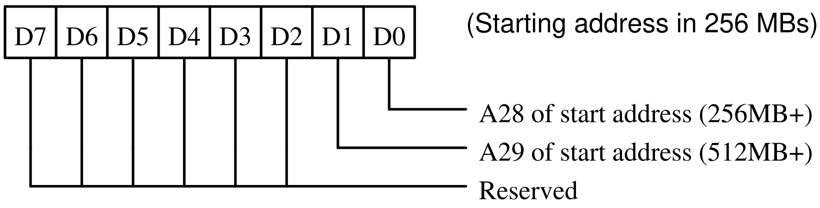
Program the banks in ascending order, such that (for n = 0 to 6) the starting address of bank n+1 is higher than the starting address of bank n. Each bank must be located in the 0 to 1G address range. See section 5.3.13.



5.3.9 Memory Bank Extended Starting Address BCRs

Index 88 to 8F	Read/Write	Reset to 00h (each BCR)

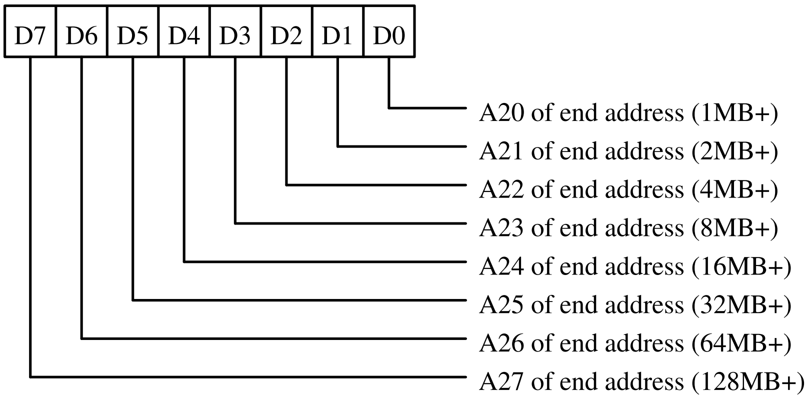
This array of eight BCRs (along with the eight starting address registers) contains the starting address for each memory bank. These BCRs contain the most-significant address bits of the starting address of the corresponding bank.



5.3.10 Memory Bank Ending Address BCRs

Index 90 to 97h	Read/Write	Reset to 00h (each BCR)

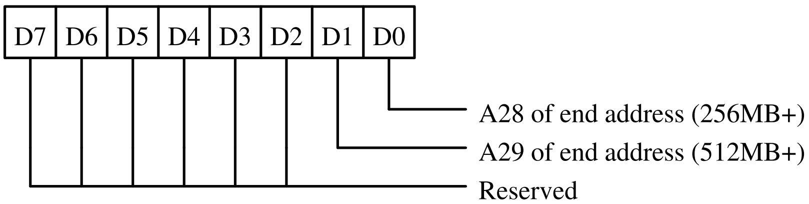
This array of eight BCRs (along with the eight extended starting address registers) contains the ending address for each memory bank. Each pair of registers maps to the corresponding RAS# decode. For example, RAS[4]# corresponds to the BCRs at index 94h and 9Ch. The eight least-significant bits of the bank ending address are contained in the ending address register, and the most-significant bits come from the corresponding extended ending address register. The ending address of the bank is entered as the address of the next highest memory location minus 1, with the least significant 20 bits truncated. For an xMB bank program, the end address is equal to the start address (from Section 5.3.9) + x–1. Each bank must be located in the 0 to 1G address range. These BCRs must be programmed in conjunction with the ending address and extended ending address registers. See section 5.3.13.



5.3.11 Memory Bank Extended Ending Address BCR

Index 98 to 9F	Read/Write	Reset to 00 (each BCR)

This array of eight 8-bit, read/write registers (along with the eight ending address registers) contains the ending address for each memory bank. These BCRs contain the most-significant address bits of the ending address of its bank.

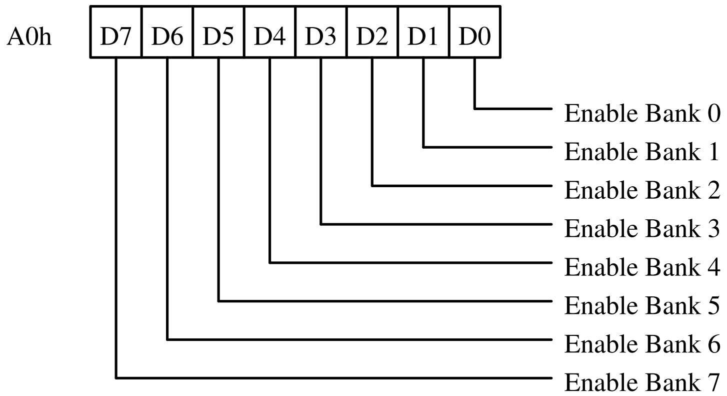


5.3.12 Memory Bank Enable BCR

Index A0	Read/Write	Reset to 00h

This BCR contains a control enable for each bank of memory. Each bank of memory must be enabled for proper refreshing. For each bit, a 0 disables that bank of memory and a 1 enables it.

This register must be programmed in conjunction with the starting address and ending address registers. If a bank is disabled by this register, the corresponding starting and ending address register entries become don't cares.



5.3.13 Memory Bank Configuration Example

In the example memory bank configuration shown in Figure 5-4, the eight memory banks are populated by different size and organization devices. For convenience, this example shows the bridge configured to address each memory bank in order with no gaps in the populated address range but this is not required. Any bank can be placed in any (1MB aligned) non-populated address range from 0 to 1G.

Table 5-10. Example Memory Bank Addressing Mode Configuration

Bank	SIMM Type	SIMM Depth	SIMMs Per Bank	SIMM Bank Topology	SIMM Size	Row x Col	BCR ()	Bits ()	Mode
0	8-Byte	1M	1	8B x 1M x 1 bank	8M	10 x 10	A4	3:1	2
1	4-Byte	4M	2	4B x 4M x 2 bank	32M	11 x 11	A4	7:5	2
2	4-Byte	4M	2	4B x 4M x 2 bank	32M	11 x 11	A5	3:1	2
3	8-Byte	8M	1	8B x 8M x 1 bank	64M	12 x 11	A5	7:5	3
4	none	—	—	—	0M	—	A6	3:1	—
5	4-Byte	4M	2	4B x 4M x 2 bank	32M	11 x 11	A6	7:5	2
6	8-Byte	16M	1	8B x 16M x 1 bank	128M	12 x 12	A7	3:1	3
7	8-Byte	4M	1	8B x 4M x 1 bank	32M	11 x 11 12 x 10	A7	7:5	2

5.3.13.1 Memory Bank Enable BCR

Program indexed BCR A0h (memory bank enable) to EFh to enable all banks except # 4.

5.3.13.2 Memory Bank Addressing Mode

As shown in Table 5-10, memory bank 0 (connected to RAS0#) contains an 8-byte x 1M SIMM (8M) with one bank (one RAS# line). It is addressed with 10 row and 10 column bits. Program bits 3:1 of indexed BCR A4h with 010b (mode 2).

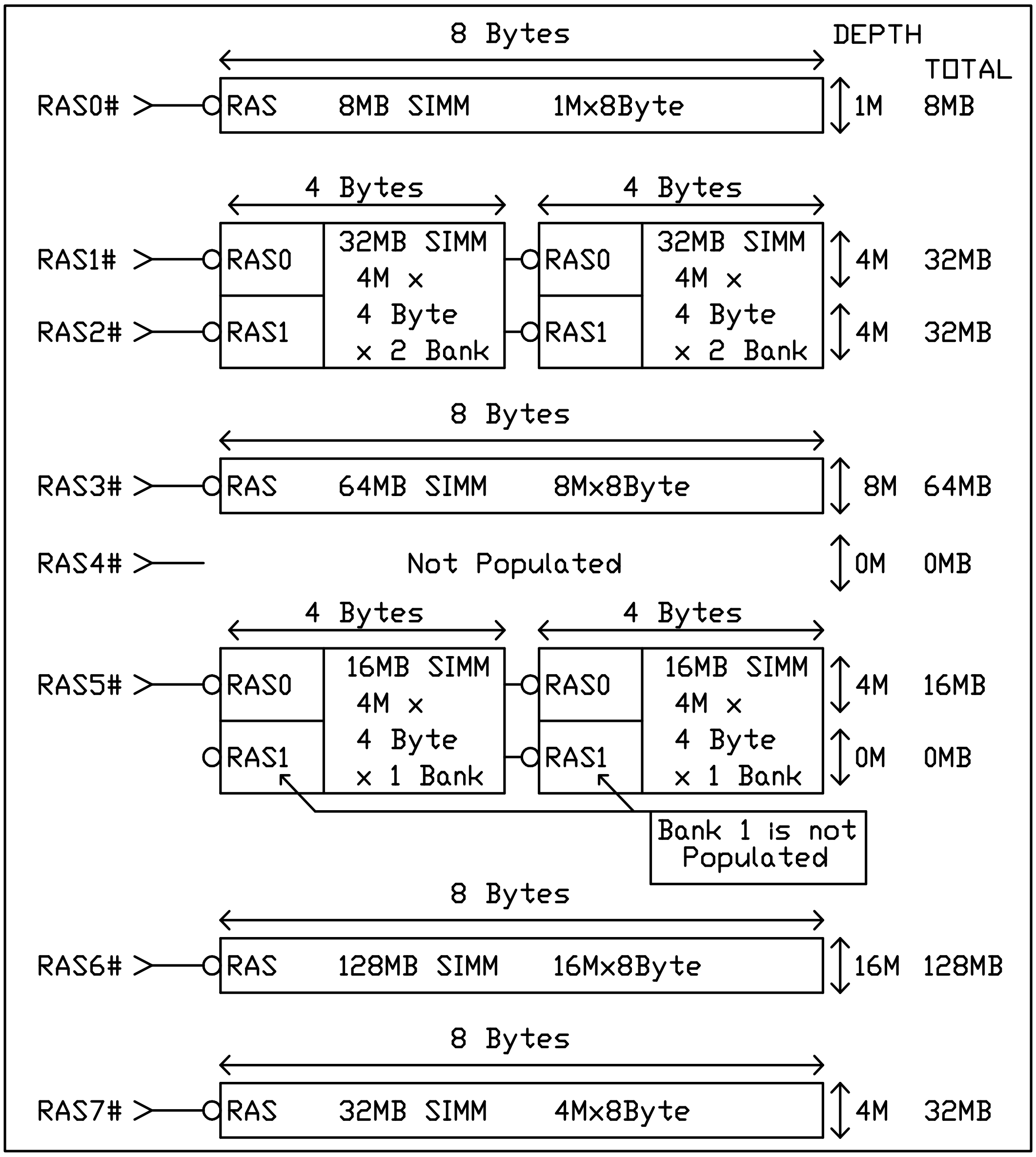


Figure 5-4. Example Memory Bank Configuration

5.3.13.3 Starting and Ending Addresses

As shown in Table 5-11, program indexed BCR 80h with 00h to configure address bits 27:20 of the bank 0 starting address. Program indexed BCR 88h with 00h to configure address bits 29:28. Also program indexed BCR 90h with 00h to configure address bits 27:20 of the bank 0 ending address, and program indexed BCR 98h with 07h to configure address bits 29:28.

The next two physical SIMM units are 4-byte x 4M x 2 bank (16M x 2 bank = 32M) SIMMs, used side by side to achieve 8-byte width. They form banks 1 and 2, each of which is 8-byte x 4M (32M). Note that bank 7 is also 32M, populated by an 8-byte x 8M (32M) SIMM. Using the same configuration, banks 1 and 2 could also be implemented using two 8-byte x 4M (32M) SIMMs. Bank 3 is configured to the same size as bank 1 plus bank 2 and is implemented using a single 8-byte x 8M (64M) SIMM.

Bank 4 is not populated, so set memory bank enable BCR bit D4 to 0. The data in the starting, extended starting, ending, and extended ending address BCRs for bank 4 is ignored. Note that empty memory banks are allowed, but that gaps in the DRAM space are not allowed.

For proper operation, program the bridge to execute memory accesses at the speed of the slowest device. If ECC or parity is selected, all devices must support the capability. And if the bridge is programmed to utilize hyper-page mode, all devices must support extended-data out transfers.

Table 5-11. Example Memory Bank Starting and Ending Address Configuration

	SIMM	Starting	Extended		Base		Ending	Extended		Base	
Bank	**Size**	**Address**	**BCR**	**Data**	**BCR**	**Data**	**Address**	**BCR**	**Data**	**BCR**	**Data**
0	8M	0000 0000	88	00	80	00	007F FFFF	98	00	90	07
1	32M	0080 0000	89	00	81	08	027F FFFF	99	00	91	27
2	32M	0280 0000	8A	00	82	28	047F FFFF	9A	00	92	47
3	64M	0480 0000	8B	00	83	48	087F FFFF	9B	00	93	87
4	0M	Don’t Care	8C	xx	84	xx	Don’t Care	9C	xx	94	xx
5	32M	0880 0000	8D	00	85	88	0A7F FFFF	9D	00	95	A7
6	128M	0A80 0000	8E	00	86	A8	127F FFFF	9E	01	96	27
7	32M	1280 0000	8F	01	87	28	147F FFFF	9F	01	97	47

5.4 Error Checking and Correction

The 660 provides three levels of memory error checking—no checking, parity checking, and error checking and correction (ECC). If no memory checking is enabled, the system can be configured to use lower-cost, non-parity DRAM.

While the system is configured for memory parity checking, the 660 performs as follows:

- Uses odd parity checking
- Detects all single bit errors
- Allows full-speed memory accesses

While the system is configured for ECC, the 660 performs as follows:

- Uses an H–matrix and syndrome ECC protocol
- Uses the same memory devices and connectivity as for parity checking
- Detects and corrects all single-bit errors
- Detects all two-bit errors
- The first beat of CPU to memory reads requires one additional CPU clock cycle
- CPU to memory burst writes and 8-byte single-beat writes are full-speed
- CPU to memory single-beat writes of less than eight bytes are implemented as read-modify-write (RMW) cycles
- PCI to memory reads are full-speed
- PCI to memory writes are full-speed while data can be gathered into 8-byte groups before being written to memory. Single beat or ungatherable writes require a read-modify-write cycle

5.4.1 Memory Parity

While the 660 memory controller is configured for memory parity checking, the bridge implements an odd parity generation and checking protocol, generating parity on memory writes and checking parity on memory reads. One parity bit is associated with each data byte and is accessed with it. When a parity error is detected during CPU to memory reads, the error is reported by means of TEA# or MCP#. When a parity error is detected during PCI to memory reads, the error is reported by means of PCI_SERR#. Memory parity checking requires one bit of parity DRAM per one byte of data DRAM.

The 660 detects all single-bit parity errors, but may not detect multi-bit parity errors. Also, an even number of parity errors will not be detected. For example, an event which causes a parity error in bytes 0, 1, 2, 3, 4, and 5 will not be detected.

5.4.2 ECC Overview

While ECC is enabled, the 660 uses the ECC logic to detect and correct errors in the transmission and storage of system memory data. The bridge implements the ECC protocol using the same connectivity and memory devices as parity checking.

While neither parity or ECC is enabled, the bridge executes memory writes of less than eight bytes in response to bus master requests to write less than eight bytes of data. The bridge always (whether parity, ECC or neither is enabled) reads data from memory in 8-byte groups, even if the bus master is requesting less than eight bytes.

While parity is enabled, the bridge also executes memory writes of less than eight bytes in response to bus master requests to write less than eight bytes of data, since writing a byte to memory also updates the associated parity bit. During memory writes, the bridge generates one parity bit for each byte of data and stores it with that byte of data. This parity bit is a function only of the data byte with which it is associated. During memory reads, the integrity of the data is parity checked by comparing each data byte with its associated parity bit.

While in ECC mode, the 660 can use either parity DRAM or nine byte ”ECC DRAM,” which is not byte addressable. **Note:** Ensure that the DRAM controller is in ECC mode <u>before</u> enabling the controller with nine byte ECC DRAM.

However, when ECC is enabled the bridge reads from and writes to system memory only in 9-byte groups (as a 72-bit entity), even though the bus master may be executing a less-than-8-byte read or write. There is one byte of ECC check byte information for eight bytes of data. During memory writes, the eight check bits are generated as a function of the 64 data bits as a whole, and the check bits are stored with the data bits as a 72-bit entity. During memory reads, all 72 bits are read, and the eight check bits are compared with the 64 data bits as a whole to determine the integrity of the entire 72-bit entity.

In ECC mode, single-bit errors are corrected. When a multi-bit error is detected during CPU to memory reads, the error is reported by means of TEA# or MCP#. When a multi-bit ECC error is detected during PCI to memory reads, the error is reported by means of PCI_SERR#. Note that the DRTRY# function of the CPU is not used, even when the 660 is in ECC mode (the Bridge will not assert DRTRY#).

Only the data returned to the CPU (or PCI) is corrected for single-bit errors. The corrected data is not written into the DRAM location.

5.4.3 ECC Data Flows

While ECC is enabled, the 660 always reads eight data bytes and one check byte from memory during CPU and PCI reads. During bus master 8-byte writes to memory, the bridge writes eight data bytes and one check byte to memory. When the bus master writes less than eight data bytes to memory, it is possible for each check bit to change due to a write to any one of the eight data bytes. The Bridge then executes a read-modify-write (RMW) cycle—reading all eight data bytes from memory, modifying the appropriate bytes with the new data, recalculating all of the check bits, and writing the new data and check bits to memory.

5.4.3.1 Memory Reads

Figure 5-5 shows a simplified data flow in a 660 system during CPU to memory read transfers. Figure 5-6 shows the simplified data flow during PCI to memory reads. The data and check bits flow from system memory into the bridge where the checking logic combines the data with the check bits to generate the syndrome. If there is a single-bit error in the data, the correction logic corrects the data and supplies it to the requesting agent. If there is a multiple-bit error in the data, the bridge signals an error to the requesting agent.

Note that the structure of the bridge as shown in Figure 5-5 through Figure 5-9 is considerably simplified and does not show the posted write buffers and other internal details.

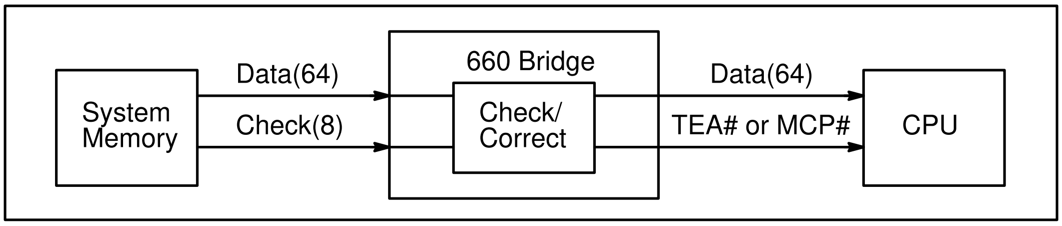


Figure 5-5. CPU Read Data Flow

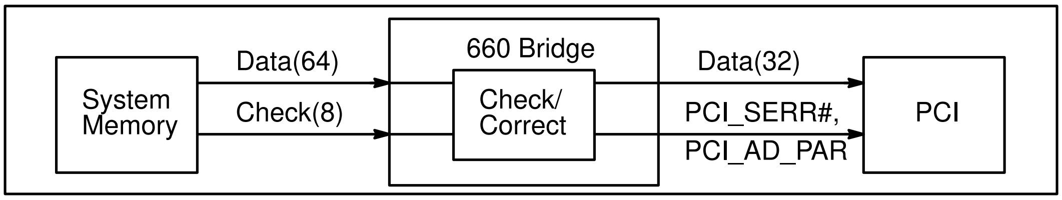


Figure 5-6. PCI Read Data Flow

5.4.3.2 Eight-Byte Writes

Figure 5-7 shows the simplified data flow in a 660 system during 8-byte CPU to memory writes. The data flows from the CPU into the bridge and out onto the memory data bus. The bridge generates the check bits based on the eight bytes of data, and stores them in memory with the data.

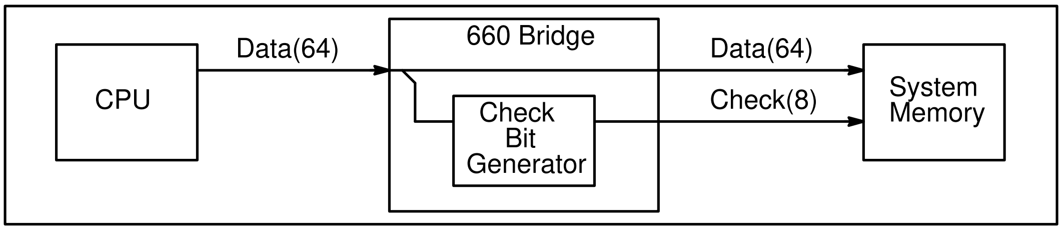


Figure 5-7. CPU 8-Byte Write Data Flow

Figure 5-8 shows the simplified data flow in a 660 based system during gathered PCI to memory 8-byte writes. During the first of the two gathered data phases (A), the data flows from the PCI bus into a 4-byte hold latch in the bridge. On the next data phase (B), the next 4–bytes of PCI data flows into the bridge, where it is combined with the data from the previous data phase. The 8-byte data then flows onto the memory data bus. The bridge generates the check bits based on the 8-byte data, and stores them in memory with the data.

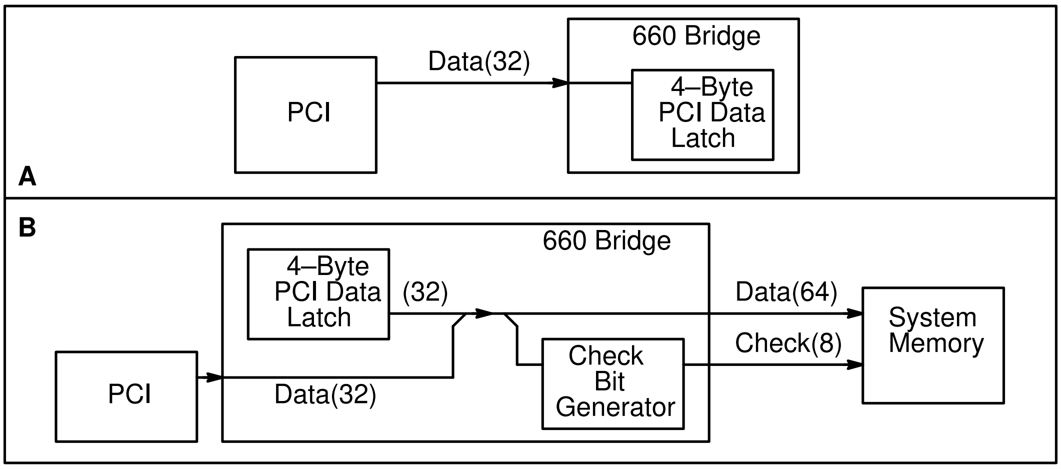


Figure 5-8. PCI 8-Byte Write Data Flow

Note that if either or both of the two gathered PCI data phases is a write of less than 4 bytes, the two data phases will still be gathered, and then the bridge will execute a RMW cycle, filling in the unwritten bytes (in the group of 8 bytes) with data from those locations in memory. The same write case while ECC is disabled does not cause a RMW cycle; the bridge merely writes only the indicated bytes, leaving the write enables of the unaccessed bytes deasserted.

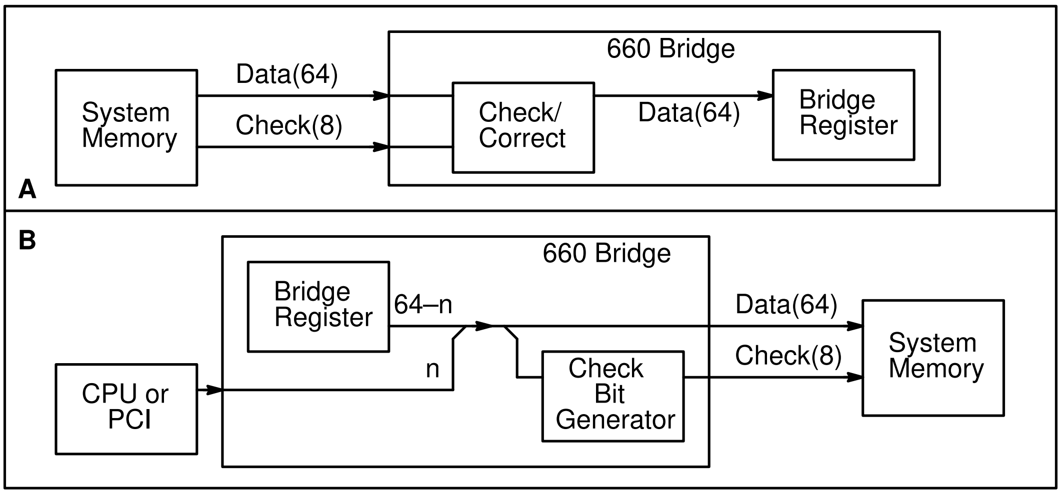


Figure 5-9. PCI or CPU Read-Modify-Write Data Flow

5.4.3.3 Less-Than Eight-Byte Writes

Figure 5-9 shows the simplified data flow during a CPU or PCI bus master to memory write of less than eight bytes, during which the bridge executes a RMW cycle. In Figure 5-9(A), the bridge reads in the data and check bits from the addressed memory locations and places this data (corrected as necessary) in a register. In Figure 5-9(B), this data is modified by the bridge, which replaces the appropriate memory data bytes with write data from the bus master. The bridge then recomputes all of the check bits and writes the new data and check bits to memory.

5.4.4 Memory Performance In ECC Mode

Enabling ECC mode on the 660 affects memory performance in various ways, depending on the transaction type. The effect is the same for both EDO and page mode DRAMs.

5.4.4.1 CPU to Memory Read in ECC Mode

ECC mode adds one CPU_CLK to single–beat CPU to memory read transfers and to the first beat of CPU to memory burst read transfers. The other beats of the burst are unaffected. During the extra CPU_CLK, the 663 holds the data while checking (and if necessary, correcting) it.

The 660 does not use the DRTRY# function, even while in ECC mode. In no–DRTRY# mode, during memory reads the 604 uses the data bus data internally as soon as it samples TA# active. The 660 supports this mode by presenting correct(ed) data to the before driving TA# valid. The Bridge does not speculatively present data to the CPU (using TA#) and then assert DRTRY# if there is a data error.

By allowing the 604 to run in no–DRTRY# node, the 660 enables the 604 to use data from the L2 cache at the full speed of the L2 cache, without requiring the 604 to insert an additional (internal to the 604) one CPU clock delay on reads.

In DRTRY# mode, during memory reads the 604 holds the data internally for one CPU clock after sampling TA# active. This delay is inserted by the 604 to allow DRTRY# to be sampled before the data is used. Thus during CPU to memory reads in ECC mode while a 604 is in DRTRY# mode, the 660 inserts a one CPU clock delay to check/correct the data, and then the 604 adds a one CPU clock delay to check for DRTRY#. Thus two CPU clocks are added to single–beat CPU to memory read transfers and to the first beat of CPU to memory burst read transfers.

5.4.4.2 CPU to Memory Write in ECC Mode

ECC mode adds no additional clock cycles to 8-byte CPU to memory write transfers. Note that all CPU bursts are composed of 8-byte beats. CPU to memory writes of less than eight bytes are handled as RMW cycles, which usually require four additional CPU clocks as compared to 8-byte writes.

5.4.4.3 PCI to Memory Read in ECC Mode

ECC mode adds no additional clock cycles to PCI to memory read transactions.

5.4.4.4 PCI to Memory Write in ECC Mode

ECC mode has a complex effect on PCI to memory writes. During PCI to memory writes, the bridge attempts to gather adjacent 4-byte data phases into 8-byte memory writes. In response to conditions during a given data phase, the bridge either gathers, writes 8 bytes, or read-modify-writes, as shown in Table 5-12. Gather and 8-byte write operations incur no performance penalties, but RMW cycles add from two to four (usually three) PCI clock cycles to the transaction time. The consequences of ECC mode delays on PCI to memory write bursts are minor and are best understood from a few examples. The following examples assume page hits and no snoop hits.

Table 5-12. Bridge Response to Various PCI Write Data Phases

Bridge Operation	Conditions	Description of Operation
Gather	This data phase is not the last data phase, and This is a 4-byte transfer (BE[3:0]#=0000), and This data phase is to the lower 4-byte word.	The bridge latches the four bytes from this data phase into the low four bytes of a hold register.
Eight-Byte Write	The previous data phase caused a gather, and This is a 4-byte transfer, and This data phase is to the upper 4-byte word.	The bridge combines the (high) four bytes from this data phase with the (low) four bytes from the previous data phase, and writes all eight bytes to memory.
Read-Modify-Write	This is a single phase transaction, or This is the first data phase and is to the upper 4-byte word, or This is the last data phase and is to the lower 4-byte word, or This is a less-than-4-byte transfer.	The bridge Reads eight bytes from memory, modifies the data by replacing the appropriate four bytes with the data from this data phase, and then writes all eight bytes to memory.

Best case (see Table 5-13) is a burst starting on a low word (memory address is 0 mod 8), composed of an even number of data phases, in which all data phases transfer four bytes of data. Notice that the first data phase is gathered and the second data phase causes an 8-byte memory write. The following pairs of data beats follow the same pattern. No page misses, snoop hits, or data beats of less than 4 bytes are encountered. This case adds no PCI clock cycles to the best-case transaction time.

Table 5-13. Bridge Response to Best Case PCI Write Burst

Data Phase	Word	Bridge Operation	Special Conditions	Performance Impact
n (last)	High	8-byte Write	None	None
n –1	Low	Gather	None	None
...	...	...	...	...
4	High	8-byte Write	None	None
3	Low	Gather	None	None
2	High	8-byte Write	None	None
1 (first)	Low	Setup + Gather	None	None

Table 5-14 shows a case where the first data phase is to a high word (memory address is 4 mod 8), and is composed of an odd number of data phases in which all data phases transfer four bytes of data. Here the total effect is to add three PCI clocks to the transaction time, which is shown during the first data phase in Table 5-14.

Table 5-14. Bridge Response to Case 2 PCI Write Burst

Data Phase	Word	Bridge Operation	Special Conditions	Performance Impact
n (last)	High	8-byte Write	None	None
n –1	Low	Gather	None	None
...	...	...	...	...
3	High	8-byte Write	None	None
2	Low	Gather	None	None
1 (first)	High	Setup + RMW	None	Add 3 PCI clocks

Table 5-15. Bridge Response to Various PCI Write Bursts

Row	Data Phase	Word	Bridge Operation	Special Conditions	Performance Impact
12	n (last)	Low	RMW	None	Add 3 PCI clocks
11	n – 1	High	8-byte Write	None	None
10	n – 2	Low	Gather	None	None
9	...	...	...	...	None
8	4	High	RMW	None	Add 3 PCI clocks
7	3	Low	RMW	Less than 4-byte transfer	Add 3 PCI clocks
6	...	...	...	...	None
5	10	High	RMW	Less than 4-byte transfer	Add 3 PCI clocks
4	9	Low	Gather	None	None
3	...	...	...	...	None
2	2	High	8-byte Write	None	None
1	1 (first)	Low	Setup + Gather	None	None

Table 5-15 shows the effect of several conditions on the transaction time. Rows 1 through 6 show the effects of a less-than-4-byte transfer at the high word location that occurs during a burst. The term *None* in the column titled *Performance Impact* in row 6 indicates that there are no residual performance penalties due to the events of rows 1 through 5.

Rows 7 through 9 show the effect of a less-than-4-byte transfer at the low word location that occurs during a burst. The term None in the column titled Performance Impact in row 9 indicates that there are no residual performance penalties due to the events of rows 7 and 8.

Rows 10 through 12 show the effect of a burst that ends at a low word location. The total performance impact from this burst is to add 12 PCI clocks to the transaction time.

Note that the performance penalty for single data phase PCI writes is three additional PCI clocks, whether the destination is the high word or the low word.

5.4.5 Check Bit Calculation

The 660 generates the check bits based on Table 5-16 (which is shown using little-endian bit numbering).

Table 5-16. Check Bit Calculation

CB (x)	Data Bits. CB(x) = XOR of Data Bits (0 is LSb)
0	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,33,34,35,39,41,42,43,47,49,50,51,55,57,58,59,63
1	8,9,10,11,12,13,14,15,24,25,26,27,28,29,30,31,32,34,35,38,40,42,43,46,48,50,51,54,56,58,59,62
2	16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,35,37,40,41,43,45,48,49,51,53,56,57,59,61
3	0,1,2,3,4,5,6,7,16,17,18,19,20,21,22,23,32,33,34,36,40,41,42,44,48,49,50,52,56,57,58,60
4	1,2,3,7,9,10,11,15,17,18,19,23,25,26,27,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47
5	0,2,3,6,8,10,11,14,16,18,19,22,24,26,27,30,40,41,42,43,44,45,46,47,56,57,58,59,60,61,62,63
6	0,1,3,5,8,9,11,13,16,17,19,21,24,25,27,29,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63
7	0,1,2,4,8,9,10,12,16,17,18,20,24,25,26,28,32,33,34,35,36,37,38,39,48,49,50,51,52,53,54,55

5.4.6 Syndrome Decode

The syndrome consists of an 8-bit quantity which is generated by the 660 as it is comparing the 8 data bytes to the check byte. This syndrome contains the results of the comparison, as shown in Table 5-17, where:

- **`ne`** means that no error has been detected.
- **`cbx`** means that check bit x is inverted (a single-bit error).
- **`dx`** means that data bit x is inverted (a single-bit error).
- **`blank`** means that a multiple bit error has occurred.

Based on the information in the syndrome, the 660 corrects all single-bit errors and signals an error to the requesting agent on multiple-bit errors.

Table 5-17. Syndrome Decode

S0 S1 S2 S3 S7 S6 S5 S4	0 0 0 0	1 0 0 0	0 1 0 0	1 1 0 0	0 0 1 0	1 0 1 0	0 1 1 0	1 1 1 0	0 0 0 1	1 0 0 1	0 1 0 1	1 1 0 1	0 0 1 1	1 0 1 1	0 1 1 1	1 1 1 1
0 0 0 0	ne	cb0	cb1		cb2				cb3							
0 0 0 1	cb 4			d8			d24			d0			d16			
0 0 1 0	cb 5			d9			d25			d1			d17			
0 0 1 1		d40	d41		d42			d44	d43			d45		d46	d47	
0 1 0 0	cb 6			d10			d26			d2			d18			
0 1 0 1																
0 1 1 0		d56	d57		d58			d60	d59			d61		d62	d63	
0 1 1 1				d12			d28			d4			d20			
1 0 0 0	cb 7			d11			d27			d3			d19			
1 0 0 1		d32	d33		d34			d36	d35			d37		d38	d39	
1 0 1 0																
1 0 1 1				d13			d29			d5			d21			
1 1 0 0		d48	d49		d50			d52	d51			d53		d54	d55	
1 1 0 1				d14			d30			d6			d22			
1 1 1 0				d15			d31			d7			d23			
1 1 1 1																

5.5 DRAM Refresh

The memory controller provides DRAM refresh logic for system memory. The memory controller supports CAS#-before-RAS# refresh only, which provides lower power consumption and lower noise generation than RAS#-only refresh. In this refresh mode, MA[11:0] are not required. Refresh of the odd banks of memory is staggered from the refresh of the even banks of memory to further reduce noise (see Figure 5-10).

During refresh,

- WE[1:0] are driven high.
- MEM_DATA[63:0] are tri-stated.
- MA[11:0] continue to be driven to their previous state.

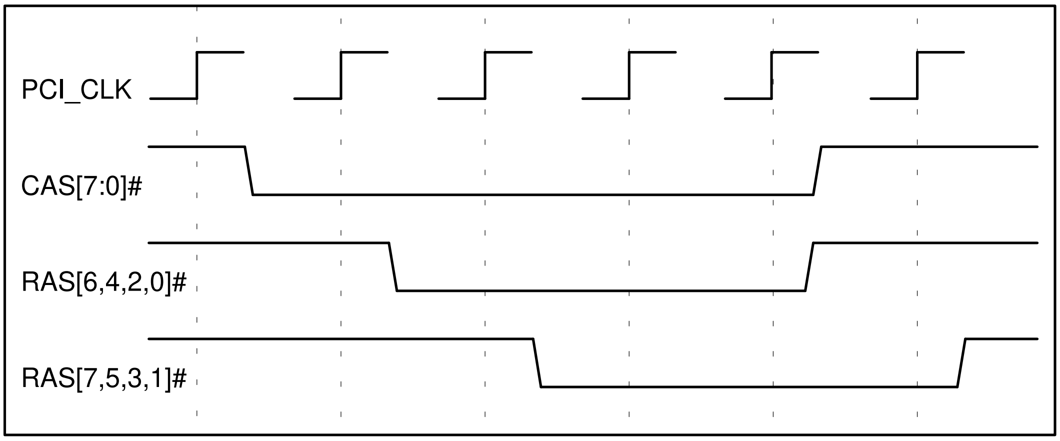


Figure 5-10. DRAM Refresh Timing Diagram

Refresh requests are generated internally by dividing down the PCI_CLK. The divisor value is programmed in the refresh timer divisor register. This register is initialized to 504, a value that provides a refresh rate of 15.1us when PCI_CLK rate is 33MHz. For other PCI_CLK frequencies, the refresh rate register must be properly configured before accessing system memory.

Refresh continues to occur even if CPU_CLK is stopped.

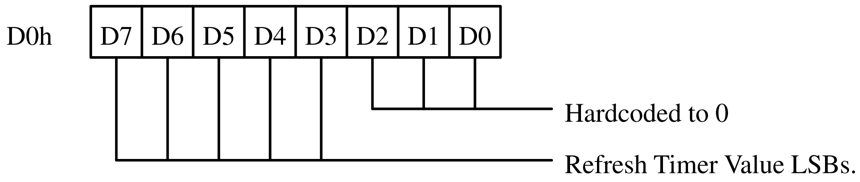
The first refresh occurs immediately upon deassertion of RESET#.

5

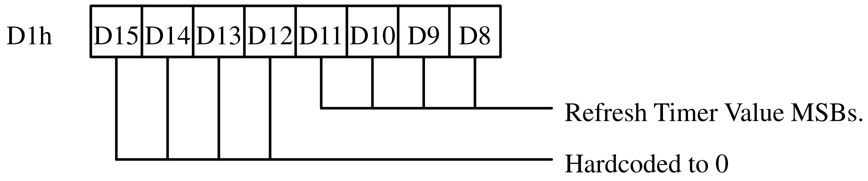
5.5.1 Refresh Timer Divisor Register

Index D0 to D1	Read/Write	Reset to F8 (D0) and 01 (D1)

The refresh timer register is a 16-bit BCR that determines the memory refresh rate. Typical refresh rates are 15.1 to 15.5 microseconds. If all DRAM in the system supports extended (slow) refresh, the refresh rate can be slower. The refresh timer is clocked by the PCI clock input to the 664 Controller. The reset value of 01F8h provides a refresh rate of 15.1 microseconds while the PCI clock is 33MHz. (01F8h equals 504 times 30ns equals 15.12us.) Bits 3–11 of the timer allow timer values from 8 to 4096.



Bits 7:3 Refresh timer (7:3) : These are the five least-significant bits of the refresh timer value.



Bits 11:8 Refresh timer (11:8) : These are the four most-significant bits of the refresh timer value.

5.6 Atomic Memory Transfers

The 660 supports atomic memory transfers by supporting the CPU reservation protocol and the PCI lock protocol.

5.6.1 Memory Locks and Reservations

The 660 supports the lwarx and stwcx atomic memory update protocol by broadcasting snoop cycles to the CPU bus during PCI to memory transactions. The bridge does not otherwise take any action, nor does it enforce an external locking protocol for CPU bus masters. See Section 5.6.1.1.

PCI bus masters can lock and unlock a 32-byte block of system memory in compliance with the PCI specification. This block can be located anywhere within the populated system memory space, aligned on a 32-byte boundary. Only a single lock may be in existence at any given time. The bridge does not implement complete bus locking. See Section 5.6.1.2.

5.6.1.1 CPU Reservation

The CPU indicates a reservation request by executing a memory read atomic (TT[0:3]=1101). If there is no PCI lock on the addressed block of memory, the bridge allows the transfer, but takes no other action. If there is a PCI lock on that block, the bridge terminates the CPU transfer with ARTRY#, does not access the memory location, and takes no other action.

The CPU removes a reservation by executing a memory read with intent to modify atomic (TT[0:3]=1111) or a memory write with flush atomic (TT[0:3]=1001). The bridge treats these accesses as a normal memory transfers.

5.6.1.2 PCI Lock

The bridge responds to the PCI lock request protocol in compliance with the PCI specification. If an agent requests a lock, and no PCI lock is in effect, the lock is granted. Once a PCI lock is granted, no other PCI locks are granted until the current lock is released.

The 660 prevents CPU bus masters and other PCI bus masters from reading or writing within a block of memory which is locked by a PCI bus master. CPU bus master accesses to a locked block are retried with ARTRY#. PCI bus master accesses to a locked block are retried with the PCI bus retry protocol. PCI and CPU bus master accesses to other areas of system memory are unrestricted.

PCI to memory transactions cause the bridge to broadcast a snoop cycle to the CPU bus. When a PCI agent is granted a memory block lock, the bridge broadcasts a write with flush (TT[0:3]=0001) cycle on the CPU bus, which causes the L1 and L2 caches to invalidate that sector (if there is an address match). This insures that there will not be a cache hit during a CPU bus accesses to a memory block which is locked by a PCI agent.

5.6.1.3 PCI Lock Release

The bridge responds to the PCI lock release protocol in compliance with the PCI Specification. If an agent releases the lock that it owned, the bridge releases the lock. The bridge generates a normal snoop cycle on the CPU bus.

5.7 DRAM Module Loading Considerations

The 660 directly drives up to eight 168 pin DRAM modules, which typically exhibit an input capacitance of less than 20pf. Table 5-18 shows some maximum input capacitance numbers that are typical of the various DRAM modules on the market. System designers may wish to buffer MA[11:0] (and perhaps WE#) if the system design requires the use of more than two banks of 72 pin SIMMs (more than one pair of 2-sided 72 pin SIMMs).

Table 5-18. Typical DRAM Module Maximum Input Capacitance

SIMM Type	SIMM Size	Maximum Input Capacitance	
		Address	WE#
72-pin	4 Meg	50pf	50pf
	8 Meg	90pf	95pf
	16 Meg	80pf	95pf
	32 Meg	160pf	190pf
168-pin	8M, 16M, 32M,128M	13pf	13pf
	64M	18pf	18pf

5.8 Related Bridge Control Registers

Bridge Control Register	Index	R/W	Bytes	See
Memory Parity Error Status	8000 0840	R	1	10.2.2.5
Single-Bit Error Counter	Index B8	R/W	1	10.3.33
Single-Bit Error Trigger Level	Index B9	R/W	1	10.3.34
Bridge Options 2	Index BB	R/W	1	10.3.36
Error Enable 1	Index C0	R/W	1	10.3.37
Error Status 1	Index C1	R/W	1	10.3.38
Single-Bit ECC Error Address	Indx CC – CF	R/W	4	10.3.44
Bridge Chip Set Options 3	Index D4	R/W	1	10.3.46

6

Section 6
L2 Cache

The L2 cache controller in the 660 controls external tag RAM and data SRAMs for a second level cache. The L2 caches as much of the system memory space from 0 to 2G as is marked as populated by DRAM. The cache architecture is direct-mapped, look-aside, and write-through. Both synchronous (burst mode) SRAMs or asynchronous SRAMs are supported for a choice of best performance or lowest cost.

6.1 L2 Controller Features

6.1.1 Cache Size

The cache size is determined externally by the tagRAM and data SRAM sizes and how they are attached. The L2 cache controller supports all cache sizes. Loading constraints limit practical cache size to the 256K to 1M range for 66MHz operation.

6.1.2 Cache Responses

On memory reads, the L2 cache only responds to bursts from CPU bus agents. Single-beat reads are ignored. Table 6-1 shows the actions taken by the L2 cache based on transfer type and single-beat or burst mode.

Note: The CPU cache inhibit (CI#) signal is not used because cache-inhibited bus operations are always single-beat.

6.1.3 Cache Configuration

The L2 controller does not have any configuration bits for the size or organization of the tag and data SRAMs. The connections to the CPU address bus determine the size of the tag RAM and data SRAMs. There is a configuration bit that identifies the SRAM type as burst or asynchronous.

Since the L2 cache only caches addresses that are present in system memory, it does not have any configuration bits for cacheable address space.

6.1.4 L2 Performance

Typical Pipelined CPU to Memory Performance at 66 MHz

Responding Device	Read	Write
L2 (9ns Synchronous SRAM)	3-1-1-1-2-1-1-1...-2-1-1-1	Snarf
L2 (15ns Asynchronous SRAM)	3-2-2-2-3-2-2-2...-3-2-2-2	Snarf

6.2 L2 Cache Responses to CPU Bus Operations

The L2 caches data for the CPU bus masters. It supplies data on read hits, and it snarfs data on read misses and all writes. Table 6-1 details the operation of the L2 cache for various transfer type codes issued by a CPU bus master. The 660 does not use TT[4].

Table 6-1. L2 Cache Responses to CPU Bus Cycles

TT[0:3]	Type	CPU Bus Cycle	Cache Hit Action	Cache Miss Action
0000		Clean sector	Ignore	Ignore
0001	Single	Write with flush	Invalidate	Ignore
	Burst	Write with flush	Update as data is written to memory	Update as data is written to memory
0010		Flush sector	Invalidate	Ignore
0011	Single	Write with kill	Invalidate	Ignore
	Burst	Write with kill	Update as data is written to memory	Update as data is written to memory
0101	Single	Read	Ignore	Ignore
	Burst	Read	Claim cycle and supply data to CPU bus	Update as data is read from memory
0110		Kill sector	Invalidate	Ignore
0111	Always Burst	Read with intent to modify	Claim cycle and supply data to CPU bus	Ignore
1000		Reserved	Ignore	Ignore
1001	Always Single	Write with flush atomic	Invalidate	Ignore
1010		External control out	Ignore	Ignore
1011		Reserved	Ignore	Ignore
1100		TLB invalidate	Ignore	Ignore
1101	Single	Read atomic	Ignore	Ignore
	Burst	Read atomic	Claim cycle and supply data to CPU bus	Update as data is read from memory
1110		External control in	Ignore	Ignore
1111	Always Burst	Read with intent to modify atomic	Claim cycle and supply data to CPU bus	Ignore

Accesses to populated memory are snooped by L2, regardless of the state of GBL#. The 660 Bridge only uses GBL# as an output.

6.3 L2 Cache Responses to PCI Bus Mastered Transactions

The 660 maintains L2 coherency during PCI to memory transactions as shown in Table 6-2 for non-603 operation, and in Table 6-3 for 603 operation. The L2 does not supply data directly to the PCI bus. The L2 is not updated during a PCI transaction.

Table 6-2. L2 Operations for PCI to Memory Transactions, Non-603 Mode

PCI Bus Transaction	CPU Bus Broadcast Snoop Cycle		L2 Operation	
	Operation	TT[0:4]	L2 Hit	L2 Miss
Memory Read	Clean	00000	Ignore	Ignore
Memory Write	Flush Sector	00100	Invalidate Block	Ignore
Initiate Lock (Read)	Single-Beat Write with Flush	00010	Invalidate Block	Ignore

Table 6-3. L2 Operations for PCI to Memory Transactions, 603 Mode

PCI Bus Transaction	CPU Bus Broadcast Snoop Cycle		L2 Operation	
	Operation	TT[0:4]	L2 Hit	L2 Miss
Memory Read	Single-Beat Read	01010	Ignore	Ignore
Memory Write	Single-Beat Write with Flush	00010	Invalidate Block	Ignore
Initiate Lock (Read)	Single-Beat Write with Flush	00010	Invalidate Block	Ignore

6.4 Error Checking Support

The 660 L2 cache can be configured as none, 64, or 72 bits wide.

When no L2 is installed, CPU bus error detection can be enabled by setting BCR[C4] bit 2 to 1, and L2 parity error detection (BCR[C4] bit 3) becomes a don't_care.

The 64-bit configuration offers decreased cost but does not offer L2 or CPU bus data parity error checking. There are two options for using 64-bit SRAM:

- Disable L2 parity error detection (BCR[C4] bit 3), and disable CPU bus data parity error detection (BCR[C4] bit 2), or
- Disconnect DPE# between the CPU and the 664, and pull up the DPE# pin of the 664. This allows CPU bus error detection to be enabled. Parity will be checked on writes, but not on reads.

The 72-bit mode supports parity bit storage and checking. In 72-bit mode, CPU bus error detection can be enabled by setting BCR[C4] bit 2 to 1, and L2 parity error detection can be enabled by setting BCR[C4] bit 3 to 1. While enabled, parity bits are stored in the SRAM when an L2 entry is updated. Parity checking is done when data is supplied from the SRAM.

If a CPU data bus parity error occurs on a read sourced by the L2 SRAM, both the L2 and CPU bus parity error bits will be set. If the data is sourced by the DRAM, then the CPU data bus parity error bit will be set, but the L2 parity error bit will not be set.

ECC mode operation does not affect SRAM error checking because ECC mode operations are confined to the 660 to DRAM interface. The SRAM is a CPU bus device which is not connected to the DRAM subsystem.

6.5 External L2 Cache Operation

The onboard L2 cache controller can be disabled to incorporate an external L2. The only constraints that the 660 imposes on such an external L2 are those imposed on all CPU bus agents (see Section 3.8). The internal L2 cache controller can be disabled to accommodate an external L2 cache with greater associativity, write-back capability, or other improved capabilities.

6.6 TagRAM

The L2 controller requires external tagRAM. The following tagRAM parts are known to be supported by the 660: IDT71B74S10 (asynchronous), and IDT71215S10 and IDT71216S10 (both synchronous). These are 10ns parts, but tagRAM speed should be determined based on total system considerations. Typical systems will operate with 12ns tagRAMs at 66MHz.

Connectivity of the 660 with an IDT71216 16k x 15 TagRAM is illustrated by the *IBM PowerPC 603/604 Reference Design Technical Specification* and the *IBM PowerPC 604 SMP Reference Design Technical Specification*.

6.6.1 TAG_MATCH

While used with the IDT parts, the TAG_MATCH input to the 660 is driven by the MATCH output of the tags. This is an active high, open collector output, which must be pulled up with a low value resistor in order to work properly.

In Figure 6-10, TAG_MATCH is tied to the MATCH output of both tagRAMs. In this case, CPU_ADDR[12] is tied to the CS input of one tagRAM, and to the CS# input of the other. Thus, one of the tags is disabled for any given access, and the MATCH output of that tagRAM will tristate. leaving the other tagRAM active. This stacks the tags on top of each other, so that one holds the tags for the upper half of the pages, and the other holds the tags for the lower half of the pages.

In contrast, in Figure 6-11, the tagRAMs are used in parallel, with half of the tag stored in each RAM. Both halves of the tag must hit (and the VALID bit be set) for the tag to MATCH. Both MATCH outputs must go active high (tristate) for the TAG_MATCH input to go high.

6.7 SRAM

The 660 L2 controller requires external SRAM, which can be either synchronous or asynchronous. Synchronous vs. asynchronous SRAM is selected via the Bridge Chipset Options 3 BCR (see section 10.3.46).

6.7.1 Synchronous

Synchronous SRAM must be capable of linear burst ordering (00–01–10–11–00–...). When synchronous SRAM is used, the SRAM_ALE output is unused (and driven high) because the synchronous SRAMs contain an address latch.

The speed of the SRAM should be determined based on total system considerations. However, typical systems will operate with 9ns devices at 66MHz.

6.7.2 Asynchronous

While asynchronous SRAM is in use, the SRAM_ALE signal controls an external address latch (74F373, for example).

The speed of the SRAM should be determined based on total system considerations. However, typical systems will operate with 15ns devices at 66MHz.

6.7.3 Dual (Sync and Async) Capable Systems

Asynchronous SRAM uses the SRAM_ADS#/ADDR0 and SRAM_CNT_EN#/ADDR1 signals as the two least significant address bits, while synchronous SRAM uses them as control signals. Synchronous SRAMs internally latch in the initial value of the two least significant address bits from the CPU address bus. Systems designed to support either type of SRAM must use either jumpers or (fast) external logic to provide the necessary signal routing.

6.8 SRAM and TagRAM Connections

Figure 6-1 through Figure 6-12 show a detailed example of the connections between the 660, the SRAM, and the TagRAM. The information in these figures is presented in block diagram form and illustrates several different combinations of synchronous and asynchronous SRAM and tagRAM that can be used to create different sizes of L2. Table 6-4 is a guide to the configuration illustrations. For example, to use synchronous SRAM and asynchronous tagRAM to implement a 512K L2, see Figure 6-2 for an SRAM connection example, and Figure 6-12 for a tagRAM connection example.

Table 6-4. Index of SRAM and TagRAM Example Configurations

SRAM	Tag RAM	256K L2		512K L2		1M L2	
		SRAM	TagRAM	SRAM	TagRAM	SRAM	TagRAM
Sync	Sync	Figure 6-1	Figure 6-8	Figure 6-2	Figure 6-9	Figure 6-3, Figure 6-4	Figure 6-10
Sync	Async	Figure 6-1	Figure 6-11	Figure 6-2	Figure 6-12	—	—
Async	Sync	Figure 6-5	Figure 6-8	Figure 6-6	Figure 6-9	Figure 6-7	Figure 6-10
Async	Async	Figure 6-5	Figure 6-11	Figure 6-6	Figure 6-12	—	—

SRAM configurations that present more than 4 SRAM device loads may not meet worst case timing specifications at 66MHz on the WE# nets. The loading effect of the SRAM configuration on CPU bus operation should also be considered.

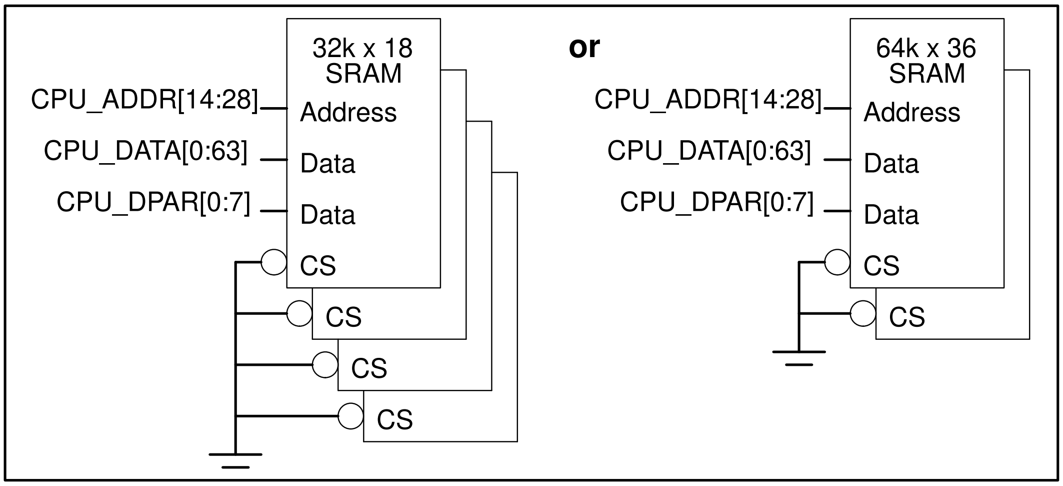


Figure 6-1. Synchronous SRAM, 256K L2

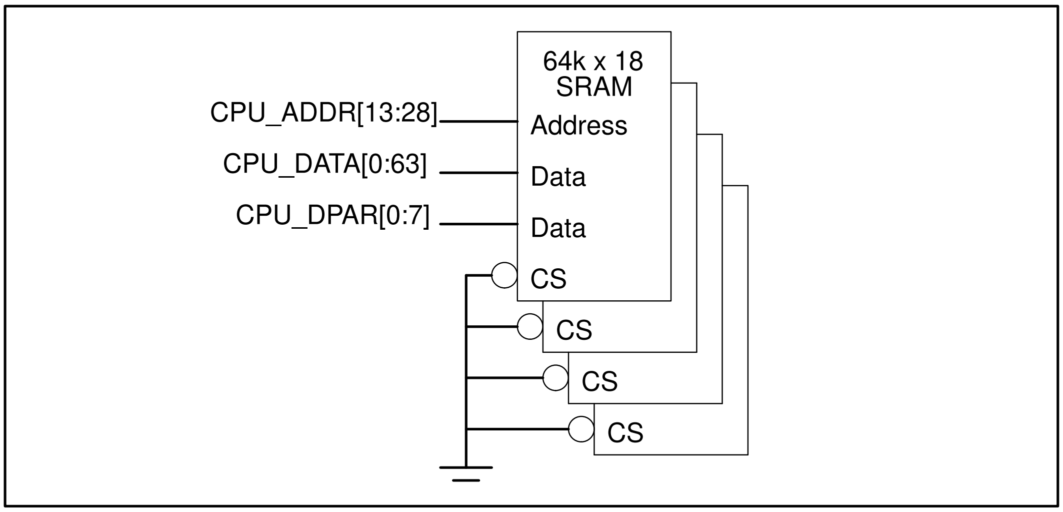


Figure 6-2. Synchronous SRAM, 512K L2

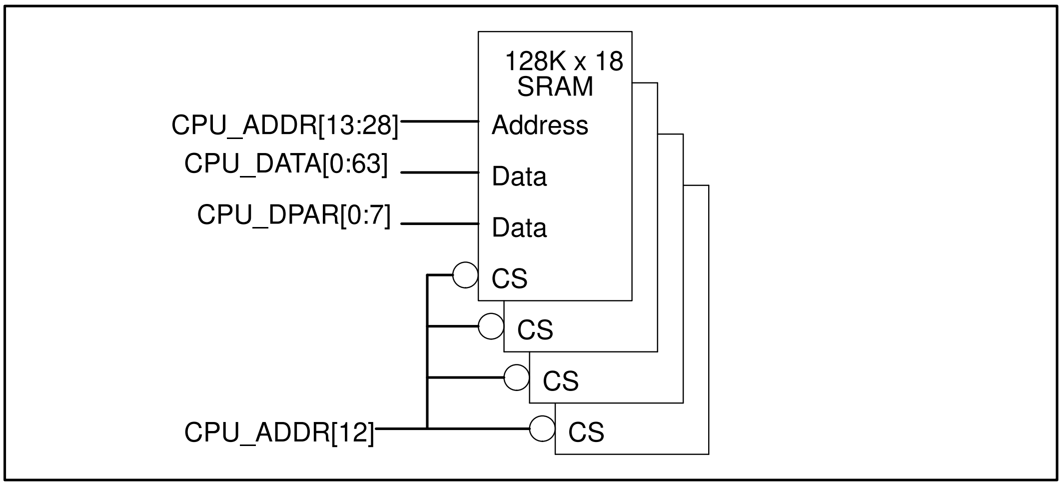


Figure 6-3. Preferred Synchronous SRAM, 1M L2

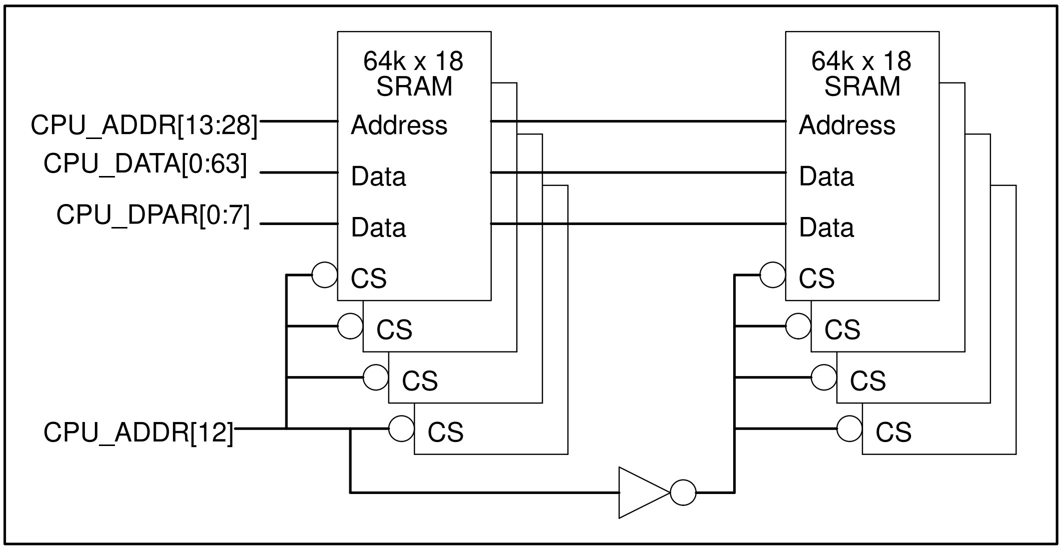


Figure 6-4. Alternate Synchronous SRAM, 1M L2

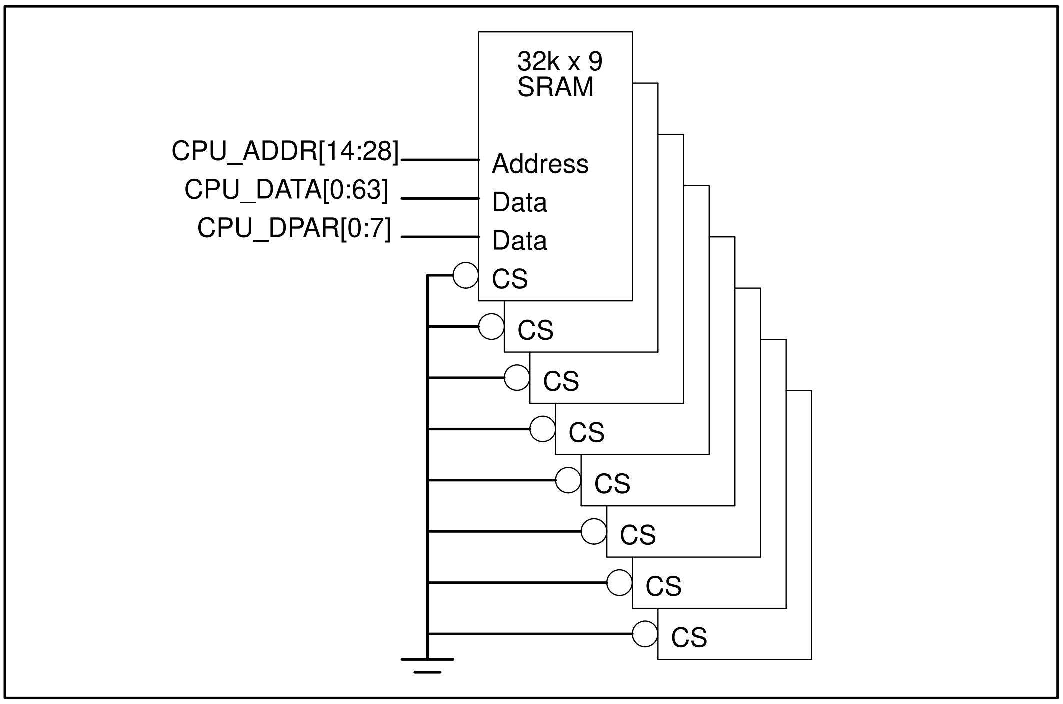


Figure 6-5. Asynchronous SRAM, 256K L2

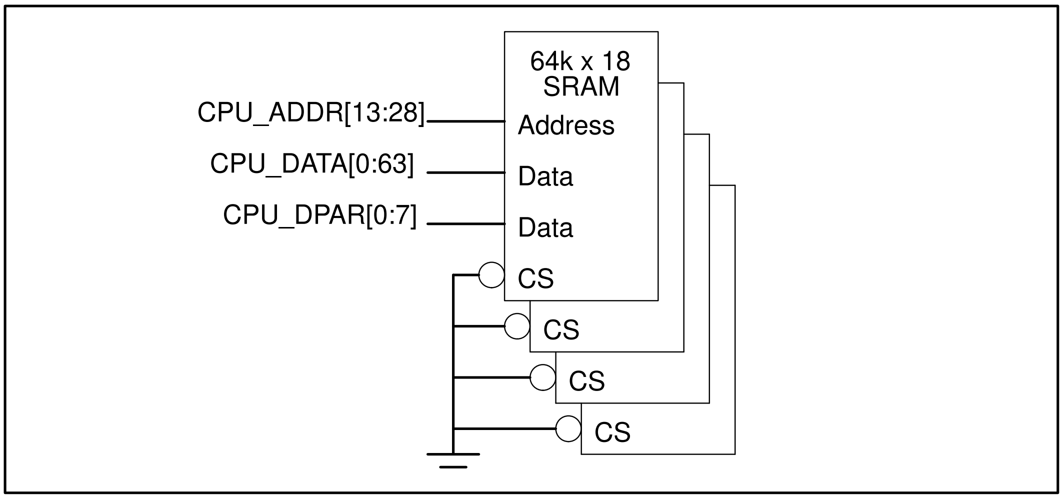


Figure 6-6. Asynchronous SRAM, 512K L2

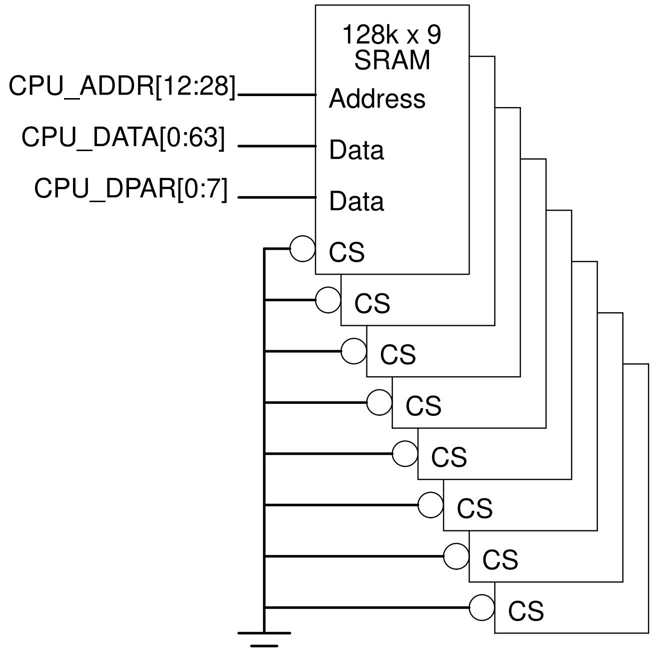


Figure 6-7. Asynchronous SRAM, 1M L2

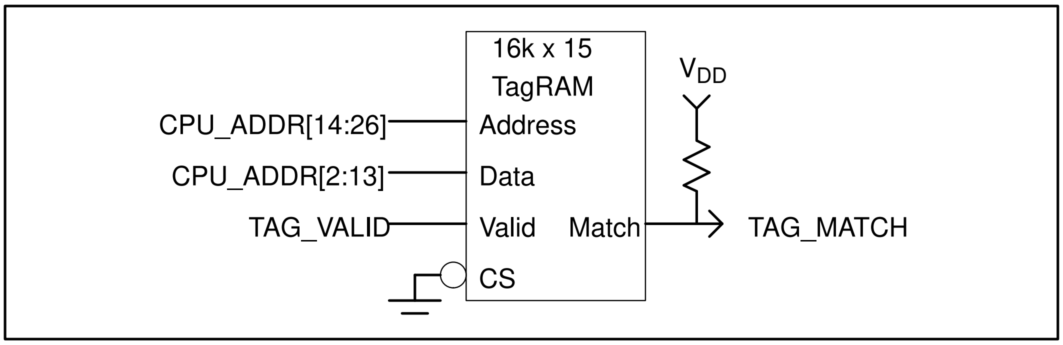


Figure 6-8. Synchronous TagRAM, 256K L2

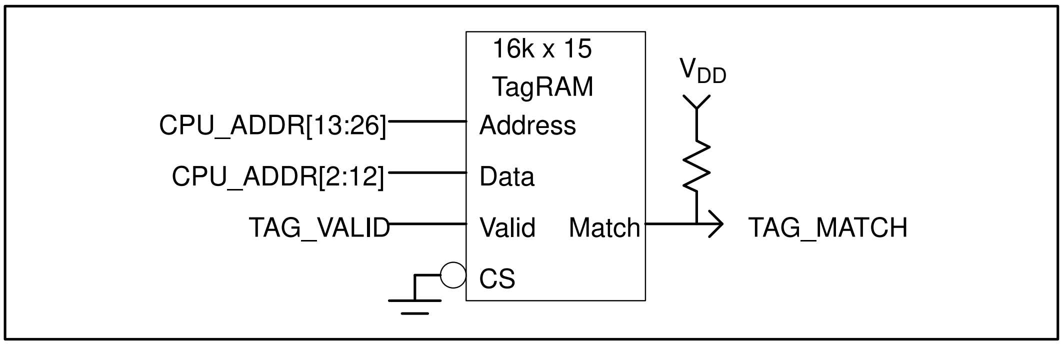


Figure 6-9. Synchronous TagRAM, 512K L2

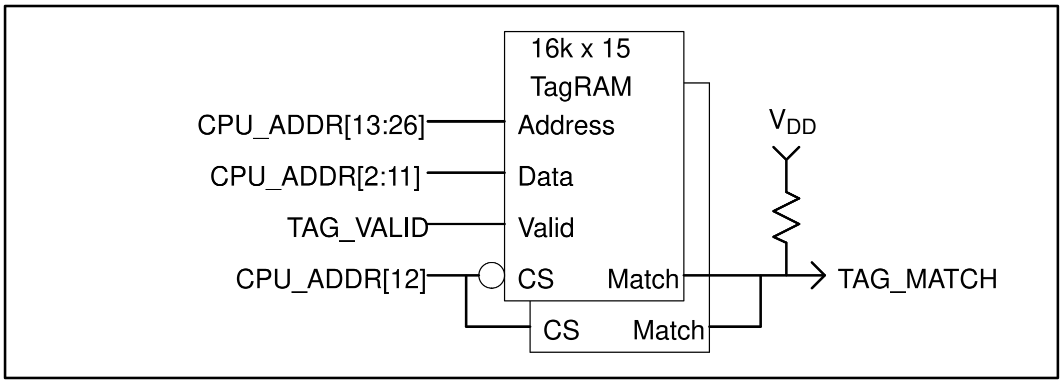


Figure 6-10. Synchronous TagRAM, 1M L2

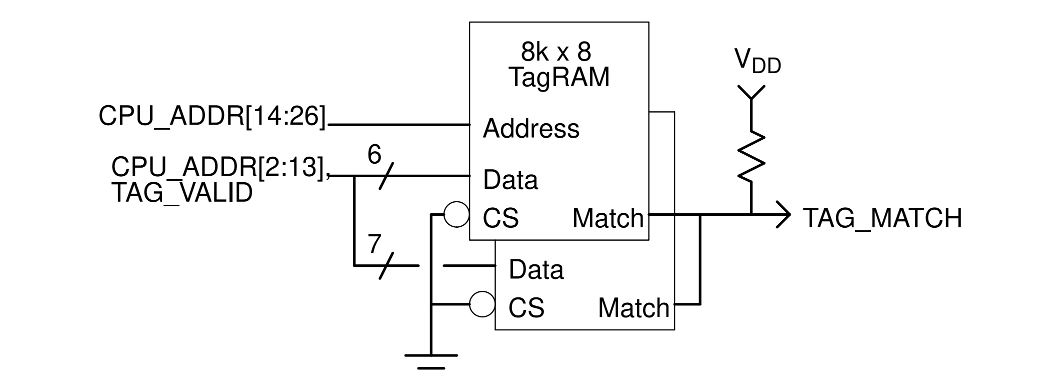


Figure 6-11. Asynchronous TagRAM, 256K L2

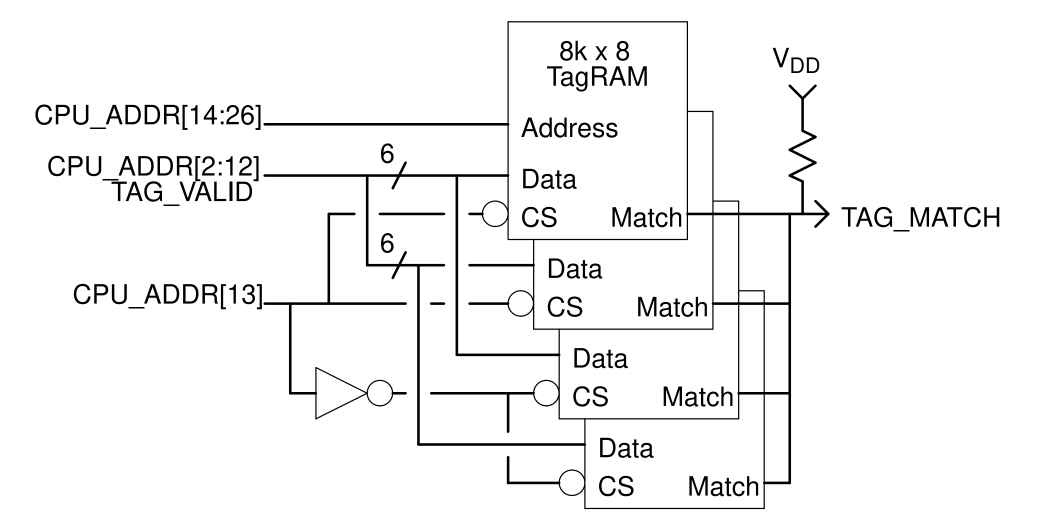


Figure 6-12. Asynchronous TagRAM, 512K L2

6.9 L2 Bridge Control Registers

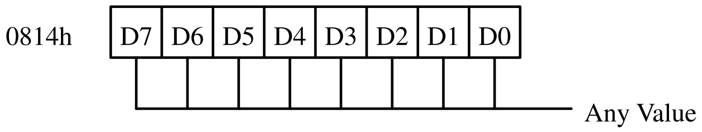
The following BCRs control the operation of the L2.

6.9.1 L2 Invalidate BCR

Direct Access 8000 0814	Write Only	Reset: Undefined

A write to this BCR causes all contents of the internal L2 cache to be invalidated (by pulsing TAG_CLR# active for several CPU clocks). The internal L2 cache does not have to be disabled during this operation. Reads to this register are undefined and do not cause an L2 invalidate.

This register can be put into external register support mode so that writes to this register are latched into this register and are also forwarded to the PCI. In this mode, reads to this register are always forwarded to the PCI, which allows functions to be added to the reserved bit locations of this register.

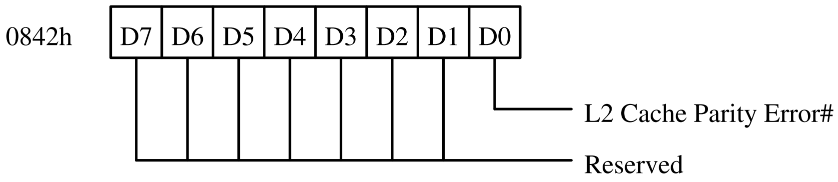


Bits 7:0 Writing any value to this register causes the L2 invalidate operation.

6.9.2 L2 Error Status BCR

Direct Access 8000 0842	Read Only	Reset to 01h

This 8-bit, read-only register indicates if a parity error has been detected during a CPU read from the L2 cache. This bit is deactivated by reading the L2 cache parity error read and clear register (port 8000 0843h). This bit may also be accessed via the error status 2 register (index C5h, section 10.3.41). Note that L2 parity errors also cause the CPU bus data parity error bit to be set.

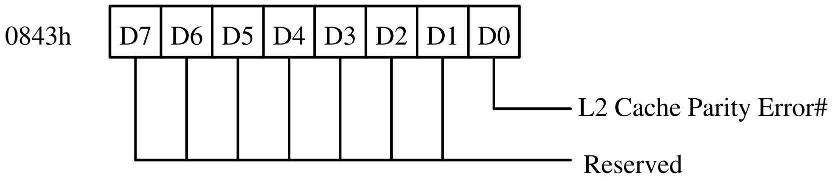


Bit 0 L2 Cache Parity Error#:
0 = Error Detected
1 = No Error Detected.

6.9.3 L2 Parity Error Read and Clear BCR

Direct Access 8000 0843	Read Only	Reset to 01h

This 8-bit, read-only register indicates if a parity error has been detect during a CPU read from the L2 cache and clears the error if it is active. This bit may also be accessed via the error status 2 register (index C5h, section 10.3.41), but that access will not automatically clear the bit. Note that L2 parity errors also cause the CPU bus data parity error bit to be set.

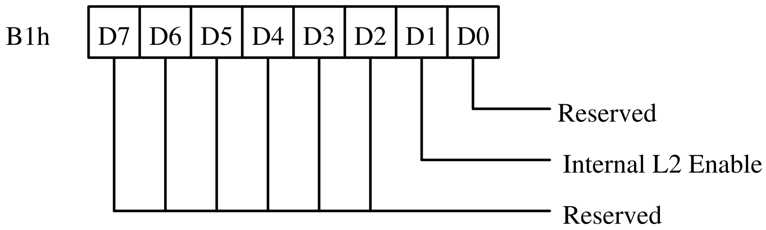


Bit 0 L2 Cache Parity Error#:
0 = Error Detected
1 = No Error Detected.

6.9.4 Cache Status Register

Index B1h	Read/Write	Reset to 43h

This 8-bit, read/write BCR shows the status of the L1 and L2 caches.



Bit 0 Reserved. Hardcoded to a 1.

Bit 1 Internal L2 Enable: This bit is used with the L2 Cache Enable bit (bit 6 of the system control 81C BCR, section 10.2.2.3) to enable the internal L2 cache. The internal L2 is enabled only when both bits are set to 1.
0 : Internal L2 disabled.
1 : Internal L2 enabled (iff L2 Cache Enable bit is set).

Bit 2:7 Reserved

6.9.5 Other L2-Related BCRs

Also see the following sections for information on related BCRs:

Bridge Control Register	Index	R/W	Bytes	See
Error Enable 2	Index C4	R/W	1	10.3.40
Error Status 2	Index C5	R/W	1	10.3.41
Bridge Chip Set Options 3	Index D4	R/W	1	10.3.46
System Control 81C	8000 081C	R/W	1	10.2.2.3

Section 7
ROM

The 660 Bridge implements a 2M ROM space from 4G–2M to 4G. The 660 provides two boot ROM device access methods which minimize pin and package count while still allowing a byte-wide Flash™ ROM device to source 8-byte wide data.

7

One ROM access method used by the 660 (referred to as the direct-attach ROM mode), attaches the ROM directly to the 660 using the PCI_AD lines. This mode is compatible with that used by the 650 Bridge, and is required when using the Intel™ SIO ISA bridge because the SIO does not support mapping of the ROM to the ISA bus. The direct-attach mode also supports ROM device writes and write-protect commands.

The other ROM access method (remote ROM mode – see section 7.2) attaches the ROM device to an external PCI agent which supports the PowerPC Reference Platform ROM space map and access protocol. CPU bus master transfers to ROM space are forwarded to the PCI bus and claimed by the PCI agent, which supplies the ROM device data. This PCI device is typically a PCI to ISA bridge. The ROM device attaches to the ISA bridge through the ISA bus lines, thereby saving a PCI bus load. The 660 supplies write-protect capability in this mode.

The ROM mode is indicated to the 660 on the strapping pin configuration bits during power-on-reset (POR). See section 8.1.3.

7.1 Direct-Attach ROM Mode

The ROM device attaches to the 660 by means of control lines and the PCI_AD[31:0] lines. When a CPU bus master reads from the ROM, the 660 masters a BCR transaction, during which it reads the ROM and returns the data to the CPU. CPU writes to the ROM and ROM write-protection operations are also forwarded to the ROM device.

ROM accesses flow from the CPU bus to the 660. As shown in Figure 7-1, the data and address flow from the 660 to the ROM over the PCI_AD lines. ROM control flows from the 660 to the ROM over control lines that are not a part of the PCI bus.

Although connected to the PCI_AD lines, the direct-attach ROM is not a PCI agent. The ROM and the PCI agents do not interfere with each other because the ROM is under

660 control, and the 660 does not enable the ROM except during ROM cycles. The 660 accesses the ROM by means of BCR transactions. Other PCI devices cannot read or write the ROM because they cannot generate BCR transactions.

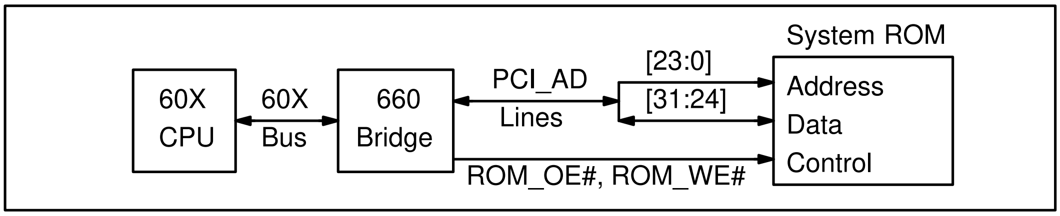


Figure 7-1. ROM Connections

7.1.1 ROM Reads

When a CPU bus master reads from memory addresses mapped to ROM space, the 660 arbitrates for the PCI bus and then masters a BCR transaction on the PCI bus. During this transaction, the 660 reads the ROM eight times, accumulates the data, and returns the double-word to the CPU. The 660 then completes the PCI transaction and releases the PCI bus.

7

The 664 drives the address of the required byte over PCI_AD[23:0] to the ROM address pins. The ROM drives back the data on PCI_AD[31:24], where it is received by the 663.

This ROM read discussion assumes that the system is in big-endian mode. For the effects of little-endian mode operation on ROM reads, see Section 7.1.1.3.

7.1.1.1 ROM Read Sequence

Figure 7-2 is a timing diagram of a CPU to ROM read transaction. This case assumes that the PCI bus is parked on the CPU, so that the 660 has a valid PCI bus grant when the CPU starts the CPU bus transfer.

Initially, the CPU drives the address and address attributes onto the CPU bus and asserts TS#. The 660 decodes the CPU transfer as a ROM read transaction. It is possible for TS# to be asserted across either a rising or falling edge of PCI_CLK. The 660 must only assert (and negate) PCI bus signals on the rising edge of PCI_CLK, so if TS# is asserted across a rising edge of PCI_CLK, the 660 waits one CPU_CLK to synchronize to the PCI bus.

The 660 initiates a BCR transaction by asserting PCI_FRAME# on the rising edge of PCI_CLK. Note that the 660 is driving PCI_AD[23:0] with the ROM address of byte 0 of the 8-byte aligned double-word. The 660 leaves PCI_AD[31:24] tri-stated, and asserts ROM_OE# to enable the ROM to drive the data onto these bits. On the next PCI_CLK, the 660 negates PCI_FRAME# and asserts PCI_IRDY#.

The ROM drives the requested data onto its data pins, across PCI_AD[31:24], and into the 660. Seven PCI_CLKs after the 660 asserts PCI_FRAME#, it sends ROM_LOAD low. On the next clock, the 660 latches in the ROM data on PCI_AD[31:24], sends ROM_LOAD# high, and increments the ROM address on PCI_AD[23:0]. The byte from the ROM is latched into a byte shift register, which accumulates the bytes in an 8-byte double-word. The contents of the shift register move through the 660 and onto the CPU data bus.

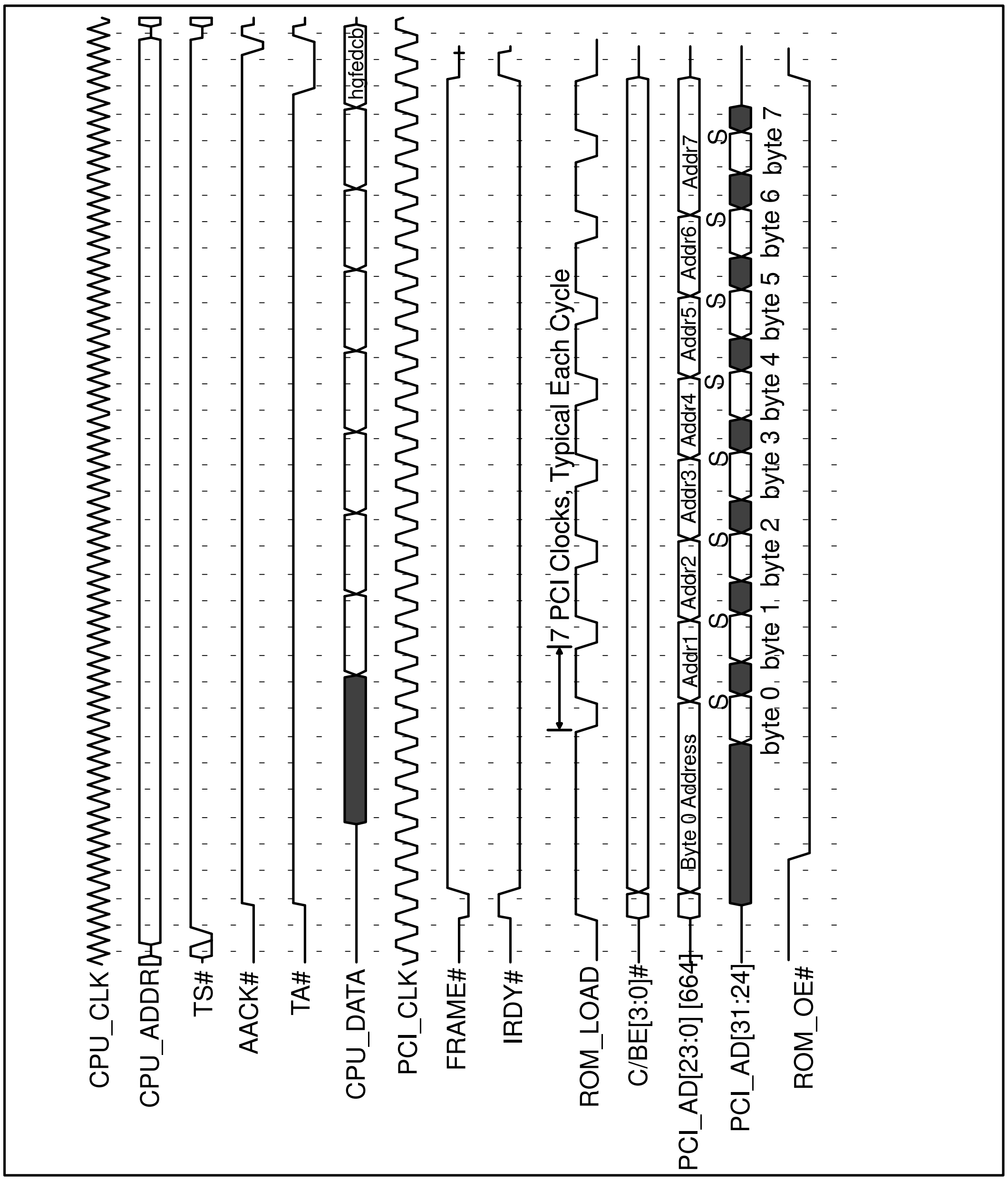


Figure 7-2. ROM Read Timing Diagram

The ROM then drives the next data byte onto PCI_AD[31:24]. Seven PCI_CLKs after it negated ROM_LOAD for the previous byte, the 660 again negates ROM_LOAD, and also shifts the previous byte of ROM data to the next position. On the next PCI_CLK, the 660 sends ROM_LOAD high and increments the ROM address on PCI_AD[23:0]. This pattern is repeated until all eight bytes have been loaded into the shift register.

After the last byte has been latched into the 660 by the falling edge of ROM_LOAD, the 660 completes the PCI transaction by deasserting PCI_IRDY#. It also negates ROM_OE# to clear the PCI bus. After the last byte of data has had time to propagate through onto the CPU data bus, the 660 signals TA# to the CPU. Table 7-1 shows the data and address flow during the transaction.

On a single-beat transfer, the CPU asserts and negates AACK# concurrently with TA#. For a burst transfer, the 660 asserts TA# for four CPU_CLKs to return four identical double-words to the CPU. This is the only difference between single-beat and burst ROM reads.

Also note that PCI_DEVSEL#, PCI_TRDY#, PCI_STOP#, and ROM_WE# are negated throughout the transaction.

Table 7-1. ROM Read Data and Address Flow

ROM Access #	ROM Data Byte	ROM Address PCI_AD[23:0]	ROM Data PCI_AD[31:24]	CPU_DATA[0:63] After Shift (BE Mode)	CPU_DATA[0:63] After Shift (LE Mode)
1	Byte 0	XX XXX0h	a	—	—
2	Byte 1	XX XXX1h	b	—	—
3	Byte 2	XX XXX2h	c	—	—
4	Byte 3	XX XXX3h	d	—	—
5	Byte 4	XX XXX4h	e	—	—
6	Byte 5	XX XXX5h	f	—	—
7	Byte 6	XX XXX6h	g	—	—
8	Byte 7	XX XXX7h	h	abcd efgh	hgfe dcba

7

7.1.1.2 Address, Transfer Size, and Alignment

During ROM reads, system ROM is linear-mapped to CPU memory space from 4G – 2M to 4G (FFE0 0000h to FFFF FFFFh). This address range is translated onto PCI_AD[23:0] as 0 to 2M (0000 0000h to 001F FFFFh). Since the CPU begins fetching instructions at FFF0 0100h after a reset, the most convenient way to use a 512K device as system ROM with the CPU is to use it from 4G – 512K to 4G. Connecting PCI_AD[18:0] to ROM_A[18:0] with no translation implements this. With this connection, the system ROM is aligned with 4G – 2M, but with alias addresses every 512K up to 4G. Other size devices can also be implemented this way.

The CPU read address need not be aligned on an 8-byte boundary. A CPU read from any ROM address of any length that does not cross an 8–byte boundary, returns all eight bytes of that double-word from the ROM. For example, the operations shown in Table 7-1 could have been caused by a CPU memory read to FF80 0100h, FF80 0101h, or FF80 0105h.

7.1.1.3 Endian Mode Considerations

In little-endian mode, the address munging done by the CPU has no effect because PCI_AD[2:0] are forced to 000 during the address phase by the 660 at the beginning of the transaction. However, in little-endian mode the byte swapper is enabled, so the bytes of ROM data returned to the CPU are swapped as shown in the last column of Table 7-1.

7.1.1.4 4-Byte Reads

The 660 handles 4-byte ROM reads (and all ROM reads of less than 8 bytes) as if they were 8-byte reads. All 8 bytes are gathered by the 660, and all 8 bytes are driven onto the CPU data bus.

7.1.2 ROM Writes

The 660 decodes a CPU store word instruction to CPU address FFFF FFF0h as a ROM write cycle. (Note that the 660 treats any access from FFE0 0000h to FFFF FFFE, with CPU_A[31]=0, as an access to FFFF FFF0.) The three low-order bytes of the CPU data word are driven onto the ROM address lines, and the high-order byte is driven onto the ROM data lines. For example, a store word instruction with data = 0012 3456h writes 56h to ROM location 00 1234h. Only single-beat, four-byte write transfers (store word) are supported. A ROM write is considered to be a BCR operation. The ROM write BCR is detailed in Section 7.3.1.

The ROM write discussion assumes that the system is in big-endian mode. For the effects of little-endian mode operation on ROM reads, see Section 7.1.2.3. In direct-attach mode, the ROM is attached to the 660 as shown in Figure 7-3.

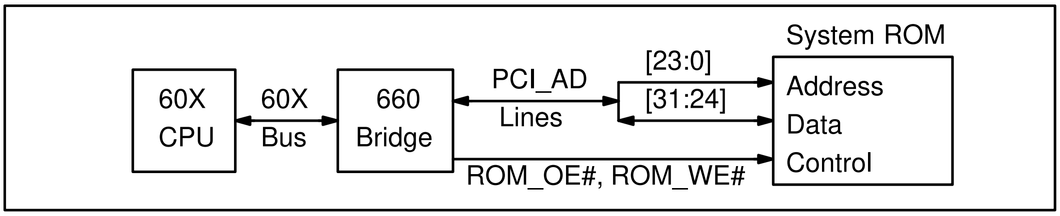


Figure 7-3. ROM Connections

7.1.2.1 ROM Write Sequence

This case assumes that the PCI bus is parked on the CPU. Initially, the CPU drives the address and address attributes onto the CPU bus and asserts TS#. The 660 decodes the CPU transfer as a ROM write transaction, which is a BCR transaction.

The 660 initiates a BCR transaction by asserting PCI_FRAME# on the rising edge of PCI_CLK. Note that the 660 is driving PCI_AD[23:0] with the ROM address and PCI_AD[31:24] with the ROM data. On the next PCI_CLK, the 660 negates PCI_FRAME# and asserts IRDY#. Four PCI_CLKs after the 660 asserts PCI_FRAME#, it asserts ROM_WE# for two PCI_CLKs.

The 660 completes the PCI transaction by deasserting PCI_IRDY#. The 660 signals TA# and AACK# to the CPU to signal transfer completion to the CPU. Also note that PCI_DEV-SEL#, PCI_TRDY#, PCI_STOP#, and ROM_OE# are negated throughout the transaction.

7.1.2.2 Write Protection

Write protection for direct-attach ROM is provided through the ROM lockout BCR (see Section 7.3.2). ROM write-lockout operations are compatible with the 650 bridge.

When a CPU bus master writes any data to memory address FFFF FFF1h, the 660 locks out all subsequent ROM writes until the 660 is reset. In addition, flash ROM devices can have the means to permanently lock out sectors by writing control sequences. Flash ROM specifications contain details. Note that the 660 treats any access from FFE0 0001 to FFFF FFFF, with CPU_A[31]=1, as an access to FFFF FFF1.

7.1.2.3 Data Flow In Little-Endian Mode

Figure 7-4 and Table 7-2 show the flow of CPU Data through the 660 to the ROM while the system is in little-endian mode. Note that the CPU Data bus is labeled in big-endian order, the PCI bus is labeled in little-endian order, and the 660 is labeled to match (and the bit significance within the bytes is maintained).

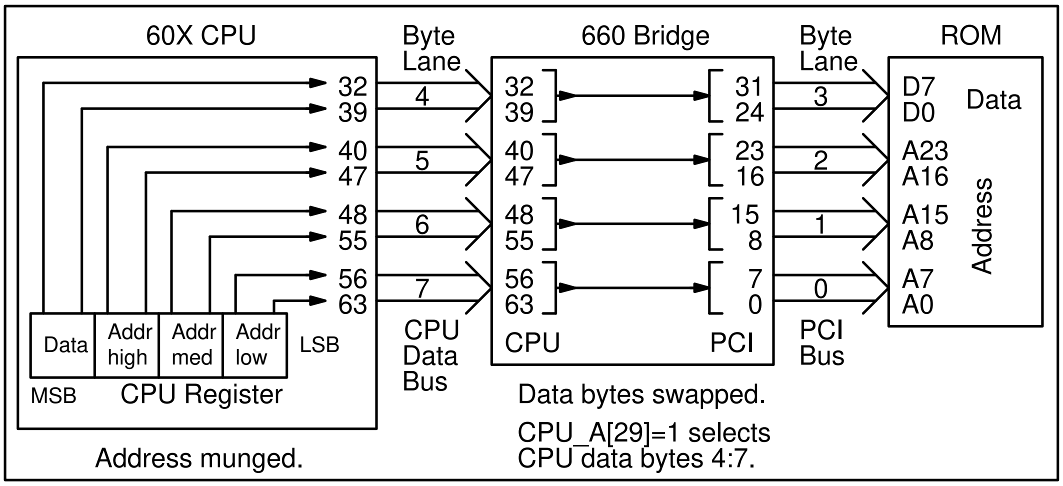


Figure 7-4. ROM Data and Address Flow In Little Endian Mode

When the CPU executes a store word instruction to FFFF FFF0h, the contents of the source register appear on CPU_DATA[32:63]. CPU_ADDR[29] is 1 (after the CPU munges the address), so the 660 selects CPU data byte lanes 4 through 7 as the source of the data. The system is in little-endian mode, so the buffer swaps the data bytes. If the register data is AB012345h, then ABh is written to address 012345h of the ROM. Only single-beat, four-byte write transfers (store word) are supported.

Table 7-2. ROM Write Data Flow in Little-Endian Mode

CPU Register	CPU DATA[0:63]	Content	PCI_AD[31:0]	ROM Signal
MSB	32:39	ROM Data	31:24	D[7:0]
	40:47	ROM Address high byte	23:16	A[23:16]
	48:55	ROM Address mid byte	15:8	A[15:8]
LSB	56:63	ROM Address low byte	7:0	A[7:0]

7.1.2.4 Data Flow In Big-Endian Mode

Figure 7-5 and Table 7-3 show the flow of CPU Data through the 660 to the ROM while the system is in big-endian mode. Note that the CPU Data bus is labeled in big-endian order, the PCI bus is labeled in little-endian order, and the 660 is labeled to match (and the bit significance within the bytes is maintained).

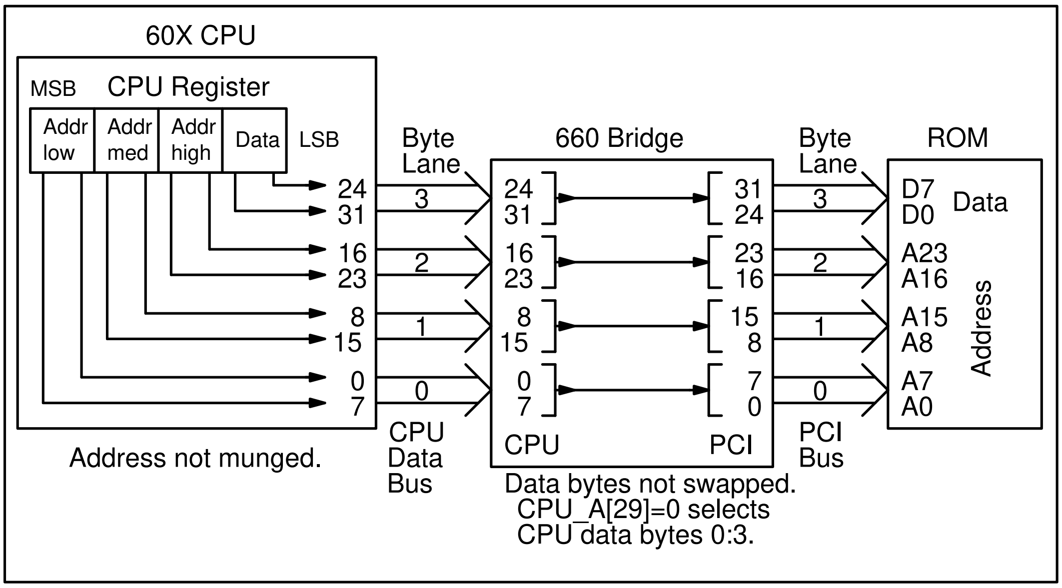


Figure 7-5. ROM Data and Address Flow In Big Endian Mode

When the CPU executes a store word instruction to FFFF FFF0h, the contents of the source register appear on CPU_DATA[0:31]. CPU_ADDR[29]=0, so the 660 selects CPU data byte lanes 0 through 3 as the source of the data. The system is in big-endian mode, so the buffer does not swap the data bytes. If the register data is 452301ABh, then ABh is written to address 012345h of the ROM. Only single-beat, four-byte write transfers (store word) are supported.

Table 7-3. ROM Write Data Flow in Big-Endian Mode

CPU Register	CPU DATA[0:63]	Content	PCI_AD[31:0]	ROM Signal
MSB	0:7	ROM Address low byte	7:0	A[7:0]
	8:15	ROM Address mid byte	15:8	A[15:8]
	16:23	ROM Address high byte	23:16	A[23:16]
LSB	24:31	ROM Data	31:24	D[7:0]

7.2 Remote ROM Mode

In a system that uses the remote ROM mode, the ROM device attaches to a PCI agent. When a CPU bus master reads from memory addresses mapped to ROM space, the 660 arbitrates for the PCI bus and then masters a memory read transaction on the PCI bus. The PCI agent claims the transaction and supplies the ROM device data. CPU writes to the ROM and ROM write-protection operations are also forwarded to the PCI agent.

As shown in Figure 7-6, the ROM access flows from the CPU to the 660 over the CPU bus, from the 660 to the PCI agent over the PCI bus, and from the PCI agent to the ROM device. The ROM device attaches to the PCI agent, not to the PCI_AD lines, so a PCI bus load is saved by the remote ROM method.

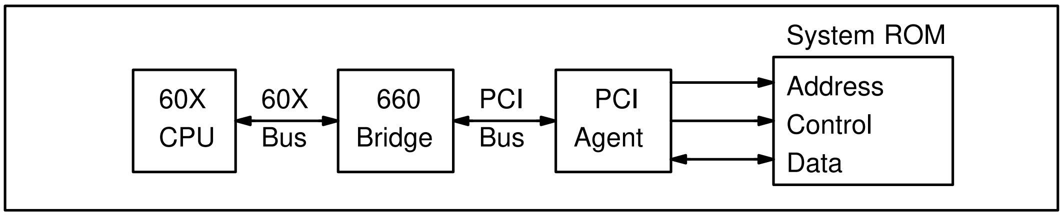


Figure 7-6. Remote ROM Connections

7.2.1 Remote ROM Reads

For remote ROM reads, the 660 arbitrates for the PCI bus, initiates eight single-byte PCI accesses, releases the PCI bus, and completes the CPU transfer. The eight single bytes of ROM data are assembled into a double-word in the 663 and passed to the CPU. Figure 7-7 shows the beginning of the operation, including the first two PCI transactions. Figure 7-8 shows the last part of the operation, including the last two PCI transactions.

During and following reset, compliant PCI agents are logically disconnected from the PCI bus except for the ability to respond to configuration transactions. These agents have not yet been configured with necessary operational parameters. PCI agents capable of the remote ROM access protocol reset with the ability to respond to remote ROM accesses before being fully configured. The CPU begins reading instructions at FFF0 0100h before it can configure the PCI devices.

If the 660 is retried any time during the sequence, it backs off and tries again (normal PCI protocol) from the beginning of the eight byte read.

The ROM read discussion assumes that the system is in big-endian mode.

7.2.1.1 Remote ROM Read Sequence

In response to a CPU bus read in the 4G – 2M to 4G address range, the 660 requests the PCI bus from the PCI arbiter. When the PCI bus is granted (or if the bus is already parked on the CPU), the 660 initiates a series of PCI memory-read transactions as shown in Table 7-4 for a CPU read from FFE0 0000h to FFFF FFFF. Note that the last column in Table 7-4 shows the effect of little-endian mode operation. See Section 7.2.1.4. The 660 follows normal PCI protocol during remote ROM reads.

The address of the first transaction is the low-order byte of the double-word pointed to by the CPU address (see Section 7.2.1.2). The 660 expects the low-order byte of ROM data in the 8-byte double-word to be returned on PCI byte lane 0, PCI_AD[7:0]. As shown in The 660 then masters seven more PCI read transactions, each time receiving back one byte of ROM data and driving it onto the CPU data bus as shown in Table 7-4. Note that the byte enables are incrementing within each 4-byte word pointed to by the PCI address.

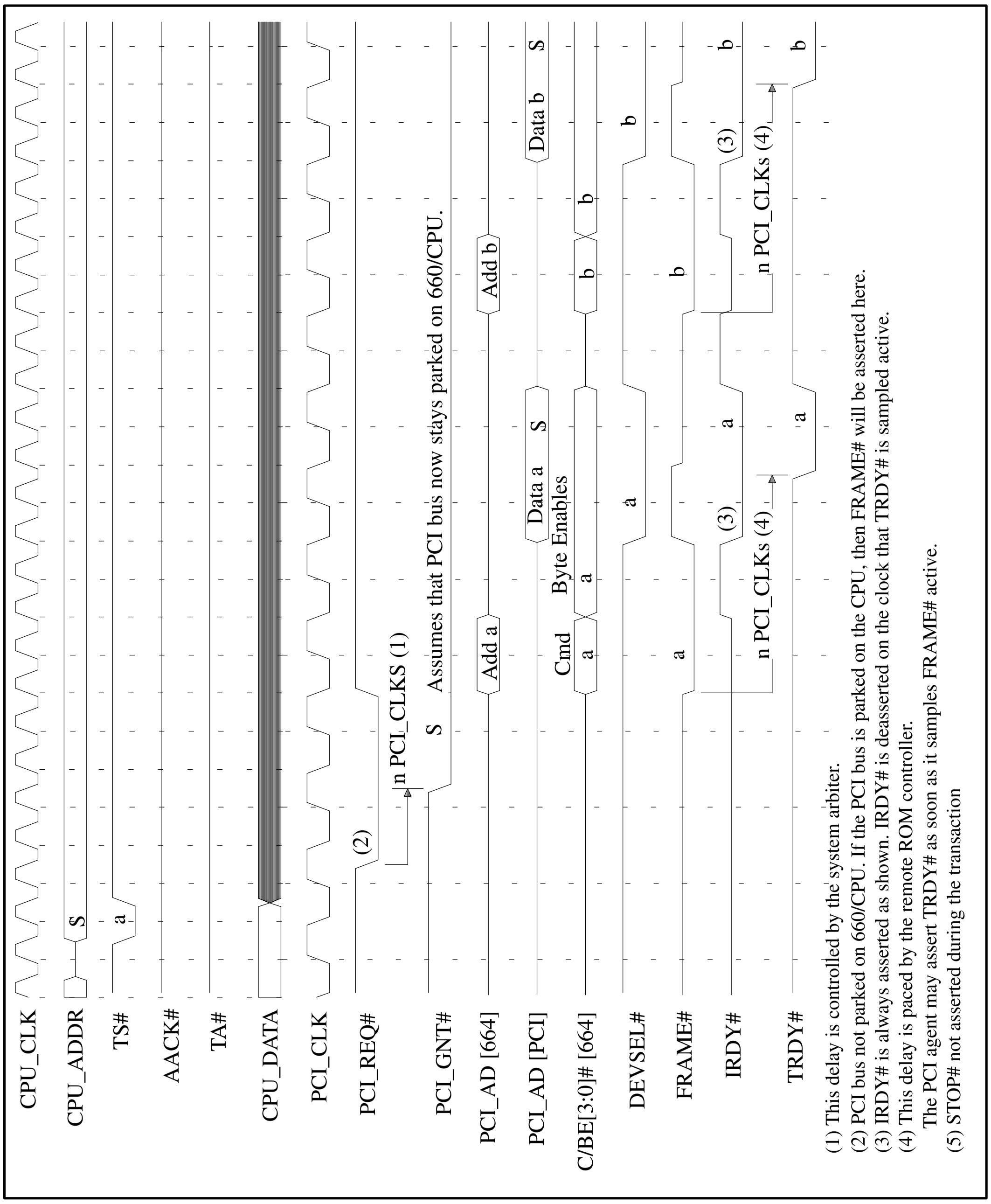


Figure 7-7. Remote ROM Read – Initial Transactions

At the completion of the eighth PCI read, the 660 drives the assembled double-word onto the CPU data bus. The 660 then signals completion of the transfer to the CPU.

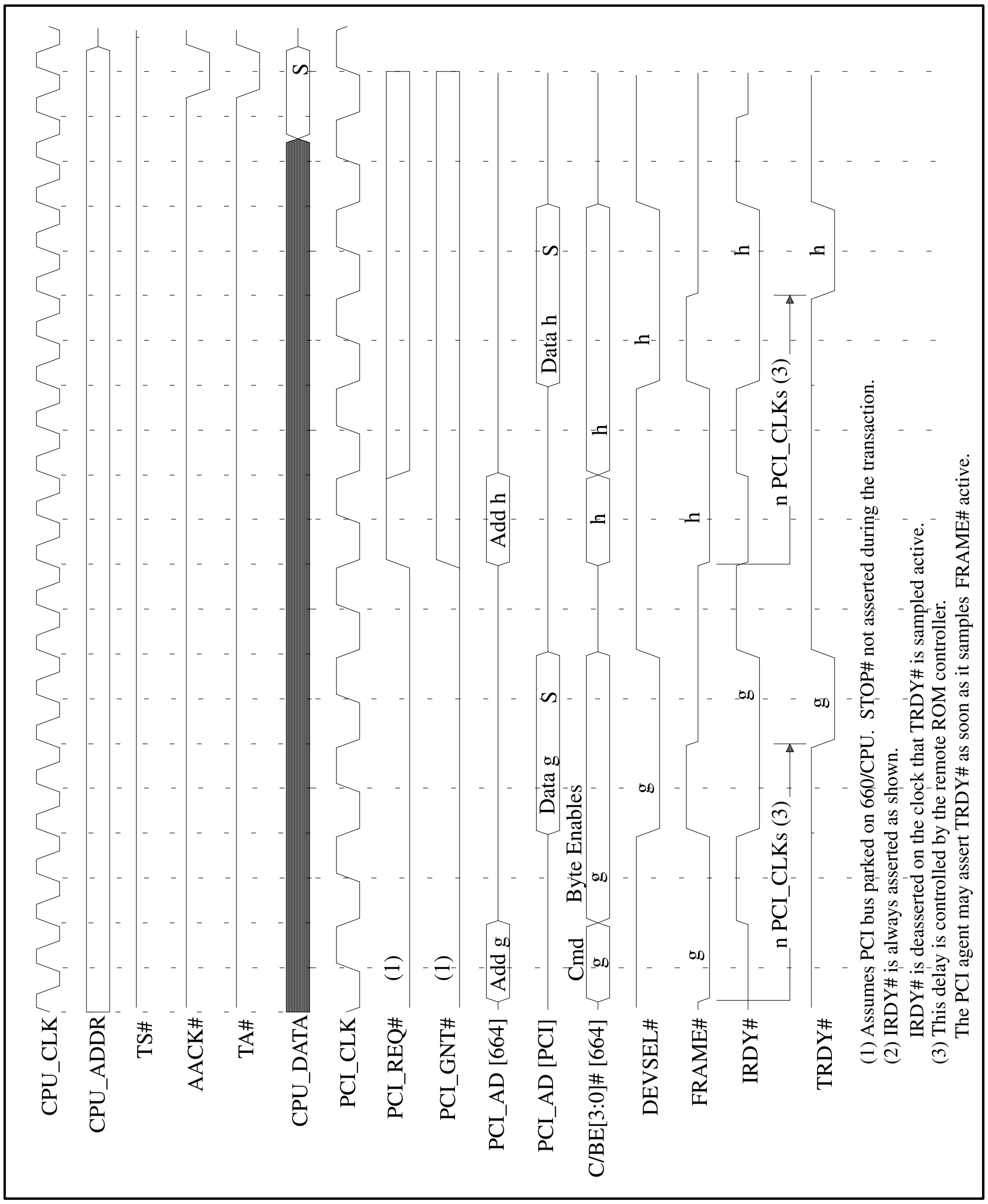


Figure 7-8. Remote ROM Read – Final Transactions

Remote ROM reads are not pipelined. The 660 does not assert AACK# to the CPU until the end of the remote ROM read sequence. The 660 asserts PCI_REQ# throughout the entire remote ROM read sequence.

Table 7-4. Remote ROM Read Sequence, CPU Address = FFFX XXX0

PCI Access #	PCI Bus Read Memory Address PCI_AD[31:0]	Byte Enables PCI_C/BE[3:0]#	ROM Addr	ROM Data	Big Endian CPU_DATA [0:63]	Little Endian CPU_DATA [0:63]
1	FFFX XXX0h	1110	0	a	—	—
2	FFFX XXX0h	1101	1	b	—	—
3	FFFX XXX0h	1011	2	c	—	—
4	FFFX XXX0h	0111	3	d	—	—
5	FFFX XXX4h	1110	4	e	—	—
6	FFFX XXX4h	1101	5	f	—	—
7	FFFX XXX4h	1011	6	g	—	—
8	FFFX XXX4h	0111	7	h	abcd efgh	hgfe dcba

7.2.1.2 Address, Transfer Size, and Alignment

The initial PCI address generated during the remote ROM read sequence is formed by copying the high-order 29 bits of the CPU address, and forcing the three low order bits PCI_AD[2:0] to 000b. This generates a base address that is aligned on an 8-byte boundary. While reading the lower 4 bytes, the 660 indicates which byte it is requesting using the PCI byte enables C/BE[3:0]#. After the first four bytes of ROM data are read, the 660 increments the address on the PCI_AD lines by 4 before executing the second four PCI reads.

The CPU read address need not be aligned on an 8-byte boundary. A CPU read from any address (in ROM space) of any length that does not cross an 8–byte boundary within a double-word returns all eight bytes of that double-word data from the ROM. For example, the operations shown in Table 7-4 could have been caused by a CPU memory read to FFF0 0100h, FFF0 0101h, or FFF0 0105h.

Errors occurring during remote ROM reads are handled as usual for the error type. No special rules are in effect.

7.2.1.3 Burst Reads

The 660 supports burst reads in remote ROM mode. The 660 supports 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 CPU burst.

A burst ROM read begins with the 660 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 burst ROM read, the 660 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.

For a single-beat read, the 660 asserts TA# and AACK# for one CPU_CLK cycle, and the CPU completes the transfer.

7.2.1.4 Endian Mode Considerations

In little-endian mode, the address munging done by the CPU has no effect because PCI_AD[2:0] are forced to 000 during the address phase by the 660 at the beginning

7

of the transaction. However, in little-endian mode the byte swapper is enabled, so the bytes of ROM data returned to the CPU are swapped as shown in the last column of Table 7-4.

7.2.1.5 Four-Byte Reads

The 660 handles 4-byte ROM reads (and all ROM reads of less than 8 bytes) as if they were 8-byte reads. All 8 bytes are gathered by the 660, and all 8 bytes are driven onto the CPU data bus.

7.2.2 Remote ROM Writes

While the 660 is configured for remote ROM operation, the 660 forwards all CPU to ROM write transfers to the PCI bus as memory writes. The PCI agent that is controlling the remote ROM acts as the PCI target during CPU to ROM write transfers, executes the write cycle to the ROM, and may provide ROM write-protection.

7.2.2.1 Write Sequence

A CPU bus master begins a remote ROM write transaction by initiating a one-byte, single-beat memory write transfer to CPU bus address range 4G – 2M to 4G (FFE0 0000h to FFFF FFFFh).

The 660 decodes the CPU transfer, arbitrates for the PCI bus, and initiates a memory write PCI transaction to the same address in the 4G – 2M to 4G address range.

The PCI agent that is controlling the remote ROM (such as the PCI to ISA Bridge), claims the transaction, manages the write cycle to the ROM device, and signals TRDY#.

The 660 then completes the PCI transaction, and signals AACK# and TA# to the CPU. Note that remote ROM writes are neither posted or pipelined.

7.2.2.2 Write Protection

Write protection can be provided by the PCI agent that controls the ROM. In addition, some flash ROM devices can have the means to permanently lock out sectors by writing control sequences. The 660 also has a write lockout in the Bridge Chipset Options 2 register (bit 0 of index BBh).

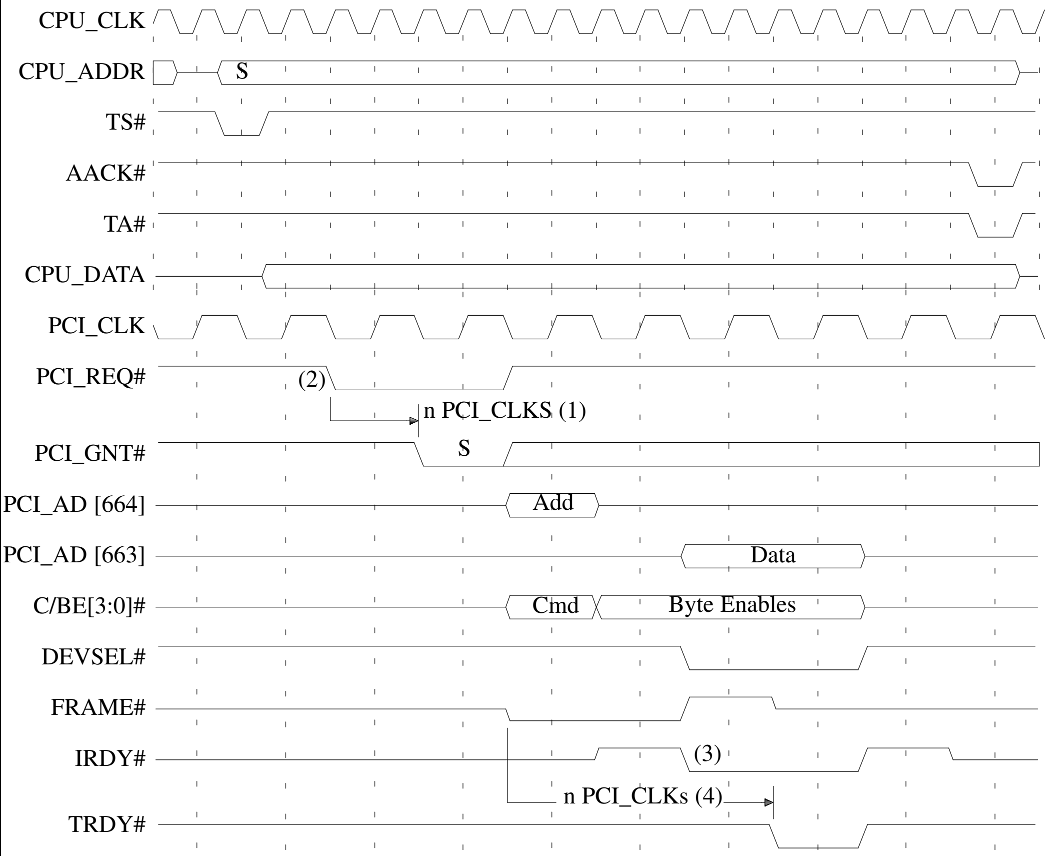
7.2.2.3 Address, Size, Alignment, and Endian Mode

In remote ROM mode, CPU memory writes from 4G – 2M to 4G cause the 660 to generate PCI bus memory write transactions to 4G – 2M to 4G. The 660 does not allow CPU masters to access the rest of the PCI memory space from 2G to 4G.

In remote ROM mode, PCI bus master memory write transactions from 4G – 2M to 4G are ignored by the 660; however, the PCI agent that controls the ROM responds to these transactions. Note that the 660 responds normally to PCI busmaster transactions from 2G to 4G–2M. In contrast, in direct-attach ROM mode, the 660 forwards PCI bus master memory transactions from 2G to 4G (to populated memory locations) to system memory from 0 to 2G.

Remote ROM writes must be one-byte, single-beat transfers.

The endian mode of the system has no net effect on a ROM write because the transfer size is one byte. The address is munged by the CPU and unmunged by the 660. The data comes out of the CPU on the byte lane associated with the munged address, and then is swapped by the 660 to the byte lane associated with the unmunged address. Thus a ROM write in little-endian mode puts the data byte in the same ROM location as does the same ROM write in big-endian mode.



(1) This delay is controlled by the system arbiter.

(2) PCI Bus not parked on CPU. If the PCI bus is parked on the CPU, FRAME# is asserted here.

(3) IRDY# is always asserted as FRAME# is deasserted.
IRDY# is deasserted on the clock that TRDY# is sampled active.

(4) This delay is paced by the remote ROM controller.
The PCI agent may assert TRDY# as soon as it samples FRAME# active.

STOP# not asserted during the transaction.

Figure 7-9. Remote ROM Write

7.3 Related Bridge Control Registers

The two BCRs most closely related to the ROM system are the ROM write BCR and the ROM lockout register. Writes to the ROM are accomplished through the ROM write BCR. Write-protection is provided by means of the ROM lockout BCR.

7.3.1 ROM Write Bridge Control Register

Direct Access FFFF FFF0h	Write Only	Reset NA

This 32-bit, write-only register is used to program the ROM in direct-attach ROM systems (see section 7.1.2). This register must be written by means of a 4-byte transfer. Bits are shown with little-endian labels.

MSb ... LSb

D31	D30	D29	D28	D27	D26	D25	D24	
								Table 7-5

D23	D22	D21	D20	D19	D18	D17	D16	
								Table 7-5

D15	D14	D13	D12	D11	D10	D9	D8	
								Table 7-5

D7	D6	D5	D4	D3	D2	D1	D0	
								Table 7-5

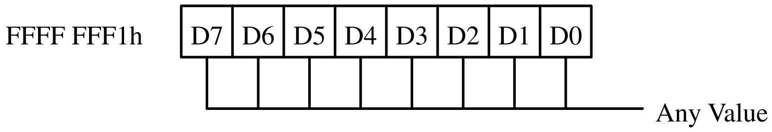
Table 7-5. ROM Write BCR Contents

BCR Byte	Content in Little-Endian System	Content in Big-Endian System
MSB	ROM Data	ROM Address low byte
	ROM Address high byte	ROM Address mid byte
	ROM Address mid byte	ROM Address high byte
LSB	ROM Address low byte	ROM Data

7.3.2 Direct-Attach ROM Lockout BCR

Direct Access FFFF FFF1h	Write Only	Reset NA

After it has been written once, this 8-bit, write-only register prevents direct-attach ROM writes.

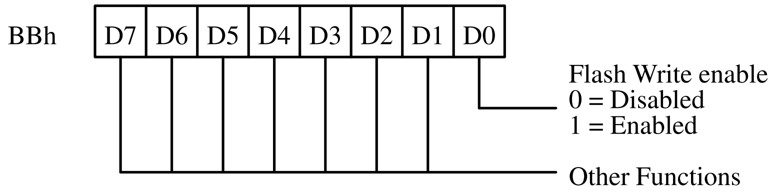


Bits 7:0 Writing any value to the register prevents all future writes to a ROM that is connected directly to the 660 through the PCI_AD lines.

7.3.3 Remote ROM Lockout Bit

The ROM write-protect bit for remote ROM is in the Bridge Chipset Options 2 register (index BBh). While enabled, writes to the remote ROM are forwarded to the PCI memory space. While disabled, writes to the remote ROM are treated as no-ops and an error is signalled. After the first time that the bit is set to 0, it cannot be set back to 1.

Index BBh	Read/Write	Reset to 4Fh



Bit 0 Flash write enable: When the ROM is remotely attached, this bit controls write access to the flash ROM address space (4G – 2M to 4G). When enabled, writes to this space are forwarded to the PCI memory space at the same address. When disabled, writes to this space are treated as no-ops and an error is signalled. After the bit is set to 0 (disabled), it cannot be reset to 1 (enabled).

7.3.4 Other Related BCRs

Bridge Control Register	Index	R/W	Bytes	See
Error Enable 1	Index C0	R/W	1	10.3.37
Error Enable 2	Index C4	R/W	1	10.3.40

7.4 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 7.2.1.3, 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
...

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 with single-beat reads.

Section 8
Exceptions: Resets, Interrupts, Errors, & Test

The 660 bridge provides exception handling support for the system. This section discusses resets, interrupts, error handling, and test modes.

8.1 Resets

The RESET# pin of the 664 must be asserted to initialize the 660 before proper operation commences. The 663 does not have a reset pin. Since the operation of the 663 is controlled by the 664, the entire 660 bridge will be properly initialized by the proper assertion of RESET#.

8.1.1 Reset Timing

The 660 is reset to the correct initial state by the proper assertion of the RESET# input of the 664. The following rules must be followed to ensure correct operation:

1. RESET# must be asserted for at least eight CPU consecutive CPU clocks. This is the minimum RESET# pulse width.
2. Both the CPU and PCI clocks must be running properly during the entire reset interval.
3. Bus activity on the PCI bus must not begin until at least 4 CPU clocks after the deassertion of RESET#.
4. Bus activity on the CPU bus also must not begin until at least 4 CPU clocks after the deassertion of RESET#.
5. Assertion and deassertion of RESET# can be asynchronous for normal operation, but if deterministic operation is required, see section 8.1.4.

All 660 outputs reach their reset state by the second CPU clock after RESET# is first sampled active. The rest of the minimum RESET# pulse width is used by the 660 to initialize internal processes, including setting internal registers and determining the CPU to PCI clock ratio.

Except as noted in section 8.1.2, all 660 outputs maintain their reset state until an external stimulus (CPU bus activity) forces them to change.

8.1.2 Reset State of 660 Pins

The following symbols are used in Table 8-1 and Table 8-2:

- — means the signal is an input. The signal does not have a required state during reset.
- Z means that the pin is tristate (hi–Z) during reset,
- U means the state of the pin during reset is undefined,
- 1 means that the pin is driven to a logic 1 state (hi)
- 0 means that the pin is driven to a logic 0 state (low)

Table 8-1. 664 Pin Reset State

664 Signal	State
AACK#	Z
AOS_RR_MMRS	1
ARTRY#	Z
C2P_WRL_OPEN	1
CAS[7:0]#	1
CPU_ADDR[0:31]	Z
CPU_BUS_CLAIM#	—
CPU_CLK	—
CPU_DATA_OE#	1
CPU_GNT1#	1
CPU_GNT2#	1
CPU_PAR_ERR#	—
CPU_RDL_OPEN	1
CPU_REQ1#	—
CPU_REQ2#	—
CRS_C2PWXS	Z
DBG#	0
DPE#	—
DUAL_CTRL_REF	1
ECC_LE_SEL	1
GBL#	Z
IGN_PCI_AD31	—
INT_CPU#	INT_REQ#
INT_REQ	—
MA[11:0]	1
MCP#	Z
MEM_BE[3:0]	1
MEM_DATA_OE#	1
MEM_ERR#	—
MEM_RD_SMPL	1
MEM_WRL_OPEN	0
MIO_TEST	0
MWS_P2MRXS	Z
NMI_REQ	—
PCI_AD[31:0]	Z
PCI_AD_OE#	1
PCI_C/BE[3:0]#	Z
PCI_CLK	—
PCI_DEVSEL#	Z
PCI_EXT_SEL	1
PCI_FRAME#	Z
PCI_GNT#	—
PCI_IRDY#	Z
PCI_LOCK#	—
PCI_OL_OPEN	1
PCI_OUT_SEL	1
PCI_PAR	Z
PCI_PERR#	Z
PCI_REQ#	1
PCI_SERR#	Z
PCI_STOP#	Z
PCI_TRDY#	Z
RAS[7:0]#	1
RESET#	0
ROM_LOAD	0
ROM_OE#	1
ROM_WE#	1
SBE#	—
SHD#	Z
SRAM_ADS#/ ADDR0	0
SRAM_ALE	1
SRAM_CNT_EN#/ ADDR1	1
SRAM_OE#	1
SRAM_WE#	1
STOP_CLK_EN#	—
TA#	1
TAG_CLR#	0
TAG_MATCH	Z
TAG_VALID	1
TAG_WE#	1
TBST#	Z
TEA#	1
TEST#	1
TS#	Z
TSIZE[0:2]	Z
TT[0:4]	Z
WE[1:0]#	1
XATS#	—

Notes: During reset, INT_CPU# is driven to the inverse of INT_REQ.
For correct operation, TEST# must always be driven high and MIO_TEST must always be driven low.

Table 8-2. 663 Pin Reset State

663 Signal	State	663 Signals	State	663 Signals	State
AOS_RR_MMRS	—	MEM_BE[0:1]	—	PCI_AD[31:0]	Z
C2P_WRL_OPEN	—	MEM_BE[2:3]	—	PCI_AD_OE#	—
CPU_CLK	—	MEM_CHECK[0:7]	Z	PCI_EXT_SEL	—
CPU_DATA[00:63]	Z	MEM_DATA[63:0]	Z	PCI_IRDY#	—
CPU_DATA_OE#	—	MEM_DATA_OE#	—	PCI_OL_OPEN	—
CPU_DPAR[0:7]	Z	MEM_ERR#	U	PCI_OUT_SEL	—
CPU_PAR_ERR#	U	MEM_RD_SMPL	—	PCI_TRDY#	—
CPU_RDL_OPEN	—	MEM_WRL_OPEN	—	ROM_LOAD	—
CRS_C2PWXS	—	MIO_TEST	0	SBE#	U
DUAL_CTRL_REF	—	MWS_P2MRXS	—	TEST#	1
ECC_LE_SEL	—				

Note: For correct operation, TEST# must always be driven high and MIO_TEST must always be driven low.

8.1.3 Configuration Strapping

There are two strapping options for 660 system configuration information which is required before the processor can execute (and which, therefore, cannot be programmed into the 660). Configuration strapping is accomplished by attaching a pullup or pulldown resister to the specified 664 output pin. During reset, the 664 tri-states these outputs, allowing them to assume the level to which they are strapped. When RESET# is deasserted, the 664 reads in the value from these pins.

Table 8-3 shows the strapping options and their associated pins. Pullup resistors should be 10K ohms to 20K ohms. Pull down resistors should be 500 ohms to 2K ohms.

In Table 8-3, 603(e) refers to either a 603 or a 603e.

Table 8-3. Configuration Strapping Options

Function	Pull Up/Down	Pin
Location of ROM	Up = Remote ROM — Down = Direct-Attach ROM	CRS_C2PWXS
603(e) in 1:1 or 3:2 CPU core:bus mode	Down = 603(e) not in 1:1 or 3:2 mode, or 601 or 604. Up = 603(e) in 1:1 or 3:2 CPU core:bus mode.	MWS_P2MRXS

8.1.4 Deterministic Operation (Lockstep Applications)

If fully deterministic operation of the chipset following RESET# is required, then the following items must be considered:

- When RESET# is deasserted, some outputs transition to a different but stable state. This results in requirement #3 in Section 8.1.1, that neither CPU or PCI bus activity is allowed to begin during the first four CPU clocks after the deassertion of RESET#.
- When RESET# is deasserted, the refresh counter begins (and continues) to run, counting the interval between refresh cycles to the memory. There are two ways to start the refresh timer deterministically:
  1. Meet the following timing requirements for RESET#, so that the clock cycle upon which RESET# is deasserted is known:
     - Setup > 4.2ns relative to a rising PCI clock edge.
     - Hold > 0ns relative to a rising PCI clock edge.
  2. Write to the Refresh Timer Divisor Register (index D1–D0) and the Suspend Refresh Timer Register (index D3–D2) to reset the refresh counters. If this is done before any DRAM accesses occur, then no bus activity will have been affected by the unknown state of the counters before this point.
- When RESET# is deasserted, the DUAL_CTRL_REF signal begins toggling. The phase of this toggling never effects any bus operations, and therefore need not be known for deterministic operation of the 660. However, if it is still desirable to control the phase of DUAL_CTRL_REF, then the following timing requirements must be met for the deassertion of RESET#:
  - Setup > 4.4ns relative to a rising CPU clock edge.
  - Hold > 0ns relative to a rising CPU clock edge.

8.2 Interrupts

The 660 features two interrupt inputs, INT_REQ and NMI_REQ. For information on interrupt acknowledge transactions, see section 3.4.6.

8.2.1 INT_REQ and INT_CPU#

As shown in Figure 8-1, the 660 inverts INT_REQ and passes it thru to the CPU as INT_CPU#, and in 601 error reporting mode, also uses INT_CPU# to report certain error conditions.

The only reason that the 660 connects to INT_CPU# is to be able to use it in reporting errors to the 601 CPU. When the bridge is not in 601 error reporting mode, the path though the 660 from INT_REQ to INT_CPU# is functionally an inverting latch. The CPU does not need the interrupt synchronized to the CPU clock, and typical interrupt controllers feature programmable output polarity, so if the target system is not using a 601, then the interrupt can be wired around the 660, without being connected to the 660. In this case, tie the INT_REQ input inactive.

However, if the system CPU is a 601, then the 660 uses INT_CPU# to report CPU bus related errors that cannot be reported with TEA# on the CPU transfer during which they occur (see section 8.4.3). In this case, the 660 asserts INT_CPU# as an interrupt and as a means of reporting errors.

8

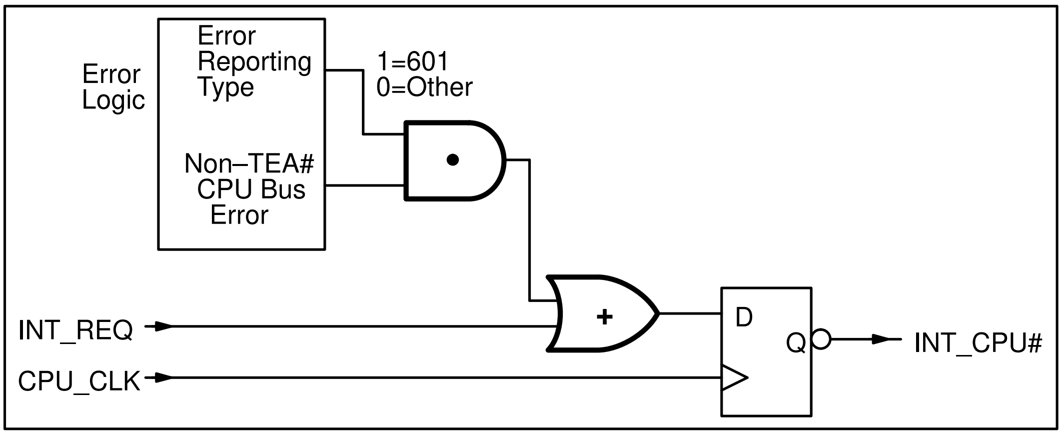


Figure 8-1. Conceptual Block Diagram of INT Logic

8.2.2 NMI_REQ

The 660 considers the NMI_REQ input to be an error indicator. Note that in Figure 8-1, there is no logical connection between NMI_REQ and INT_CPU, except through the error handling logic. See section 8.5.15 for more information on NMI_REQ.

8.2.3 Interrupt-Related Bridge Control Registers

Bridge Control Register	Index	R/W	Bytes	See
Interrupt Acknowledge	BFFF FFF0	R	1	3.4.6

8.3 Error Handling Protocol

The 660 supports the detection and reporting of several types of errors. The errors are reported to the CPU or the PCI and status information is saved in the 660 register set so that error type determination can be done by the CPU.

All errors (except NMI) are related to either a transfer on the CPU bus or a transaction on the PCI bus. Memory errors are related to the CPU bus when they occur as a result of a CPU to memory transfer. Memory errors are related to the PCI bus when they occur as a result of a PCI to memory transfer. Errors detected on the PCI bus when the 660 is the PCI busmaster are related to a CPU bus cycle. Therefore, the only errors related to a PCI bus cycle are errors that are detected while the 660 is a PCI target (PCI to system memory transactions).

Errors related to a CPU bus transfer are reported to the CPU by means of the MCP# or the TEA# signal (and INT_CPU# for 601). Errors related to a PCI bus transaction are reported to the PCI by means of the PCI_PERR# or the PCI_SERR# signals.

Each error that can be detected has an associated mask. If the error is masked, then the detection of that error condition is disabled. There are also assertion masks for the MCP#, TEA#, and PCI_SERR# signals that prevent reporting of any error by means of that signal (these masks do not affect the detection of the error).

Once an error is detected and the appropriate status, address, and control information is saved, the detection of all subsequent error detection is disabled until the current error is reset.

8.3.1 NMI Errors

The assertion of NMI is controlled by external logic, and is treated as an error by the 660. This error is not related to either a CPU transfer or a PCI transaction.

8.3.1.1 Error Handling Protocol

Report error to the CPU by means of MCP#.

Logic external to the 660 typically provides mask and status bits for NMI.

Also see section 8.5.15.

8.3.2 CPU Bus Related Errors

8.3.2.1 Error Types

- **Errors Reported With TEA# (cycle still active)**
  - CPU bus unsupported transfer type
  - CPU bus unsupported transfer size
  - CPU bus XATS# asserted
  - PCI master abort generated (660 is PCI busmaster) during CPU to PCI transaction. This does not include BCR accesses and PCI configuration transactions (e.g., buswalks).
- **Errors Reported With MCP# (cycle active or completed)**

  These errors never cause a TEA#.

  - CPU data bus parity error
  - CPU bus write to locked flash
  - CPU bus memory select error
  - Memory parity error during CPU to memory transfer
  - Memory single-bit ECC error trigger exceeded during CPU to memory transfer
  - Memory multi-bit ECC error during CPU to memory transfer
  - L2 cache parity error
  - PCI bus data parity error (660 is PCI busmaster) during CPU to PCI transaction
  - PCI target abort received (660 is PCI busmaster) during CPU to PCI transaction.

8.3.2.2 Error Handling Protocol

If the error is masked, do not detect the error.

If the error is detected, perform the following steps:

1. Set status bit indicating error type.
2. Set status bit indicating error during CPU cycle.
3. Save CPU address and control bus values.
4. Report error to the CPU. (Reported by means of TEA# if the CPU cycle is still active or by means of MCP# if the CPU cycle has ended.)

PCI bus data parity errors also cause PCI_PERR# to be asserted.

There is a status bit (PCI Status Register bit 15) that is set whenever any type of PCI bus parity error is detected. The setting of this status bit is not maskable.

8.3.3 PCI Bus Related Errors

During CPU to PCI transactions, the 660 does not check PCI bus address parity.

8.3.3.1 Error Types

During a PCI to memory transaction (in which the 660 is the PCI target):

- PCI bus address parity error
- PCI bus data parity error (see Section 8.3.3.3)
- PCI memory select error (see Section 8.5.5)
- Memory parity error
- Memory single–bit ECC error trigger exceeded
- Memory multi-bit ECC error

8.3.3.2 Error Handling Protocol

If the error is masked, the 660 does not detect the error. If the error is detected, perform the following steps.

1. Set status bit indicating error type.
2. Set status bit indicating error during PCI cycle.
3. Save PCI address and control bus values, in general.
4. Report error to the PCI. If the error is a PCI bus data parity error, then report by means of PCI_PERR#. If the error is not a PCI bus data parity error then report by means of PCI_SERR#.
5. If the PCI cycle is still active (not the last data phase), then target abort the cycle. No error is reported to the CPU

The 660 can be enabled to report PCI bus data parity errors with PCI_SERR#. This method should only used if it is determined that PCI_PERR# is not supported by some (or all) of the PCI masters in the system.

8.3.3.3 PCI Bus Data Parity Errors

While the 660 is the PCI busmaster (during CPU to PCI transactions):

- During reads, the 660 monitors the PCI_AD (and C/BE# and PCI_PAR) lines to detect data parity errors during the data phases. If an error is detected, the 660 asserts PCI_PERR#. Unless masked, the 660 will report the error to the CPU bus using MCP#. This error does not cause the 660 to alter the PCI transaction in any way.
- During writes, the 660 monitors PCI_PERR# to detect data parity errors that are detected by the target. Unless masked, the 660 will report the error to the CPU bus using MCP#. This error does not cause the 660 to alter the PCI transaction in any way.

While the 660 is the PCI target (during PCI to memory transactions):

- During reads, the 660 does not monitor PCI_PERR#, and so will not detect a data parity error.

- During writes, the 660 monitors the PCI_AD (and C/BE# and PCI_PAR) lines to detect data parity errors during the data phases. If an error is detected, the 660 asserts PCI_PERR#. The 660 will not report the error to the CPU bus. This error does not cause the 660 to alter the PCI transaction in any way.

8.3.4 All Ones Select

- For a CPU to DRAM read to a non-populated memory location, data driven onto the CPU bus is undefined, and it may be whatever is floating on the memory data bus.
- During a CPU to PCI memory or I/O transaction to which there is no response (no DEVSEL#), the 660 master aborts the PCI transaction, asserts TEA# to the CPU, and (during a read) drives all ones on the CPU data bus.
- During a CPU to PCI configuration read to which there is no response (no DEVSEL#), the 660 master aborts the PCI transaction, asserts TEA# to the CPU, and drives all ones on the CPU data bus. TEA# assertion should be disabled during bus walks.

8.4 Error Reporting Protocol

In general, when the 660 recognizes an error condition, it sets various status BCRs, saves address and control information (for most bus related errors), disables further error recognition (until the current error is cleared), and reports the error to either the CPU or PCI bus.

Unless otherwise noted, the 660 takes no further error handling action, but relies on the CPU/software or PCI agent to take the next step in the error handling procedure. The 660 continues to react appropriately to CPU and PCI bus traffic, the state of the memory controller is unchanged, current and pipelined CPU and PCI transactions are unaffected, and the behavior and state of the 660 is unaffected.

For example, if a memory parity error is reported to the CPU using MCP#, and the CPU does not respond to the MCP#, then the 660 will in all ways continue to behave as if the MCP# had not been asserted. However, various BCRs will contain the error status and address information, and further error recognition will be disabled until the CPU resets the error in the 660 BCRs.

8.4.1 Error Reporting With MCP#

In general, the following errors are reported to the CPU using MCP#:

- NMI errors
- Errors that occur because of a CPU transfer (see Section 8.3.2.1).

The 660 reports an error with MCP# by asserting MCP# to the CPU bus for 2 CPU clocks. The 660 does not itself take any other action. All current and pipelined CPU and PCI bus transactions are unaffected. The state of the memory controller is unaffected. The assertion of MCP# does not cause any change in the behavior or state of the 660.

8.4.2 Error Reporting With TEA#

CPU bus related errors that are detected while the CPU is running a cycle that can be terminated immediately are reported using TEA# (see Section 8.3.2.1). Errors reported in this way are a direct result of the CPU transfer that is currently in progress. For example, when the 660 detects a transfer size error, it terminates the CPU transfer with TEA# instead of with TA#.

The 660 reports an error with TEA# by asserting TEA# to the CPU in accordance with the PowerPC bus protocol. The data beat on which TEA# is asserted becomes the final data beat. The 660 does not itself take any other action. All other current and pipelined CPU and PCI bus transactions are unaffected. The state of the memory controller is unaffected. The assertion of TEA# does not cause any other change in the behavior or state of the 660.

8.4.3 Error Reporting to 601 CPU

The 601 CPU does not have an MCP# pin. In cases where MCP# would be asserted, the 660 does the following:

1. Asserts INT_CPU#.
2. On the next PCI interrupt acknowledge from the CPU, no interrupt acknowledge is generated to the PCI. Instead, the CPU cycle is ended by asserting TEA# (and returning a value of FFh).

The 660 must be set up to operate in this manner when the CPU is a 601. 601 error reporting mode is enabled by bit 4 in the Bridge Chipset Options 3 BCR (index D4h).

8.4.4 Error Reporting With PCI_SERR#

The 660 asserts PCI_SERR# for one PCI clock when a PCI bus non-data-parity error (system error) is detected and the 660 is a PCI target. As is the case with the other error reporting signals, the 660 may perform various operations in response to a detected error condition, but it does not automatically perform further actions just because it asserts PCI_SERR#

The 660 does not monitor PCI_SERR# as driven by other PCI agents. PCI_SERR# is not an input to the 660.

8.4.5 Error Reporting With PCI_PERR#

The 660 asserts PCI_PERR# for one PCI clock to report PCI bus data parity errors that occur while the 660 is receiving data; during PCI to memory writes and CPU to PCI reads. The 660 asserts PCI_PERR# in conformance to the PCI specification.

8.5 Error Handling Details by Error Type

8.5.1 CPU Bus Transfer Type or Size Error

This error is generated when the CPU generates a bus operation that is not supported by the 660 (see Table 8-4). An error is not generated if the cycle is claimed by another CPU device (CPU_BUS_CLAIM# asserted).

Table 8-4. Invalid CPU Bus Operations

TT[0:4]	Operation
1000x	Reserved
1011x	Reserved

Only the following transfer sizes are supported. Other transfer sizes are not supported.

- 1-byte to 8-byte single-beat reads or writes to memory within a double-word boundary
- Burst reads or writes to memory (32 bytes, aligned to double-word)
- 1-byte to 4-byte single-beat reads or writes to the PCI bus that do not cross a 4-byte boundary
- 8-byte single-beat writes to the PCI bus within an 8-byte boundary
- All accesses not to memory or PCI with sizes of 1 to 4 bytes within a 4-byte boundary
- ROM reads support the same sizes as memory.

Transfer type and size errors can be controlled by the indexed register set. The mask is at register C0h bit 0. If an error is detected, status bits at register C1h bits 1:0 are set to 10. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. Transfer type and size errors are reset by writing a 1 to register C1h bit 0 or register C1h bit 1. The indexed register set uses the same mask and error reset bits for XATS# that it uses for unsupported transfer types.

Transfer type and size errors can also be controlled by the 650-compatible register set. The mask cannot be controlled by means of this register set. If an error is detected, the status bit at 8000 0844h bit 0 is cleared. The address is saved at BFFF EFF0h. This error can be reset by reading BFFF EFF0h. Note that the 650-compatible register set does not differentiate between XATS# errors and unsupported transfer type errors.

8.5.2 CPU Bus XATS# Asserted Error

This error is generated when the CPU asserts the XATS# signal.

The XATS# error can be controlled by the indexed register set. The mask is at register C0h bit 0. If an error is detected, the status bits at register C1h bits 1:0 are set to 01. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C1h bit 0 or register C1h bit 1. The indexed register set uses the same mask and error reset bits for XATS# and for unsupported transfer types.

This error can also be controlled by the 650-compatible register set. The mask cannot be controlled by means of this register set. If an error is detected, the status bit at 8000 0844h bit 0 is cleared. The address is saved at BFFF EFF0h. This error can be reset by reading BFFF EFF0h. The 650-compatible register set does not differentiate between XATS# errors and unsupported transfer type errors.

8.5.3 CPU Data Bus Parity Error

This error is generated when a parity error on the CPU data bus is detected during a transfer between the CPU and the 660. The full CPU data bus is always checked for parity regardless of which bytes lanes actually carry valid data. The parity is odd, which means that an odd number of bits, including the parity bit, are driven high. The 660 directly checks the parity during CPU write cycles. The 660 detects CPU bus parity errors by sampling the DPE# signal from the CPU during CPU read cycles.

This error is also generated when an L2 cache data parity error (see section 8.5.9) is detected.

CPU_DPAR[0] indicates the parity for CPU_DATA[0:7]. CPU_DPAR[1] indicates the parity for CPU_DATA[8:15] and so on.

This error can be controlled by the indexed register set. The mask is at register C4h bit 2. If an error is detected, the status bit at register C5h bit 2 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h, the CPU control is saved in register C3h, and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C5h bit 2.

This error cannot be controlled by means of the 650-compatible register set.

8.5.4 CPU Bus Write to Locked Flash

This error is generated when the CPU attempts to write to flash memory when write to flash ROM has been disabled (locked out). If the flash ROM is directly attached to the 660 (see configuration strapping), CPU writes to FFFF FFF0h are detected as an error if writing has been locked out by means of 660 compatible register FFFF FFF1h (see note in Sections 7.1.2 and 7.1.2.2). If the flash is remotely attached then CPU writes to the 4G – 2M to 4G address space are detected as an error if writing has been locked out by means of register BBh bit 0.

This error can be controlled by the indexed register set. The mask is at register C4h bit 0. If an error is detected, the status bit at register C5h bit 0 is set. Register C7h bit 4 is cleared to indicate error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C5h bit 0.

This error cannot be controlled by means of the 650-compatible register set.

8.5.5 Memory Select Error

This error is generated if a device addresses the system memory space (CPU addresses from 0 to 2G and PCI addresses from 2G to 4G) when memory is not present at that address. The 660 only claims PCI accesses by asserting PCI_DEVSEL# when the access is to an address where memory is present.

The 660 disconnects PCI burst cycles at 1M boundaries. This ensures that a PCI master cannot begin a transfer at an address where memory is present and then burst (incrementing the address) to an address where memory is not present; therefore, the memory select error is never generated on PCI accesses to system memory.

The 660 always and totally ignores bursts to unpopulated memory locations (no bits are set, no error checking, and no change in machine state) (e.g., does not assert DEVSEL#). PCI protocol requires that the initiating busmaster master abort the transaction.

For the PCI bus side, if a PCI busmaster initiates a memory transaction that is not to a populated memory location, the 660 ignores it. Any other PCI bus agent is then free to DEVSEL# the transaction as a target. This behavior enables PCI peer to peer memory transactions.

Note, however, that when ECC mode is enabled, memory select error detection is disabled, and access is made to an address between 0 and 1G in which no memory resides, then an ECC error may be detected. To avoid this error, in ECC mode, accesses in the 0 to 1G range above populated memory must be done with multi-bit and single-bit errors disabled.

The memory select error can be controlled by the indexed register set. The mask is at register C0h bit 5. If an error is detected, the status bit at register C1h bit 5 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C1h bit 5.

This error cannot be controlled by means of the 650-compatible register set.

8.5.6 System Memory Parity Error

When memory is being operated in parity mode, this error is generated if a parity error is detected during a read from system memory. Memory parity is odd, which means that an odd number of bits including the parity bit are driven high.

MEM_CHECK[0] indicates the parity for MEM_DATA[7:0]. MEM_CHECK[1] indicates the parity for MEM_DATA[15:8] and so on.

The system memory parity error can be controlled by the indexed register set. The mask is at register C0h bit 2. If an error is detected, the status bit at register C1h bit 2 is set.

If the parity error occurred while the CPU was accessing memory, then register C7h bit 4 is cleared to indicate the error occurred during a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5.

If the parity error occurred while the PCI was accessing memory, then register C7h bit 4 is set to indicate the error occurred during a PCI cycle. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

This error can be reset by writing a 1 to register C1h bit 2. Note that register locations listed above are used to indicate single-bit ECC errors if the memory is being operated in ECC mode.

This error can also be controlled by means of the register in the 650-compatible register set. The mask cannot be controlled by means of this register set. If an error is detected, the status bit at 8000 0840h bit 0 is cleared. The address is saved at BFFF EFF0h. This error can be reset by reading BFFF EFF0h.

8.5.7 System Memory Single-Bit ECC Error

When memory is being operated in ECC mode, single-bit errors are detected and corrected. Since single-bit errors are corrected, generally no error reporting is necessary. But when a single-bit error is detected, the single–bit error counter register (register B8h) is incremented. If the count in the single-bit error counter register exceeds the value in the single-bit error trigger level register (B9h), the system memory address is saved in the single-bit ECC error address register (register CCh), then the 660 generates a single-bit ECC trigger exceeded error. The 660 only latches the address in register CC while the single-bit error trigger level is exceeded, and it is not updated unless the trigger level has been exceeded and the transaction caused a single-bit ECC error.

Under certain conditions, this register may fail to capture the address when a single-bit ECC error occurs.

The trigger exceeded error can be controlled by the indexed register set. The mask is at register C0h bit 2. If an error is detected, the status bit at register C1h bit 2 is set.

If the error occurs while the CPU is accessing memory, register C7h bit 4 is cleared to indicate the error occurred during a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is save in register C7h bit 5.

If the error occurs while the PCI is accessing memory, register C7h bit 4 is set to indicate the error occurred during a PCI cycle. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

This error can be reset by writing a 1 to register C1h bit 2. Note that register locations listed above are used to indicate memory parity errors if the memory is being operated in parity mode.

This error cannot be controlled by means of the 650-compatible register set.

8.5.8 System Memory Multi-Bit ECC Error

When memory is being operated in ECC mode, this error is generated if a multi-bit ECC error (uncorrectable) is detected during a read from system memory.

The multi-bit ECC error can be controlled by the indexed register set. The mask is at register C0h bit 3. If an error is detected, the status bit at register C1h bit 3 is set.

If the error occurs while the CPU is accessing memory, then register C7h bit 4 is cleared to indicate the error occurred during a CPU cycle. The CPU bus address is saved in register C8h. The CPU control is saved in register C3h, and the CPU number is saved in bit 7 of register C7h.

If the error occurs while the PCI is accessing memory, then register C7h bit 4 is set to indicate the error occurred during a PCI cycle. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

This error can be reset by writing a 1 to register C1h bit 3.

This error cannot be controlled by means of the 650-compatible register set.

8.5.9 L2 Cache Parity Error

This error is generated when a parity error is detected during a CPU read from the L2 cache. The parity is checked by the CPU which drives DPE# to the 660. When this error is detected, the 660 indicates both this error and a CPU bus data parity error.

This error can be controlled by the indexed register set. The mask is at register C4h bit 3. If an error is detected, the status bit at register C5h bit 3 is set. Register C7h bit 4 is cleared to indicate error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C5h bit 3.

This error can also be controlled by means of a register in the 650-compatible register set. The mask cannot be controlled by means of this register set. If an error is detected, the status bits at 8000 0842h bit 0 and 8000 0843h bit 0 is cleared. The address is not saved in a 650-compatible register (register BFFF EFF0h is undefined). This error can be reset by reading 8000 0843h.

8.5.10 PCI Bus Data Parity Error While PCI Master

This error is generated when a PCI bus data parity error is detected during a CPU to PCI transaction. The 660 checks parity during read cycles and samples PCI_PERR# during write cycles. The bridge asserts PCI_PERR# if a parity error is detected on a read cycle. The PCI bus uses even parity, which means that an even number of bits including the parity bit are driven high.

This error can be controlled by the indexed register set. The mask is at register 04h bit 6. If an error is detected, the status bit at register 06h bit 8 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register 06h bit 8.

When this error is detected, the status bit at register 06h bit 15h is set, regardless of the state of the mask at register 04h bit 6. The status bit at register 06h bit 15 is set by all types of PCI bus parity errors. This bit is cleared by writing a 1 to register 06h bit 15.

This error cannot be controlled by means of the 650-compatible register set.

Unless masked, the 660 will report this error to the CPU as a PCI bus data parity error while PCI master, using MCP#.

8.5.11 PCI Target Abort Received While PCI Master

This error is generated when a target abort is received on the PCI bus during a cycle which is mastered by the 660 for CPU access to the PCI bus.

This error can be controlled by the indexed register set. The mask is at register C0h bit 7. If an error is detected, the status bit at register 06h bit 12 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is save in register C7h bit 5. This error can be reset by writing a 1 to register 06h bit 12.

This error cannot be controlled by means of the 650-compatible register set.

8.5.12 PCI Master Abort Detected While PCI Master

This error is generated when a master abort is detected on the PCI bus during a cycle which is mastered by the 660 for CPU access to the PCI bus. Master aborts occur when no target claims a PCI memory or I/O cycle—PCI_DEVSEL# is never asserted.

Note that some operating systems intentionally access unused memory and I/O addresses to determine what devices are located on the PCI bus. These operating systems do not expect an error (the 660 asserts TEA#) to be generated by these accesses. When using such an operating system, it is necessary to leave this error masked.

During CPU to PCI configuration transactions (common during bus walks) and during BCR transactions, the 660 will not assert TEA# due to this condition, but it will master abort the PCI cycle according to the protocol.

This error can be controlled by the indexed register set. The mask is at register C4h bit 4. If an error is detected, the status bit at register 06h bit 13 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register

C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register 06h bit 13.

This error cannot be controlled by means of the 650-compatible register set.

8.5.13 PCI Bus Address Parity Error While PCI Target

This error is generated when a parity error is detected during the address phase of a PCI access where the 660 is the PCI target of a PCI access to system memory.

This error can be controlled by the indexed register set. The mask is at register 04h bit 6. This error does not have an explicit status bit to indicate its occurrence. However, the following status bits are set:

1. Register 06h bit 14 is set to indicate that PCI_SERR# has been asserted. This bit is cleared by writing a 1 to register 06h bit 14. (BCR04 b8 must be 1 to enable SERR# assertion due to PCI bus address parity errors.)
2. Register 06h bit 11 is set to indicate signalled target abort if the cycle was target aborted. This bit is cleared by writing a 1 to register 06h bit 11.
3. Register 06h bit 15 is set to indicate a PCI bus parity error regardless of the state of the mask at register 04h bit 6. Note that this bit is set by all types of PCI bus parity errors. This bit is cleared by writing a 1 to register 06h bit 15.

Register C7h bit 4 is set to indicate an error on a PCI cycle. If the 660 asserts SERR# due to this error, then the PCI address is saved in register C8h, and the PCI control is saved in register C7h; else, these registers are not updated due to this error.

This error cannot be controlled by means of the 650-compatible register set.

8.5.14 PCI Bus Data Parity Error While PCI Target

This error is generated when a PCI bus data parity error is detected during a PCI to memory write transaction. The PCI bus uses even parity, which means that an even number of bits including the parity bit are driven high.

This error can be controlled by means of the registers in the indexed register set. The mask is at register 04h bit 6. This error does not have an explicit status bit to indicate its occurrence. However, the following status bits are set:

1. Register 06h bit 11 is set to indicate signalled target abort if the cycle was target aborted. This bit is cleared by writing a 1 to register 06h bit 11.
2. Register 06h bit 15 is set to indicate a PCI bus parity error regardless of the state of the mask at register 04h bit 6. Note that this bit is set by all types of PCI bus parity errors. This bit is cleared by writing a 1 to register 06h bit 15.
3. Register 06h bit 14, which indicates PCI_SERR#, is set if the mask at register C0h bit 6 is disabled (cleared). Note that the mask at register C0h bit 6 allows the 660 to signal PCI_SERR# in addition to PCI_PERR# for this error. This bit is cleared by writing a 1 to register 06h bit 14.

Register C7h bit 4 is set to indicate an error on a PCI cycle. If the 660 asserts SERR# due to this error, then the PCI address is saved in register C8h, and the PCI control is saved in register C7h; else, these registers are not updated due to this error.

This error cannot be controlled by means of the 650-compatible register set.

During PCI to memory reads, the 660 does not monitor PCI_PERR# to detect data parity errors. Therefore this error is never generated during PCI to memory reads.

This error is not reported to the CPU.

8.5.15 NMI_REQ Asserted Error

This error is generated when the NMI input is sampled asserted by the 660. External logic can assert this signal for any type of catastrophic error it detects. The external logic should also assert this signal if it detects PCI_SERR# asserted. The 660 does not treat NMI_REQ as an interrupt, but as an error indicator.

NMI is handled somewhat differently from the bus related error sources.

- There are no 660 BCRs associated with NMI_REQ. The external logic that asserted NMI to the 660 provides mask and status information.
- The NMI_REQ input contains no edge detection logic. The 660 has no memory of any previous state of NMI_REQ.
- In general, the assertion of NMI_REQ has no effect on any other processes in the 660.
- MCP# is generally asserted continuously while NMI_REQ is sampled valid, however:
  - If an error was detected before the NMI_REQ was detected, then the error handling logic will not sample the NMI_REQ input (and thus will not detect it) until the previous error is cleared using the appropriate BCRs.
    - If the NMI_REQ is deasserted before the previous error is cleared, the NMI_REQ will be lost.
    - If NMI_REQ is still asserted when the previous error is cleared, then the NMI_REQ will be sampled asserted, and MCP# will begin to be asserted.
  - The 660 will not assert MCP# while NMI_REQ is active unless MCP# assertion is enabled in BCR(BA) bit 0. As before, if NMI_REQ is active when MCP# assertion is enabled, then MCP# will be asserted.
  - Unlike with the bus related error sources, when the 660 samples NMI_REQ valid, it does not disable further error detection. Thus PCI and CPU bus related errors will still be detected and handled in the normal fashion. However, if the detected bus related error causes MCP# to be asserted (for 2 CPU clocks) then at the end of the second CPU clock, MCP# will be and remain deasserted until the current error is cleared using the appropriate BCRs, even if NMI_REQ is still asserted.

8.5.16 Error-Related Bridge Control Registers

Bridge Control Register	Index	R/W	Bytes	See
System Control 81C	8000 081C	R/W	1	10.2.2.3
Memory Parity Error Status	8000 0840	R	1	10.2.2.5
L2 Error Status	8000 0842	R	1	10.2.2.6
L2 Parity Error Read and Clear	8000 0843	R	1	10.2.2.7
Unsupported Transfer Type Error	8000 0844	R	1	10.2.2.8
System Error Address	BFFF EFF0	R	4	10.2.2.13
PCI Command	Index 04 – 05	R/W	2	10.3.5
PCI Device Status	Index 06 – 07	R/W	2	10.3.6
Single-Bit Error Counter	Index B8	R/W	1	10.3.33
Single-Bit Error Trigger Level	Index B9	R/W	1	10.3.34
Bridge Options 1	Index BA	R/W	1	10.3.35
Error Enable 1	Index C0	R/W	1	10.3.37
Error Status 1	Index C1	R/W	1	10.3.38
CPU Bus Error Status	Index C3	R	1	10.3.39
Error Enable 2	Index C4	R/W	1	10.3.40
Error Status 2	Index C5	R/W	1	10.3.41
PCI Bus Error Status	Index C7	R/W	1	10.3.42
CPU/PCI Error Address	Index C8 – CB	R/W	4	10.3.43
Single-Bit ECC Error Address	Indx CC – CF	R/W	4	10.3.44
Bridge Chip Set Options 3	Index D4	R/W	1	10.3.46

8

8.6 Test Modes

The TEST# and MIO_TEST pins of the 663 and 664 are intended for use by the IBM manufacturing process only. The inclusion of the following information in this section is the total extent to which IBM supports the use of these pins by external customers.

8.6.1 LSSD Test Mode

Tie the TEST# input of both the 663 and the 664 high during normal operation. Do not allow these signals to be casually asserted. Caution is advised in the use of LSSD test mode.

LSSD test mode is enabled on the 663 (asynchronously) by asserting TEST# to the 663. In the same way, LSSD test mode is enabled on the 664 (asynchronously) by asserting TEST# to the 664. In LSSD test mode, the 663 and 664 pins shown in Table 8-5 are redefined to become LSSD test mode pins. These pins have LSSD functions only while the 663 (or 664) is in LSSD test mode. Otherwise the pins perform normally. IBM uses LSSD test mode to verify the logical operation of the 663 and the 664.

Table 8-5. LSSD Test Mode Pin Definitions

Test Pin Name	664 Pin	664 Pin Normal Name	663 Pin	663 Pin Normal Name
TEST_ACLK#	194	MEM_ERR#	149	ECC_LE_SEL
TEST_BCLK#	129	XATS#	170	DUAL_CTRL_REF
TEST_CCLK#	133	DPE#	163	MEM_BE[2]
SCAN_IN	192	CPU_PAR_ERR#	145	MEM_DATA_OE#
SCAN_OUT	47	ROM_OE#	174	CPU_PAR_ERR#
RI#	56	NMI_REQ	161	MEM_BE[0]
DI#	151	STOP_CLK_EN#	162	MEM_BE[1]

In LSSD test mode, never assert more than one of TEST_ACLK#, TEST_BCLK#, TEST_CCLK#, and RESET# at the same time, as this may damage the device by provoking excessive internal current flows.

In LSSD test mode, the DI# pin controls the drivers of the 663 (and the 664). Assertion of the DI# pin asynchronously causes all of the 663 (or 664) output drivers (push–pull, tri–state, open–driver, or bi–directional) to be tristated.

In LSSD test mode, the RI# pin controls the receivers of the 663 (and the 664). Assertion of RI# causes all of the 663 (or 664) inputs to report a certain pattern to the internal logic. This has no effect on the external operation of the device that can be used by an external customer.

The 660 must be reset properly after leaving LSSD test mode in order to assure correct normal mode operation.

8.6.2 MIO Test Mode

Tie the MIO_TEST input of both the 663 and the 664 low during normal operation. Do not allow these signals to be casually asserted. IBM does not support any use of MIO test mode by external customers.

MIO test mode is enabled on the 663 (asynchronously) by asserting MIO_TEST to the 663. In the same way, MIO test mode is enabled on the 664 (asynchronously) by asserting MIO_TEST to the 664. IBM uses LSSD test mode to verify the voltage switching levels of the inputs of the 663 and the 664.

8

Section 9
Endian Mode

Data represented in memory or media storage is said to be in big endian (BE) order when the most significant byte is stored at the lowest numbered address, and less significant bytes are at successively higher numbered addresses.

Data is stored in little endian (LE) order when it is stored with the order of bytes reversed from that of BE order. In other words, the most significant byte is stored at the highest numbered address. The endian ordering of data never extends past an 8-byte group of storage.

PowerPC systems normally operate with big endian (BE) byte significance, which is the native mode of the PowerPC CPU. Internally, the CPU always operates with big endian addresses, data, and instructions, which is ideal for operating systems such as AIX™, which store data in memory and on media in big endian byte significance. In BE mode, neither the CPU nor the 660 perform address or data byte lane manipulations that are due to the endian mode. Addresses and data pass ”straight through” the CPU bus interface and the 660.

The CPU also features a mode of operation designed to efficiently process code and operating systems such as WindowsNT™, which store data in memory and on media in LE byte significance. The 660 also supports this mode of operation.

When the 660 is in little endian mode, data is stored in memory with LE ordering. The 660 has hardware to select the proper bytes in the memory and on the PCI bus (via address transforms), and to steer the data to the correct CPU data lane (via a data byte lane swapper).

Table 9-1 summarizes the operation of the PowerPC system in the two different modes.

Table 9-1. Endian Mode Operations

Mode	What the CPU Does	What the 660 Does
Big Endian (BE)	No munge, no shift	No unmunge, no swap
Little Endian (LE)	Address Munged & Data Shifted	Address Unmunged & Data Swapped

In BE mode, the CPU emits the address unchanged, and does not shift the data. The 660 passes the address and data through to the target without any changes (that are due to endian mode).

In LE mode, the CPU transforms (munges) the three least significant address bits, and shifts the data on the byte lanes to match the munged address. In LE mode, the 660 unmunges the address and swaps the data on the byte lanes.

9.1 What the CPU Does

9.1.1 The CPU Address Munge

The CPU assumes that the significance of memory is BE. When it operates in LE mode, it internally generates the same effective address as the LE code would generate. Since it assumes that the memory is stored with BE significance, it transforms (munges) the three low order addresses when it activates the address pins. For example, in the 1-byte transfer case, address 7 is munged to 0, 6 to 1, 5 to 2, and so on. Table 9-2 shows the address transform rules for the allowed LE mode transfer sizes.

Table 9-2. CPU LE Mode Address Transform

Transfer Size	Address Transform
8	None
4	Physical Address[29:31] XOR 100 => A[29:31]
2	Physical Address[29:31] XOR 110 => A[29:31]
1	Physical Address[29:31] XOR 111 => A[29:31]

9.1.2 The CPU Data Shift

The data transfer occurs on the byte lanes identified by the address pins and transfer size (TSIZ) pins in either BE or LE mode. In LE mode, the CPU shifts the data from the byte lanes pointed to by the unmunged address, over to the byte lanes pointed to by the munged address. This shift is linear in that it does not rotate or alter the order of the bytes, which are now in the proper set of byte lanes. Note that the individual bytes are still in BE order.

9.2 What the 660 Does

While the reference design is operating properly, data is stored in system memory in the same endian mode as the mode in which the CPU operates. That is, the byte significance in memory is BE in BE mode and it is LE in LE mode. Because of this, hardware is included in the 660 that (in LE mode) will swap the data bytes to the correct byte lanes, and that will transform (or un-munge) the address coming from the CPU.

9.2.1 The 660 Address Unmunge

In LE mode, the 660 unmunges address lines A[29:31]. This unmunge merely applies the same XOR transformation to the three low-order address lines as did the CPU. This effectively reverses the effect of the munge that occurs within the CPU. For example, if the CPU executes a one-byte load coded to access byte 0 of memory in LE mode, it will munge its internal address and emit address A[29:31] = 7h. The 660 will then unmunge the 7 on A[29:31] back to 0, and use this address to access memory.

9.2.2 The 660 Data Swapper

The 660 contains a byte swapper. As shown in Figure 9-1, the byte swapper is placed between the CPU data bus and the memory and PCI data busses. This allows the byte lanes to be swapped between the CPU bus and the PCI bus, or between the CPU bus and memory, but not between the PCI bus and memory. Thus, when a PCI busmaster accesses memory, the 660 does not change either the address or the data location to adjust for endian mode. In either mode, data is stored or fetched from memory at the address presented on the PCI bus.

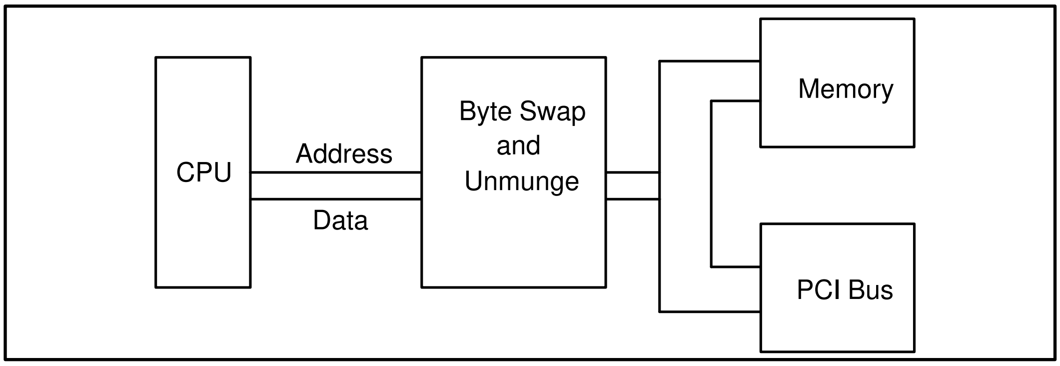


Figure 9-1. Data Flow Location of 660 Byte Swapper

In BE mode, the 660 byte swapper is off, and data passes through it with no changes. In LE mode, the byte swapper is on, and the order of the byte lanes is rotated (swapped) about the center. As shown in Table 9-3, the data on CPU byte lane 0 is steered to memory byte lane 7, the data on CPU byte lane 1 is steered to memory byte lane 6, and so on. During reads, the data flows in the opposite direction over the same paths. Notice that the swapping pattern is always the same, and is not affected by the transfer size, as is the munge operation and the data shift inside the CPU.

9

Table 9-3. 660 Endian Mode Byte Lane Steering

CPU Byte Lane	BE Mode Connection		LE Mode Connection	
	Memory Byte Lane	PCI Byte Lane	Memory Byte Lane	PCI Byte Lane
0 CPU_DATA[0:7]	0 MEM_DATA[7:0]	0 PCI_AD[7:0]	7 MEM_DATA[63:56]	7* PCI_AD[31:24]
1 CPU_DATA[8:15]	1 MEM_DATA[15:8]	1 PCI_AD[15:8]	6 MEM_DATA[55:48]	6* PCI_AD[23:16]
2 CPU_DATA[16:23]	2 MEM_DATA[23:16]	2 PCI_AD[23:16]	5 MEM_DATA[47:40]	5* PCI_AD[15:8]
3 CPU_DATA[24:31]	3 MEM_DATA[31:24]	3 PCI_AD[31:24]	4 MEM_DATA[39:32]	4* PCI_AD[7:0]
4 CPU_DATA[32:39]	4 MEM_DATA[39:32]	4* PCI_AD[7:0]	3 MEM_DATA[31:24]	3 PCI_AD[31:24]
5 CPU_DATA[40:47]	5 MEM_DATA[47:40]	5* PCI_AD[15:8]	2 MEM_DATA[23:16]	2 PCI_AD[23:16]
6 CPU_DATA[48:55]	6 MEM_DATA[55:48]	6* PCI_AD[23:16]	1 MEM_DATA[15:8]	1 PCI_AD[15:8]
7 CPU_DATA[56:63]	7 MEM_DATA[63:56]	7* PCI_AD[31:24]	0 MEM_DATA[7:0]	0 PCI_AD[7:0]

* In this table, PCI byte lanes 3:0 refer to the data bytes associated with PCI_C/BE[3:0]# when the third least significant bit of the target PCI address (PCI_AD[29]) is 0. PCI byte lanes [7:4] refer to the data bytes associated with PCI_C/BE[3:0]# when PCI_AD[29] is 1.

9.3 Bit Ordering Within Bytes

The LE convention of numbering bits is followed for the memory and PCI busses, and the CPU busses are labeled in BE nomenclature. The various busses are connected to the 660 with their (traditional) native significance maintained (BE for CPU, and LE for PCI and memory), so that MSb connects to MSb and so on. The bit paths between the CPU and memory data busses are shown in Table 9-4 for both BE and LE mode operation.

Except for the three least significant bits, the address busses are not affected by endian mode, because the effect of endian mode never extends past an 8-byte group of data.

The various nets are not renamed depending on the endian mode of the system. CPU_DATA[37] remains CPU_DATA[37], in both BE and LE modes.

The effects of endian mode do not extend below the data byte level. The bits within a byte are never rearranged due to endian mode. Note that the various busses are connected so as to preserve the significance of the bus, not the nomenclature. Thus Table 9-4 shows that in BE mode, CPU_DATA[0:7] connects to MEM_DATA[7:0], and in LE mode, CPU_DATA[0:7] connects to MEM_DATA[63:56]. In each mode, the MSb of the CPU data byte, CPU_DATA[0], connects to the MSb of the memory data byte (either MEM_DATA[63] or MEM_DATA[7], depending on endian mode).

9.4 Byte Swap Instructions

The Power PC architecture defines both word and halfword load/store instructions that have byte swapping capability. Programmers will find these instructions valuable for dealing with the BE nature of this architecture. For example, if a 32-bit configuration register of a typical LE PCI device is read in BE mode, the bytes will appear out of order unless the ”load word with byte swap” instruction is used. The byte swap instructions are:

9

- lhbrx (load half word byte-reverse indexed)
- lwbrx (load word byte-reverse indexed)
- sthbrx (store half word byte-reverse indexed)
- stwbrx (store word byte-reverse indexed)

The byte-reverse instructions should be used in BE mode to access LE devices and in LE mode to access BE devices. Also use these instructions in BE mode to access the BCRs.

9.5 CPU Alignment Exceptions In LE Mode

PowerPC CPUs do not support certain instructions and data alignments in the LE mode. When the CPU encounters an unsupportable situation, it takes an internal alignment exception (machine check) and does not produce an external bus cycle. See the latest CPU documentation for details. Examples include:

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

Table 9-4. 660 Bit Transfer

CPU_DATA[ ]	MEM_DATA[ ]		CPU_DATA[ ]	MEM_DATA[ ]	
	BE Mode	LE Mode		BE Mode	LE Mode
0	7	63	32	39	31
1	6	62	33	38	30
2	5	61	34	37	29
3	4	60	35	36	28
4	3	59	36	35	27
5	2	58	37	34	26
6	1	57	38	33	25
7	0	56	39	32	24
8	15	55	40	47	23
9	14	54	41	46	22
10	13	53	42	45	21
11	12	52	43	44	20
12	11	51	44	43	19
13	10	50	45	42	18
14	9	49	46	41	17
15	8	48	47	40	16
16	23	47	48	55	15
17	22	46	49	54	14
18	21	45	50	53	13
19	20	44	51	52	12
20	19	43	52	51	11
21	18	42	53	50	10
22	17	41	54	49	9
23	16	40	55	48	8
24	31	39	56	63	7
25	30	38	57	62	6
26	29	37	58	61	5
27	28	36	59	60	4
28	27	35	60	59	3
29	26	34	61	58	2
30	25	33	62	57	1
31	24	32	63	56	0

9.6 Endian Mode Examples

Many designers feel that mastering endian mode operations is quite challenging. This section presents examples of endian mode operations.

9.6.1 One Byte Transfers

Figure 9-2 is an example of a 1-byte write of data *a* to address xxxx xxx0**.** The BE case shows data *a* leaving the CPU bus interface on CPU byte lane 0, with a corresponding CPU bus address of xxxx xxx0. The 660 does not swap the data bytes, and does not munge the address, so if system memory is the target of the transaction, then data *a* flows through to the memory data bus on byte lane 0. If the target of the transaction is the PCI bus (memory transaction), then data *a* is driven onto PCI byte lane 0, and PCI_AD[2:0] will be 000.

In the LE case of Figure 9-2, the CPU bus interface shifts data *a* from byte lane 0 to byte lane 7, and munges the address from 0 to 7. The 660 bridge swaps the data byte lanes to put data *a* on memory byte lane 0. The 660 unmunges the address back to 0, and drives xxxx xxx0 onto the memory address bus. If the target of the transaction is the PCI bus (memory transaction), then data *a* is driven onto PCI byte lane 0, and PCI_AD[2:0] will be 000.

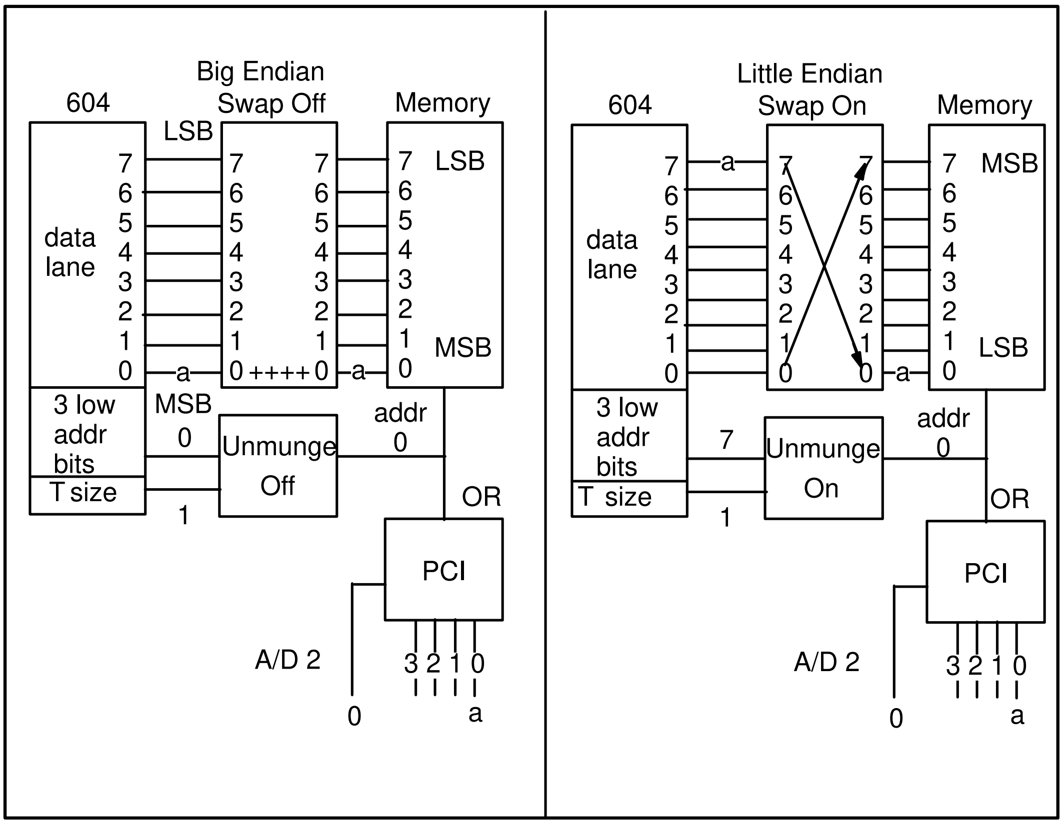


Figure 9-2. One Byte Transfer at Address xxxx xxx0

Figure 9-3 is an example of a 1-byte write of data *a* to address xxxx xxx2**.** The BE case shows data *a* leaving the CPU bus interface on CPU byte lane 2, with a corresponding CPU bus address of xxxx xxx2. The 660 does not swap the data bytes, and does not munge the address, so if system memory is the target of the transaction, then data *a* flows through to the memory data bus on byte lane 2. If the target of the transaction is the PCI bus (memory transaction), then data *a* is driven onto PCI byte lane 2, and PCI_AD[2:0] will be 010.

In the LE case of Figure 9-3, the CPU bus interface has shifted data *a* from byte lane 2 to byte lane 5, and munged the address from 2 to 5. The 660 bridge swaps the data byte lanes to put data *a* on memory byte lane 2. The 660 unmunges the address back to 2, and drives xxxx xxx2 onto the memory address bus. If the target of the transaction is the PCI bus (memory transaction), then data *a* is driven onto PCI byte lane 2, and PCI_AD[2:0] will be 010.

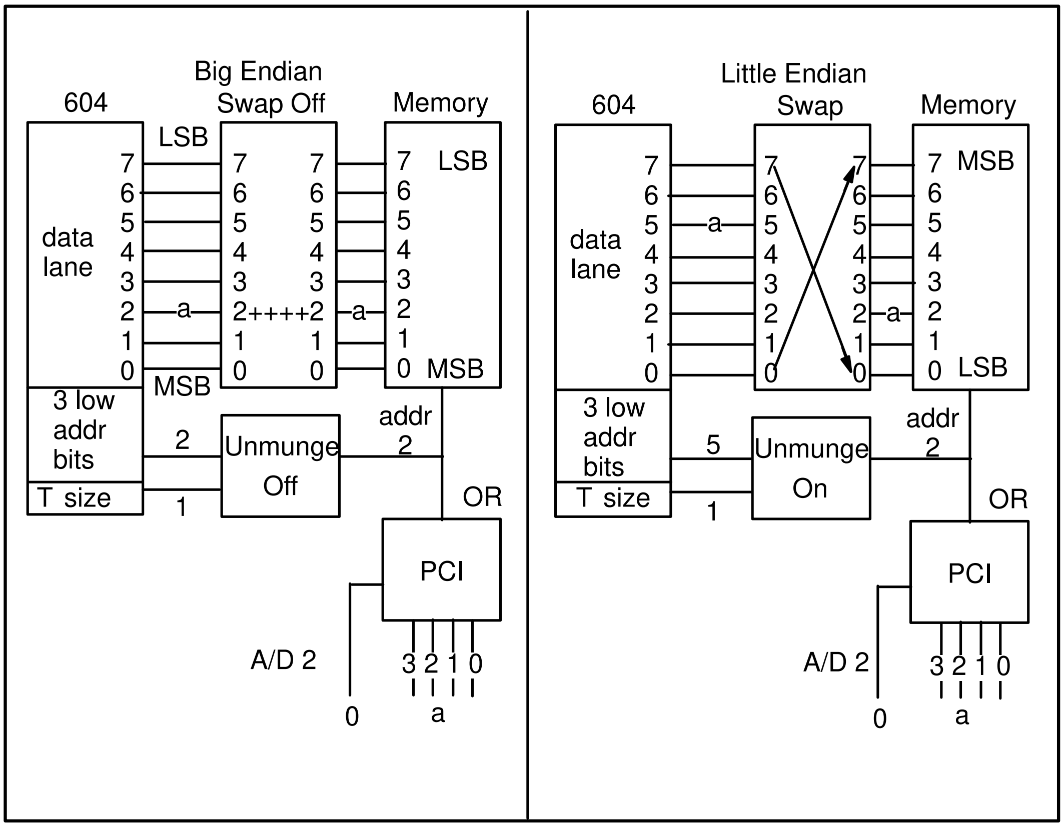


Figure 9-3. One Byte Transfer at Address xxxx xxx2

9.6.2 Two Byte Transfers

Figure 9-4 is an example of a 2-byte write of data *ab* to address xxxx xxx0**.** The BE case shows data *ab* leaving the CPU bus interface on CPU byte lanes 0:1, with a corresponding CPU bus address of xxxx xxx0. The 660 does not swap the data bytes, and does not munge the address, so if system memory is the target of the transaction, then data *ab* flows through to the memory data bus on byte lanes 0:1. If the target of the transaction is the PCI bus (memory transaction), then data *ab* is driven onto PCI byte lanes 0:1, and PCI_AD[2:0] will be 000.

In the LE case of Figure 9-4, the CPU bus interface shifts data *ab* from byte lanes 0:1 to byte lanes 6:7, and munges the address from 0 to 6. The 660 bridge swaps the data byte lanes to put data *ab* on memory byte lanes 1:0. The 660 unmunges the address back to 0, and drives xxxx xxx0 onto the memory address bus. If the target of the transaction is the PCI bus (memory transaction), then data *ab* will be driven onto PCI byte lane 1:0, and PCI_AD[2:0] will be 000.

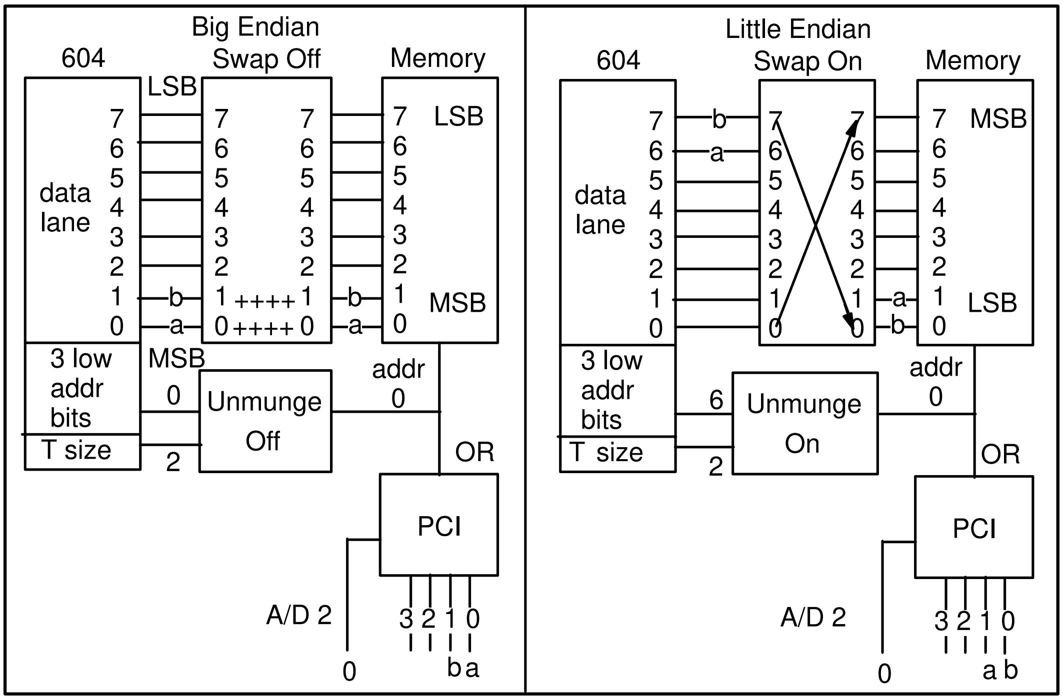


Figure 9-4. Two Byte Transfer at Address xxxx xxx0

9.6.3 Four Byte Transfers

Figure 9-5 is an example of a 4-byte write of data *abcd* to address xxxx xxx4**.** The BE case shows data *abcd* leaving the CPU bus interface on CPU byte lanes 4:7, with a corresponding CPU bus address of xxxx xxx4. The 660 does not swap the data bytes, and does not munge the address, so if system memory is the target of the transaction, then data *abcd* flows through to the memory data bus on byte lanes 4:7. If the target of the transaction is the PCI bus (memory transaction), then data *abcd* is driven onto PCI byte lanes 0:3, and PCI_AD[2:0] will be 100.

In the LE case of Figure 9-5, the CPU bus interface shifts data *abcd* from byte lanes 4:7 to byte lanes 0:3, and munges the address from 4 to 0. The 660 bridge swaps the data byte lanes to put data *abcd* on memory byte lanes 7:4. The 660 unmunges the address back to 4, and drives xxxx xxx4 onto the memory address bus. If the target of the transaction is the PCI bus (memory transaction), then data *abcd* is driven onto PCI byte lane 3:0, and PCI_AD[2:0] will be 100.

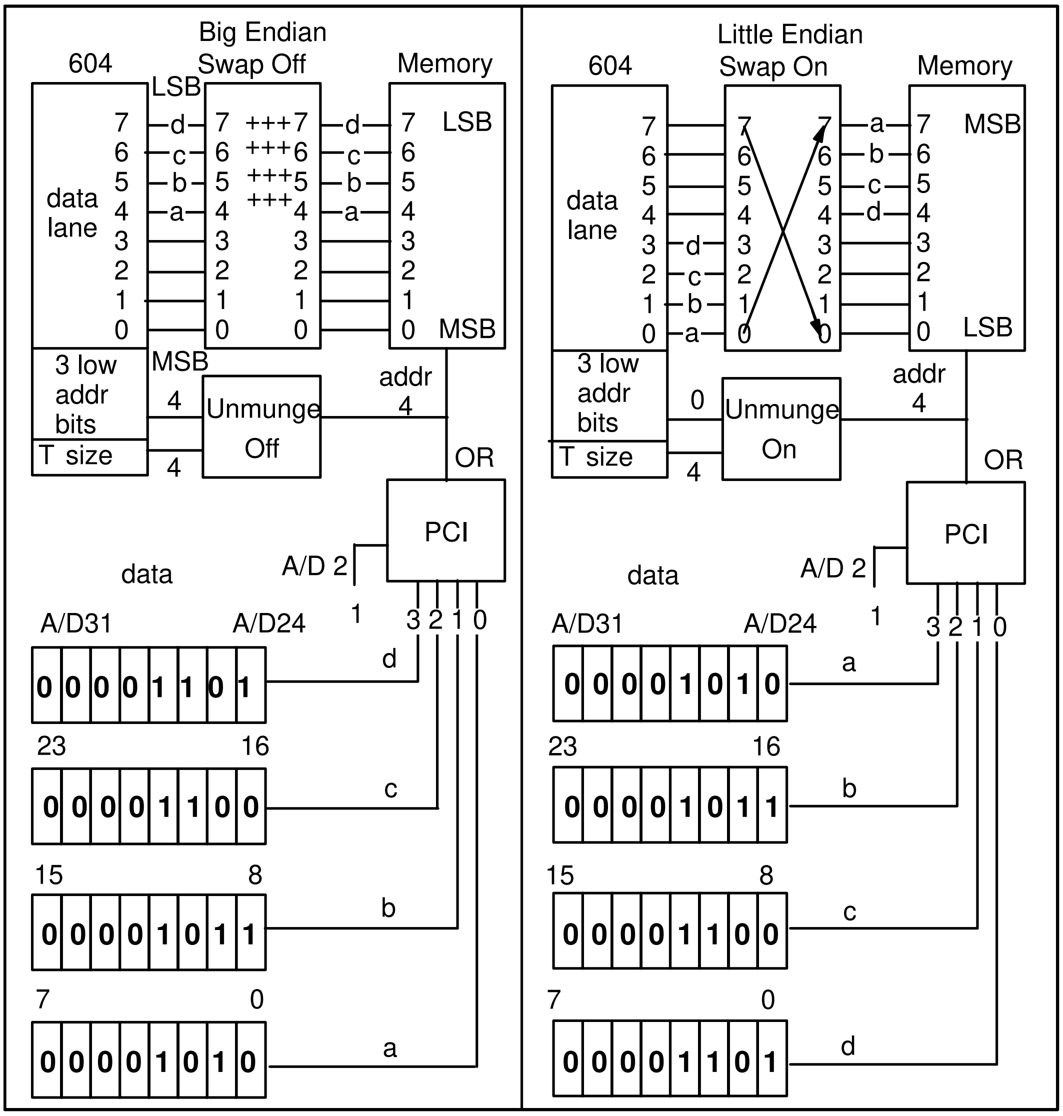


Figure 9-5. Four Byte Transfer at Address xxxx xxx4

9.6.4 Three byte Transfers

There are no explicit Load/Store three-byte instructions; however, three-byte transfers occur as a result of unaligned four-byte loads and stores as well as a result of move multiple and string instructions.

The TSIZ=3 transfers with address pins = 0, 1, 2, 3, 4, or 5 may occur in BE. All of the other TSIZ and address combinations produced by move multiple and string operations are the same as those produced by aligned or unaligned word and half-word loads and stores.

Since move multiples, strings, and unaligned transfers cause machine checks in LE mode, they are not of concern in the BE design.

9.6.5 Eight Byte Transfers

Most instruction fetching is with cache on. In this case, instructions are fetched in eight-byte doublewords. Figure 9-6 shows the instruction alignment. I1=abcd, I2=efgh at address xxxx xxx0

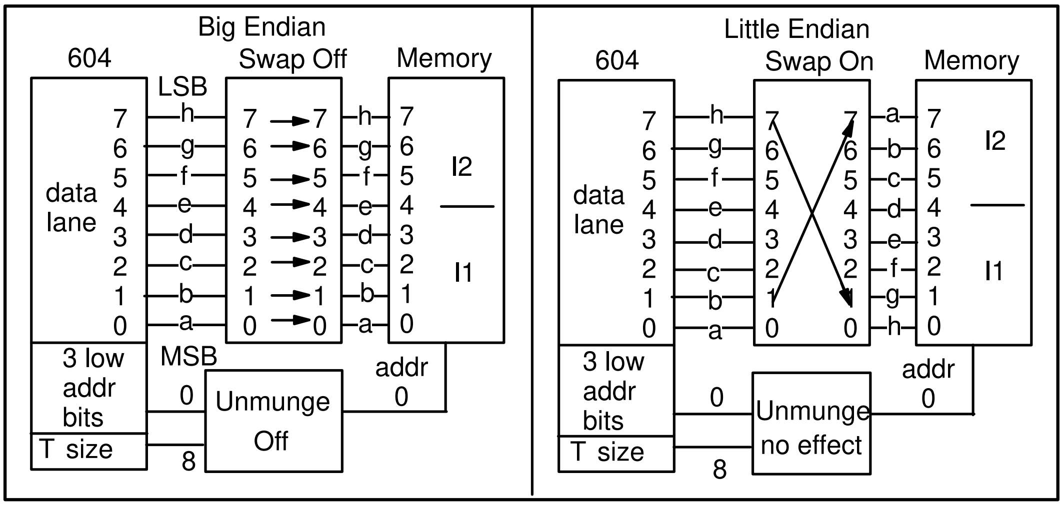


Figure 9-6. Eight Byte Transfer at Address xxxx xxx0

Figure 9-6 is an example of an 8-byte write of data *abcdefgh* to address xxxx xxx0**.** The BE case shows data *abcdefgh* leaving the CPU bus interface on CPU byte lanes 0:7, with a corresponding CPU bus address of xxxx xxx0. The 660 does not swap the data bytes, and does not munge the address, so if system memory is the target of the transaction, then data *abcdefgh* flows through to the memory data bus on byte lanes 0:7.

In the LE case of Figure 9-6, the CPU bus interface shifts the data 0 lanes (no shift), and munges the address from 0 to 0. The 660 bridge swaps the data byte lanes to put data *abcdefgh* on memory byte lanes 7:0. The 660 unmunges the address back to 0, and drives xxxx xxx0 onto the memory address bus.

9.7 Endian Mode Flow Oriented Examples

This section contains examples of endian mode data flows in a different format

9.7.1 1-Byte Example at Address xxxx xxx1

```
1 Byte        Big Endian                    Little Endian
         MSB                               MSB
CPU Bus   0 1 2 3   4 5 6 7 - byte lane -  0 1 2 3   4 5 6 7     A=1 *
  A=1       b               - data      -    b
      off | | | |   | | | | -  swapper  -  \/\/\/\/\/\/\/\/ ON
            b               - data      -              b
memory    0 1 2 3   4 5 6 7 - byte lane -  0 1 2 3   4 5 6 7     A=6 **
  A=1    MSB                                             MSB
              \ \ \ \                               / / / /
                 b                                   b
PCI Bus        0 1 2 3      - 1st data beat -     0 1 2 3
 AD[2:0]=000  MSB                                      MSB  AD[2:0]=100 **
```

 * This address has been munged by the CPU.
** This address has been munged by the CPU and unmunged by the 660.

9.7.2 2-Byte Example at Address xxxx xxx0

```
2 Byte        Big Endian                    Little Endian
         MSB                               MSB
CPU Bus   0 1 2 3   4 5 6 7 - byte lane -  0 1 2 3   4 5 6 7     A=0 *
  A=0     a b               - data      -  a b
      off | | | |   | | | | -  swapper  -  \/\/\/\/\/\/\/\/ ON
          a b               - data      -              b a
memory    0 1 2 3   4 5 6 7 - byte lane -  0 1 2 3   4 5 6 7     A=6 **
  A=0    MSB                                             MSB
              \ \ \ \                               / / / /
               a b                                   b a
PCI Bus        0 1 2 3      - 1st data beat -     0 1 2 3
 AD[2:0]=000  MSB                                      MSB  AD[2:0]=100 **
```

 * This address has been munged by the CPU.
** This address has been munged by the CPU and unmunged by the 660.

9

9.7.3 4-Byte Example at Address xxxx xxx0

```
4 Byte       Big Endian                   Little Endian
         MSB                              MSB
CPU Bus    0 1 2 3  4 5 6 7 - byte lane -  0 1 2 3  4 5 6 7     A=0 *
  A=0      a b c d          - data      -  a b c d
      off | | | |  | | | | -  swapper  -  \/\/\/\/\/\/\/\/ ON
           a b c d          - data      -           d c b a
memory     0 1 2 3  4 5 6 7 - byte lane -  0 1 2 3  4 5 6 7     A=4 **
  A=0    MSB                                             MSB
              \ \ \ \                              / / / /
               a b c d                           d c b a
PCI Bus        0 1 2 3     - 1st data beat -     0 1 2 3
 AD[2:0]=000  MSB                                      MSB  AD[2:0]=100 **
```

* This address has been munged by the CPU.
** This address has been munged by the CPU and unmunged by the 660.

9.7.4 8-Byte Example at Address xxxx xxx0

```
8 Byte         Big Endian                   Little Endian
         MSB                              MSB
CPU Bus    0 1 2 3  4 5 6 7 - byte lane -  0 1 2 3  4 5 6 7
           a b c d  e f g h - data      -  a b c d  e f g h
      off | | | |  | | | | -  swapper  -  \/\/\/\/\/\/\/\/ ON
           a b c d  e f g h - data      -  h g f e  d c b a
memory     0 1 2 3  4 5 6 7 - byte lane -  0 1 2 3  4 5 6 7
         MSB                                             MSB
             \ \ \ \                                | | | |
              a b c d                               d c b a
PCI Bus AD2=0 0 1 2 3 - 1st data beat -    AD2=0    0 1 2 3
             MSB                                         MSB
               / / / /                             \ \ \ \
              e f g h                               h g f e
       AD2=1  0 1 2 3 - 2nd data beat -    AD2=1    0 1 2 3
             MSB                                         MSB
```

9.8 Tabular Endian Mode Examples

The following tables illustrate CPU to memory and CPU to PCI transfers of various sizes in both BE and LE modes. Each table illustrates a subset of the following information:

- The CPU bus address and byte lanes
- The 660 byte lane, memory byte lanes, and active CAS#
- The PCI address and byte lanes

9.8.1 One-Byte CPU to Memory Transfer in BE Mode

CPU A31 30 29	CPU add	BYTE LANE	663 BYTE LANE*	MEM BYTE LANE	CAS# ACTIVE
0 0 0	0	0 MSB	0	0	0
1 0 0	1	1	1	1	1
0 1 0	2	2	2	2	2
1 1 0	3	3	3	3	3
0 0 1	4	4	4	4	4
1 0 1	5	5	5	5	5
0 1 1	6	6	6	6	6
1 1 1	7	7 LSB	7	7	7
NOT MUNGED				SWAP OFF	NOT UNMUNGED

*At the CPU side.

9.8.2 One-Byte CPU to Memory Transfer in LE Mode

CPU A31 30 29	CPU add	CPU BYTE LANE	663 BYTE LANE*	MEM BYTE LANE	CAS# ACTIVE
0 0 0	0	0 MSB	0	7	7
1 0 0	1	1	1	6	6
0 1 0	2	2	2	5	5
1 1 0	3	3	3	4	4
0 0 1	4	4	4	3	3
1 0 1	5	5	5	2	2
0 1 1	6	6	6	1	1
1 1 1	7	7 LSB	7	0	0
MUNGED				SWAP ON	UNMUNGED

*At the CPU side.

9.8.3 One-Byte CPU to PCI Transfer in BE Mode

CPU A31	30	29	CPU add	BYTE LANE	663 BYTE LANE	PCI BYTE LANE	A/D** 2 1 0	BE# 3 2 1 0
							(0=active byte enable)	
0	0	0	0	0 MSB	0	0	0 0 0	1 1 1 0
1	0	0	1	1	1	1	0 0 1	1 1 0 1
0	1	0	2	2	2	2	0 1 0	1 0 1 1
1	1	0	3	3	3	3	0 1 1	0 1 1 1
0	0	1	4	4	4	0	1 0 0	1 1 1 0
1	0	1	5	5	5	1	1 0 1	1 1 0 1
0	1	1	6	6	6	2	1 1 0	1 0 1 1
1	1	1	7	7 LSB	7	3	1 1 1	0 1 1 1
NOT MUNGED						SWAP OFF	NOT UNMUNGED	

**AD[0:1] set to 00 for all PCI transactions except I/O cycles.

9.8.4 One-Byte CPU to PCI Transfer in LE Mode

604 A31	30	29	604 add	BYTE LANE	663* BYTE LANE	PCI BYTE LANE	A/D ** 2 1 0	BE# 3 2 1 0
							(0=active byte enable)	
0	0	0	0	0 MSB	0	3	1 1 1	0 1 1 1
1	0	0	1	1	1	2	1 1 0	1 0 1 1
0	1	0	2	2	2	1	1 0 1	1 1 0 1
1	1	0	3	3	3	0	1 0 0	1 1 1 0
0	0	1	4	4	4	3	0 1 1	0 1 1 1
1	0	1	5	5	5	2	0 1 0	1 0 1 1
0	1	1	6	6	6	1	0 0 1	1 1 0 1
1	1	1	7	7 LSB	7	0	0 0 0	1 1 1 0
MUNGED						SWAP ON	UNMUNGED	

*At the CPU side. **AD[0:1] set to 00 for all PCI transactions except I/O cycles.

9.8.5 Two-Byte CPU to Memory or PCI Transfer

PROG TARG ADDR	BE MODE 604 add	BE MODE BE a29:31	LE MODE (x or w 110) Add	LE MODE a29:31	BE OR LE Target bytes	BE OR LE CAS# 0	BE OR LE CAS# 7	BE OR LE PCI CBE# AD2	BE OR LE PCI CBE# 3210
0	0	000	6	110	0–1	0011	1111	0	1100
1	1	001	7 E	111	1–2	E 1001	1111	0	E 1001
2	2	010	4	100	2–3	1100	1111	0	0011
3	3	011	5 E	101	3–4	E 1110	0111	1	E PPPP
4	4	100	2	010	4–5	1111	0011	1	1100
5	5	101	3 E	011	5–6	E 1111	1001	1	E 1001
6	6	110	0	000	6–7	1111	1100	1	0011
7	N	NNN	1 E	001	NNN	E NNNN	NNNN	N	E NNNN

N= not emitted by 60X because it crosses 8 bytes (transforms to 2 singles in BE, machine CH in LE)
P= not allowed on PCI (crosses 4 bytes)
E= causes exception (does not come out on 604 bus) in LE mode

9.8.6 Rearranged 2-Byte Transfer Information

This table contains the same information as found in section 9.8.5, but is arranged to show the CAS# and PCI byte enables that activate as a function of the address presented at the pins of the CPU and as a function of BE/LE mode.

2 BYTE XFERS 60X ADDRESS PINS		BE CAS#0:7 0	BE CAS#0:7 7	BE PCI A2	BE CBE# 3210	LE CAS#0:7 0	LE CAS#0:7 7	LE PCI AD2	LE CBE# 3210
0	000	0011	1111	0	1100	1111	1100	1	0011
1	001	1001	1111	0	1001	E NNNN	NNNN	N	E NNNN
2	010	1100	1111	0	0011	1111	0011	1	1100
3	011	1110	0111	0	PPPP	E 1111	1001	1	E 1001
4	100	1111	0011	1	1100	1100	1111	0	0011
5	101	1111	1001	1	1001	E 1110	0111E	0	E PPPP
6	110	1111	1100	1	0011	0011	1111	0	1100
7	111	NNNN	NNNN	N	NNNN	E 1001	1111E	0	E 1001

N= not emitted by 60X because it crosses 8 bytes (transforms to 2 singles in BE, machine CH in LE)
P= not allowed on PCI (crosses 4 bytes)
E= causes exception (does not come out on 604 bus) in LE mode

9.8.7 Four-Byte CPU to Memory or PCI Transfer

PROG TARG ADDR	BE MODE 604 BE add	a29:31	LE MODE (x or w 100) add	a29:31	BE OR LE Target bytes	BE OR LE CAS# 0:7 0 7	BE OR LE PCI CBE# AD2	3210
0	0	000	4	100	0-3	0000 1111	0	0000
1	1	001	5	E 101	1-4	E 1000 0111	0	E PPPP
2	2	010	6	E 110	2-5	E 1100 0011	0	E PPPP
3	3	011	7	E 111	3-6	E 1110 0001	1	E PPPP
4	4	100	0	000	4-7	1111 0000	1	0000
5	5	NNN	1	E NNN	N-N	NNNN NNNN	1	E NNNN
6	6	NNN	2	E NNN	N-N	NNNN NNNN	1	E NNNN
7	7	NNN	3	E NNN	N-N	NNNN NNNN	1	E NNNN

N= not emitted by 60X because it crosses 8 bytes (transformed into 2 bus cycles)
P= not allowed on PCI (crosses 4 bytes)
E= causes exception (does not come out on 604 bus) in LE mode

9.8.8 Rearranged 4-Byte Transfer Information

This table contains the same information as found in section 9.8.7, but it is arranged to show the CAS# and PCI byte enables that activate as a function of the address presented at the pins of the CPU and as a function of BE/LE mode.

4 BYTE XFERS 60X ADDRESS PINS		BE CAS#0:7 0 7	BE PCI CBE# A2	3210	LE CAS#0:7 0 7	LE PCI CBE# AD2	3210
0	000	0000 1111	0	0000	1111 0000	0	0000
1	001	1000 0111	0	PPPP	E NNNN NNNN	0	E NNNN
2	010	1100 0011	0	PPPP	E NNNN NNNN	0	E NNNN
3	011	1110 0001	0	PPPP	E NNNN NNNN		E NNNN
4	100	1111 0000	1	0000	0000 1111	1	0000
5	101	NNNN NNNN	1	NNNN	E 1000 0111	1	E PPPP
6	110	NNNN NNNN	1	NNNN	E 1100 0011	1	E PPPP
7	111	NNNN NNNN	1	NNNN	E 1110 0001	1	E PPPP

N= not emitted by 60X because it crosses 8 bytes (transformed into 2 bus cycles)
P= not allowed on PCI (crosses 4 bytes)
E= causes exception (does not come out on 604 bus) in LE mode
X= not supported in memory controller (crosses 4-byte boundary

9.9 Changing BE/LE Mode

There are two BE/LE mode controls. One is inside the CPU and the other is a register bit on the motherboard. The CPU interior mode is not visible to the system hardware. The BE mode bit is a bit in I/O space which is memory mapped just like other I/O registers. It defaults to BE mode.

The 604 CPU always powers up in BE mode and begins fetching code. Thus the first of the ROM code must be BE code. Care must be taken when switching endian mode in order to synchronize the internal and external modes, to flush all caches, and to avoid executing extraneous code.

The following process switches the system from BE to LE mode:

1. Disable L1 caching. Disable L2 caching.
2. Flush all system caches.
3. Turn off interrupts immediately after servicing all outstanding interrupts.
4. Mask all interrupts.
5. Set the CPU state and the LE bit to LE. Note that CPU is now in LE mode. All instructions must be in LE order.
6. Put interrupt handlers and CPU data structures in LE format.
7. Enable caches.
8. Enable Interrupts.
9. Start the LE operating system initialization.

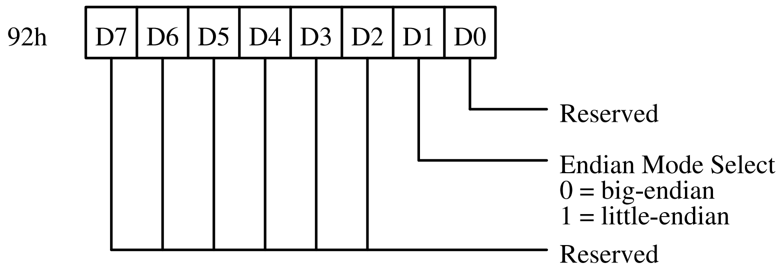
9.9.1 Special Port 92 Mirror BCR

Direct Access 8000 0092	Read/Write	Reset to 00

CPU reads of this register always cause the bridge to arbitrate for the PCI bus and then to execute a single-byte PCI I/O read of location 8000 0092 (the data returned is sourced by a PCI agent). CPU writes to this register are latched into the bridge BCR and also forwarded to the PCI bus as a single-byte PCI I/O write to location 8000 0092. This allows the 660 Bridge to implement the function associated with this register without running wires from an external register into the 664 Controller and also allows systems to support other functions in bits 7:2 and 0 of special port 92. This transaction is usually subtractively decoded by the I/O bus bridge for access to an external logic register.

The port 92 register is a PCI target device external to the Bridge. It either actively or subtractively decodes and responds to a PCI I/O read or write of PCI address 8000_0092 with DEVSEL# and TRDY#, according to correct PCI protocol. Regardless of whether the Bridge is in internal or external register mode, reads and writes of this register are forwarded to the PCI bus. Reads always return the data which is sourced by the PCI device (the contents of the internal BCR are used during the operation of the Bridge). The Bridge always drives the data lines during writes, and the data is also latched into the internal BCR. If the PCI target does not respond to the transaction (during either a read or a write and in both internal and external modes) an error is generated and correct operation does not occur.

CPU writes to this register are not posted or pipelined to ensure that the results take effect before the CPU begins any other bus operations. Other PCI bus masters are able to access the external register, but during writes the data is not latched into the 660 Bridge.



Bit 1 Endian mode select: If the value of this bit is changed, the endian mode is switched immediately following completion of the CPU write to the register.

Section 10
Bridge Control Registers

The Bridge Control Registers (BCRs) allow the designer a high degree of control over system operations. CPU bus masters access BCRs to configure the operation of the 660 Bridge, to generate PCI configuration transactions, and to access various system components. Some BCR accesses are to real registers, either internal or external to the 660 Bridge. Some BCR accesses do not address real registers, per se, but cause the Bridge to initiate various other sequences, such as PCI configuration transactions or ROM writes.

Accesses to the 660 Bridge BCR set allow the CPU to:

- Configure, monitor, and control the 660 bridge,
- Perform PCI type 0 and type 1 configuration transactions,
- Perform PCI I/O transactions to external logic,
- Execute PCI interrupt acknowledge transactions, and
- Write to and lockout the ROM.

All BCR information is presented in little-endian notation. All BCR addresses are given for contiguous mode. When accessing the BCRs in BE mode, use byte-swap instructions.

CPU writes to the register set are never posted or pipelined. This ensures that an operation changed by the write takes effect before any other CPU bus operation is executed. Unless otherwise noted, all changes to the BCRs take effect immediately. Thus a BCR write that changes error checking from parity to ECC mode will take effect before the next memory access.

10.1 Overview

BCRs may be grouped as direct-access BCRs (which are accessed using a single CPU transfer) and indexed BCRs (which are accessed through the PCI/BCR configuration address and PCI/BCR configuration data registers using two CPU transfers).

10.1.1 Direct-Access Bridge Control Registers

A CPU bus master reads or writes a direct-access BCR by initiating a memory read or write to an address listed in Table 10-1. In response, the 660 Bridge requests the PCI bus from the PCI arbiter. Upon being granted the PCI bus (or if the bus is already parked on the CPU), the Bridge initiates one of the following transactions:

- A PCI BCR transaction to access a 650 Bridge compatible BCR
- A PCI BCR transaction to access an indexed BCR
- A PCI I/O transaction to access an external logic register
- A type 0 PCI configuration transaction to a PCI agent
- A PCI interrupt acknowledge transaction
- A direct-connect flash ROM write or ROM write lockout transaction.

The 650 compatible direct access BCRs are provided to ease software migration from the IBM27–82650 Bridge chipset to the 660 Bridge chipset. These BCRs are especially useful during system and software development.

The 660 Bridge is a much more programmable device than the 650 Bridge, but using the 650 compatible direct access BCRs tends to limit the programmability of the 660 Bridge to that of the 650 Bridge. Thus whenever possible, the designer should use the 660 Bridge (indexed) BCR set rather than the 650 compatible direct access BCR set.

10.1.2 Indexed Bridge Control Register Access

The indexed BCRs shown in Table 10-2 are accessed by means of the PCI/BCR configuration address BCR (address 8000 0CF8h, section 10.3.1.1) and the PCI/BCR configuration data BCR (address 8000 0CFCh, section 10.3.1.2). This method of access is described by the *PCI Local Bus Specification (Rev. 2.0),* and is therein defined as *Configuration Mechanism #1*. Note that these two registers are direct access BCRs.

The sequence of events that the Bridge generates to access the BCRs is called a PCI BCR transaction. This transaction is very similar to a PCI configuration transaction.

10.1.3 Indexed Bridge Control Registers

The 660 Bridge features a large indexed BCR set, which allows extensive programmability of the Bridge and the system. Indexed BCR operations allow CPU bus masters to perform the following operations:

- Access the 660 Bridge internal indexed BCRs
- Initiate PCI type 0 configuration transactions
- Initiate PCI type 1 configuration transactions

The target registers of PCI agents that claim the PCI configuration transactions appear to the CPU to be virtual indexed BCRs.

10.2 Direct-Access BCRs

A CPU bus master reads or writes a direct-access BCR by initiating a memory read or write to an address listed in Table 10-1.

Table 10-1. Direct-Access BCRs With Section References

Bridge Control Register	CPU Bus Address	R/W	Bytes	Resultant Bus Transaction (See Section)	See Section
Special Port 92 Mirror	8000 0092	R/W	1	PCI I/O to 0000 0092	10.2.2.1
L2 Invalidate	8000 0814	W	1	PCI BCR (10.2.1) or PCI I/O to 0000 0814	10.2.2.2
System Control 81C	8000 081C	R/W	1	PCI BCR (10.2.1) or PCI I/O to 0000 081C	10.2.2.3
Memory Controller Misc	8000 0821	R/W	1	PCI BCR (10.2.1)	10.2.2.4
Memory Parity Error Status	8000 0840	R	1	PCI BCR (10.2.1)	10.2.2.5
L2 Error Status	8000 0842	R	1	PCI BCR (10.2.1)	10.2.2.6
L2 Parity Error Read and Clear	8000 0843	R	1	PCI BCR (10.2.1)	10.2.2.7
Unsupported Transfer Type Error	8000 0844	R	1	PCI BCR (10.2.1)	10.2.2.8
I/O Map Type	8000 0850	R/W	1	PCI BCR (10.2.1) or PCI I/O to 0000 0850	10.2.2.9
PCI/BCR Configuration Address	8000 0CF8	R/W	4	PCI BCR (10.2.1)	10.3.1.1
PCI/BCR Configuration Data	8000 0CFC	R/W	4	PCI BCR (10.2.1) or PCI Configuration	10.3.1.2
PCI Type 0 Configuration Addresses IBM27–82650 Compatible	8080 08xx 8080 10xx 8080 20xx 8080 40xx 8080 80xx 8081 00xx 8082 00xx 8084 00xx 8088 00xx 8090 00xx 80A0 00xx 80C0 00xx	R/W	4	PCI Configuration	3.4.5.1
System Error Address	BFFF EFF0	R	4	PCI BCR (10.2.1)	10.2.2.13
Interrupt Acknowledge	BFFF FFF0	R	1	PCI Interrupt Acknowledge	3.4.6
ROM Write	FFFF FFF0	W	4	PCI BCR (ROM)	7.1.2
ROM Lockout	FFFF FFF1	W	1	PCI BCR (ROM)	7.1.2.2

In response, the 660 Bridge requests the PCI bus from the PCI arbiter. Upon being granted the PCI bus (or if the bus is already parked on the CPU), the bridge initiates one of the following transactions:

- A PCI BCR transaction to access a 650 Bridge compatible BCR
- A PCI BCR transaction to access an indexed BCR
- A PCI I/O transaction to access an external logic register

- A type 0 PCI configuration transaction to a PCI agent
- A PCI interrupt acknowledge transaction
- A direct-connect flash ROM write or ROM write lockout transaction

10.2.1 PCI BCR Transactions

A PCI BCR transaction is the result of a CPU transfer to a CPU address location shown in Table 10-1. Figure 10-1 shows the information flows in the system and Figure A-17 shows the basic read timing. During writes, the data flows from the CPU to the 663 over the CPU data lines and from the buffer to the controller over the PCI_AD lines. During reads, the data flows from the 664 to the buffer over the PCI_AD lines and from the buffer to the CPU over the CPU data bus. This allows data to be passed on the PCI_AD lines from the 663 chip, which attaches to the CPU data bus, to the 664 chip, which contains the physical registers.

Only the CPU is capable of causing PCI BCR transactions. Attempts by PCI bus masters to access the BCRs are not claimed by the bridge.

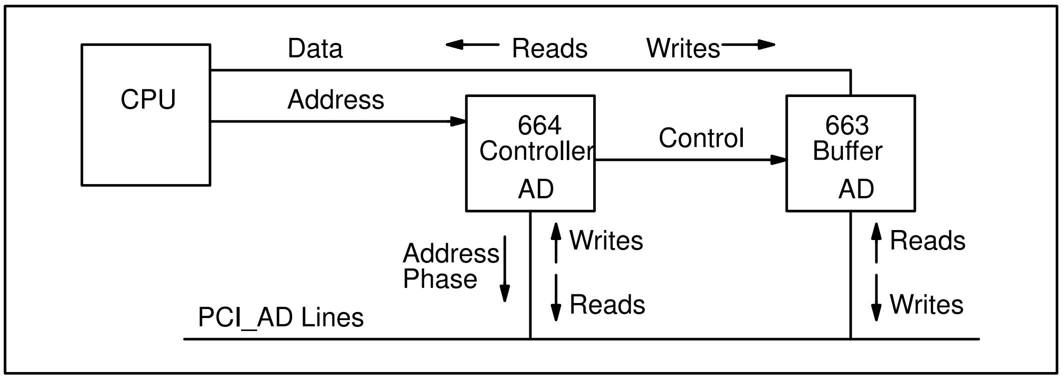


Figure 10-1. BCR Configuration Information Flow

The CPU initiates the PCI BCR transaction by beginning a memory transfer with TS#. The controller decodes the transfer as a BCR access. On the next rising edge of PCI_CLK, the controller asserts PCI_REQ#. Some number of PCI_CLK cycles later, the PCI arbiter grants the PCI bus to the CPU by asserting PCI_GNT#. On the next PCI_CLK (clock 1), the bridge asserts PCI_FRAME# to start the PCI transaction.

During the address phase, the controller drives Bh onto the PCI_C/BE# lines (during reads and writes) to signal a configuration transaction, and drives all zeros onto the PCI_AD lines for the address phase. Since the address is 0000 0000h, none of the IDSEL#s are activated.

During the data phase, the controller drives 0000b onto the PCI_C/BE# lines, and asserts PCI_IRDY#.

During read transactions, the controller drives the data from the selected register onto the PCI_AD lines during the data phase. The data travels over the PCI_AD lines to the buffer and then over the CPU data lines to the CPU.

During write transactions, the controller disables its drivers, and after a turn-around cycle, the buffer drives the data from the CPU bus onto the PCI_AD lines. The controller then latches the required data into the BCR.

At the end of the data phase, the controller signals TA# and AACK# to the CPU to terminate the CPU bus transfer. The controller deasserts PCI_IRDY# and PCI_FRAME#, ending the PCI transaction and allowing another PCI bus master to begin a transaction. Note that PCI_DEVSEL#, PCI_STOP#, IRDY#, and all IDSEL#s remain high during the entire transaction. The 660 Bridge does not drive them high. It tristates them, and they stay high because nothing drives them low. Other PCI agents monitoring the bus see a configuration transaction which is master aborted.

The 660 Bridge deasserts PCI_REQ# on PCI_CLK two. The arbiter can deassert PCI_GNT# to the bridge on or after PCI_CLK one, because PCI_FRAME# remains asserted throughout the transaction, preventing a bus idle state.

10

10.2.2 Direct-Access BCR Listing

This section contains detailed listings of the direct-access BCR set. All addresses provided are in the contiguous address mode. The non-contiguous address is different. All registers are described in little-endian fashion with the highest numbered bit (usually D7) as the most-significant bit and D0 as the least-significant bit. In big-endian mode, the bit ordering is the same, with the left-most bit still as the most-significant bit, but typically referred to as D0. Byte addresses are specified so that (for a single byte transaction) the address is the same regardless of the endian mode.

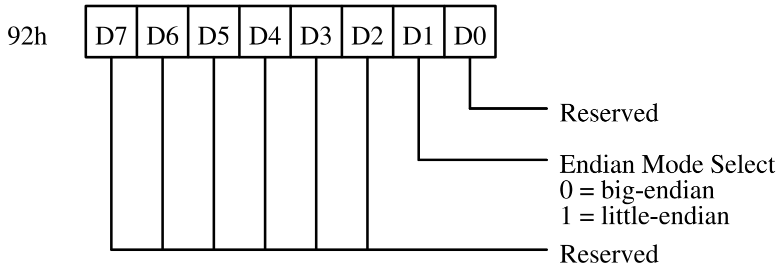
10.2.2.1 Special Port 92 Mirror BCR

Direct Access 8000 0092	Read/Write	Reset to 00

CPU reads of this register always cause the bridge to arbitrate for the PCI bus and then to execute a single-byte PCI I/O read of location 8000 0092 (the data returned is sourced by a PCI agent). CPU writes to this register are latched into the bridge BCR and also forwarded to the PCI bus as a single-byte PCI I/O write to location 8000 0092. This allows the 660 Bridge to implement the function associated with this register without running wires from an external register into the 664 Controller and also allows systems to support other functions in bits 7:2 and 0 of special port 92. This transaction is usually subtractively decoded by the I/O bus bridge for access to an external logic register.

The port 92 register is a PCI target device external to the Bridge. It either actively or subtractively decodes and responds to a PCI I/O read or write of PCI address 8000_0092 with DEVSEL# and TRDY#, according to correct PCI protocol. Regardless of whether the Bridge is in internal or external register mode, reads and writes of this register are forwarded to the PCI bus. Reads always return the data which is sourced by the PCI device (the contents of the internal BCR are used during the operation of the Bridge). The Bridge always drives the data lines during writes, and the data is also latched into the internal BCR. If the PCI target does not respond to the transaction (during either a read or a write and in both internal and external modes) an error is generated and correct operation does not occur.

CPU writes to this register are not posted or pipelined to ensure that the results take effect before the CPU begins any other bus operations. Other PCI bus masters are able to access the external register, but during writes the data is not latched into the 660 Bridge.



Bit 1 Endian mode select: If the value of this bit is changed, the endian mode is switched immediately following completion of the CPU write to the register.

10.2.2.2 L2 Invalidate BCR

Direct Access 8000 0814	Write Only	Reset: Undefined

A write to this 8-bit, write-only register causes all contents of the internal L2 cache to be invalidated (by pulsing TAG_CLR# active for several CPU clocks). The internal L2 cache does not have to be disabled during this operation. Reads to this register are undefined and do not cause an L2 invalidate.

This register can be put into external register support mode so that writes to this register are latched into this register and are also forwarded to the PCI. In this mode, reads to this register are always forwarded to the PCI, which allows functions to be added to the reserved bit locations of this register.

0814h	D7	D6	D5	D4	D3	D2	D1	D0

Any Value

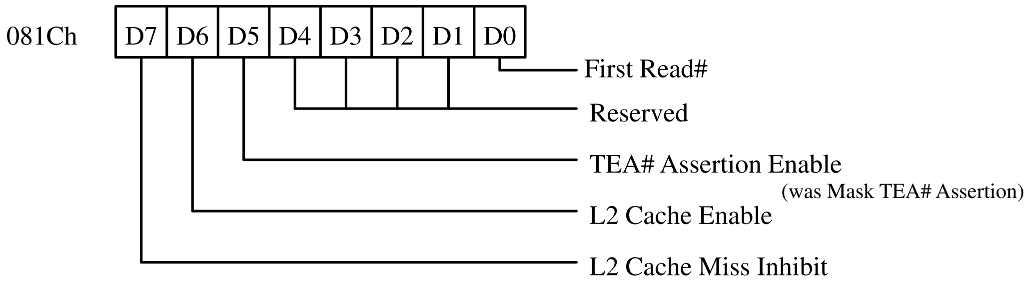
Bits 7:0 Writing any value to this register causes the L2 invalidate operation.

10.2.2.3 System Control 81C BCR

Direct Access 8000 081C	Read/Write	Reset to 00

This BCR controls the L2 cache, the TEA# mask, and other functions.

This 8-bit, read/write BCR can be put into external register mode so that CPU writes to this BCR are latched into the bridge BCR and are also forwarded to the PCI bus as a PCI I/O write to location 0000 081C (the bridge arbitrates for the PCI bus and then executes a single byte, single data phase PCI I/O write). In this mode, reads to this BCR are always forwarded to the PCI as I/O reads to location 0000 081C, which allows the system to use these bits externally (the external logic must provide the proper read value of all bits including bits 7:5).



Bit 0 First_Read#. This bit is 0 the first time this BCR is read after reset. Subsequent reads of this BCR return 1 for this bit. This bit cannot be written by software. Writes to this BCR do not affect the state of this bit (including writes before the initial read).

Bits 4:1 Reserved. May be used externally in external register support mode.

Bit 5 TEA# assertion enable. When enabled, the 664 will assert TEA# when an error condition is detected. Some error conditions can cause a bus hang (for example, XATS# assertion) so the 664 still terminates such cycles with TEA#, even if this bit is set to 0. This bit can also be accessed via the bridge chipset options 1 BCR (index BAh, see section 10.3.35). This bit may be used externally in external register support mode.

0 : TEA# assertion disabled.

1 : TEA# assertion enabled.

Bit 6 L2 Cache Enable: When disabled, the L2 cache does not respond to any cycles (and does not maintain coherence or update the tags). Disabling the L2 cache does not cause its contents to be invalidated. This bit only enables the internal L2 if the internal L2 controller is enabled by bit 1 of the cache status register (index B1h, section 10.3.31). This bit may be used in external register mode to enable/disable an external L2.
0 = L2 cache disabled or not present.
1 = L2 cache enabled.

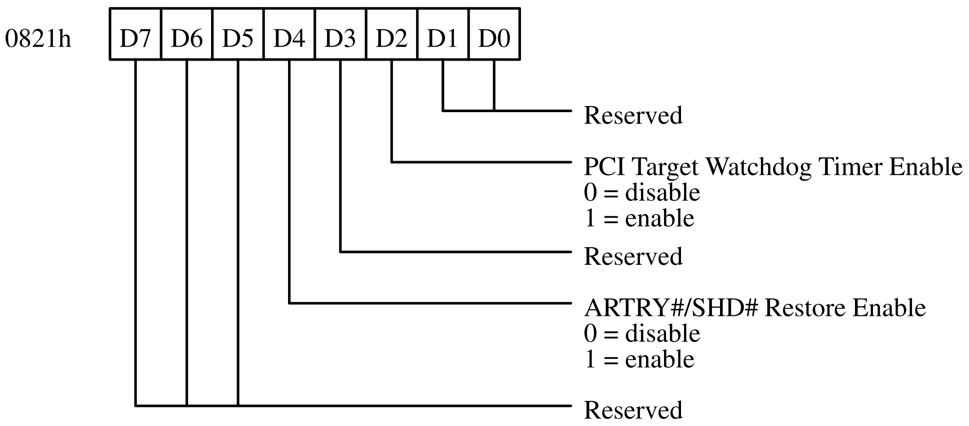
Bit 7 L2 Cache Miss enable: This prevents L2 cache misses from updating the L2 cache. This allows the L2 cache to retain its contents during memory accesses and remain coherent (snooping and tag updates continue to occur).
0 = Prevent updates
1 = Normal operation (allow updates)

10

10.2.2.4 Memory Controller Miscellaneous BCR

Direct Access 8000 0821	Read/Write	Reset to 14h

This 8-bit, read/write register controls miscellaneous functions. The control of memory controller timings from this register is not supported. Instead, this is done by means of the 80h to A7h registers in the 660 Bridge primary register space.



Bit 2 PCI Target Watchdog Timer Enable: This bit controls the enable of the PCI target watchdog timer. When enabled, this timer is cleared and begins counting when a PCI target activates PCI_DEVSEL#. The timer stops when the PCI cycle terminates. If the PCI cycle has not terminated when the timer count reaches 2000 PCI clocks, the cycle is master aborted.

Note: The PCI specification places no limitation on the response time of PCI targets; therefore, this timer needs to be disabled if slow PCI devices are used. Disabling this timer allows the system to lock up if a PCI target hangs after asserting PCI_DEVSEL#.

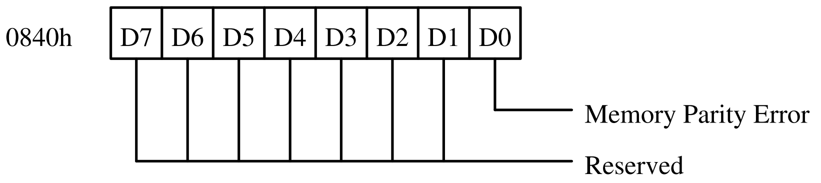
Bit 4 ARTRY#/SHD# Restore Enable: This bit controls the enable of the ARTRY# and SHD# restore logic in the 664. When enabled, the 664 restores ARTRY# and SHD# regardless of what device drove the signal active. This mode can only be used if ARTRY# and SHD# restore is disabled on all other CPU bus devices (for 601, if HID0[29] = 1, and for 603/604, if HID0[7] = 1).

When bit 4 is disabled, the 664 only restores ARTRY# and SHD# if it drove it active.

10.2.2.5 Memory Parity Error Status BCR

Direct Access 8000 0840	Read Only	Reset to 01h

This 8-bit, read-only register indicates the status of memory parity or multi-bit ECC errors. This bit is deactivated by reading the system error address register (port BFFF EFF0h). Bit 0 may also be accessed via the error status 1 register (index C1h, section 10.3.38). Note that L2 parity errors also cause the CPU bus data parity error bit to be set.

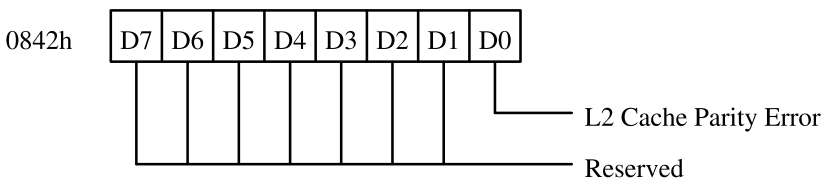


Bit 0 Memory Parity Error:
0 = Error Detected
1 = No Error Detected.

10.2.2.6 L2 Error Status BCR

Direct Access 8000 0842	Read Only	Reset to 01h

This 8-bit, read-only register indicates if a parity error has been detected during a CPU read from the L2 cache. This bit is deactivated by reading the L2 cache parity error read and clear register (port 8000 0843h). This bit may also be accessed via the error status 2 register (index C5h, section 10.3.41). Note that L2 parity errors also cause the CPU bus data parity error bit to be set.

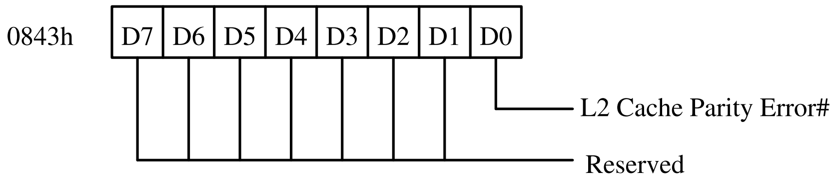


Bit 0 L2 Cache Parity Error:
0 = Error Detected
1 = No Error Detected.

10.2.2.7 L2 Parity Error Read and Clear BCR

Direct Access 8000 0843	Read Only	Reset to 01h

This 8-bit, read-only register indicates if a parity error has been detect during a CPU read from the L2 cache and clears the error if it is active. This bit may also be accessed via the error status 2 register (index C5h, section 10.3.41), but that access will not automatically clear the bit. Note that L2 parity errors also cause the CPU bus data parity error bit to be set.

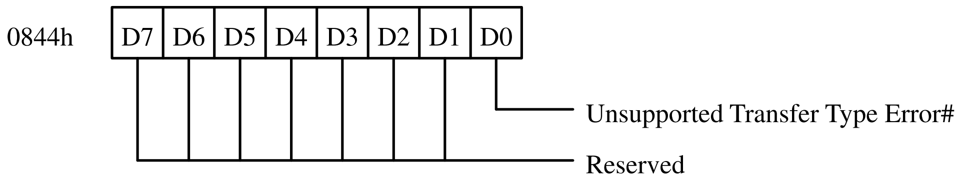


Bit 0 L2 Cache Parity Error:
0 = Error Detected
1 = No Error Detected.

10.2.2.8 Unsupported Transfer Type Error BCR

Direct Access 8000 0844	Read Only	Reset to 01h

This is an 8-bit, read-only register that indicates the status of unsupported transfer type errors or XATS# asserted from the CPU. This bit is cleared (to 1) by reading the system error address register (port BFFF EFF0h). Transfer type error status is also available via the error status 2 register (index C1h, section 10.3.38).



Bit 0 Unsupported Transfer Type Error:
0 = Error Detected
1 = No Error Detected.

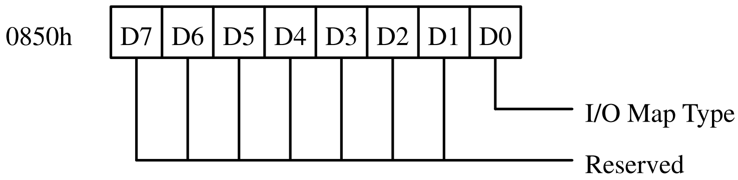
10.2.2.9 I/O Map Type BCR

Direct Access 8000 0850	Read/Write	Reset to 01h

This 8-bit, read/write register determines if the I/O mapping is contiguous or non–contiguous.

This register can be put into external register support mode so that CPU writes to this register are latched into the bridge BCR and are also forwarded to the PCI bus as a PCI I/O write to location 0000 0850h (the bridge arbitrates for the PCI bus and then executes a single byte, single data phase PCI I/O write). In this mode, reads to this register are always forwarded to the PCI (in the same manner), as I/O reads to location 0000 0850h, which allows system designers to externally add function to the reserved bit locations of this register. However, if this mode is used, the external logic must provide the proper read value of all bits including bit 0.

CPU writes to this register are not posted or pipelined to ensure that the results take effect before the CPU begins any other bus operations. This transaction is typically subtractively decoded by the I/O bus bridge for access to an external logic register. Other PCI bus masters are able to access the external registers, but during writes the data will not be latched into the 660 Bridge.



Bit 0 I/O Map Type: This bit can also be written by means of the bridge chip set options 1 register (indexed BAh, section 10.3.35)
0 = Non–Contiguous
1 = Contiguous.

10.2.2.10 PCI/BCR Configuration Address BCR

Direct Access 8000 0CF8	Read/Write	Reset to 0000 0000

This BCR is one of a pair of BCRs that are used to access the indexed BCRs and PCI configuration space, as discussed in section 10.3.1. This BCR is described in section 10.3.1.1.

10.2.2.11 PCI/BCR Configuration Data BCR

Direct Access 8000 0CFC to CFF	Read/Write	Reset: Undefined

This BCR is one of a pair of BCRs that are used to access the indexed BCRs and PCI configuration space, as discussed in section 10.3.1. This BCR is described in section 10.3.1.2.

10.2.2.12 PCI Type 0 Configuration Addresses

Direct Access 8080 08xx to 808C 00xx	Read/Write	Reset: Undefined

Accesses to these 650 Bridge compatible addresses cause the Bridge to run type 0 PCI configuration transactions. PCI configuration accesses can be 1-, 2-, or 4-byte only. Eight byte PCI configuration register accesses are not supported.

10.2.2.13 System Error Address BCR

Direct Access BFFF EFF0	Read Only	Reset: Undefined

This 32-bit, read-only register indicates the address at which a parity or multi-bit ECC error or an illegal transfer error occurred. This register must be accessed by means of a 4-byte transfer.

Reading this register deasserts the memory parity error indicator (which can be read by means of the memory parity error status register port 8000 0840h and primary register C1h) and the illegal transfer error indicator (which can be read by means of the illegal transfer error register port 8000 0844 and primary register C1h).

10.2.2.14 Interrupt Acknowledge BCR

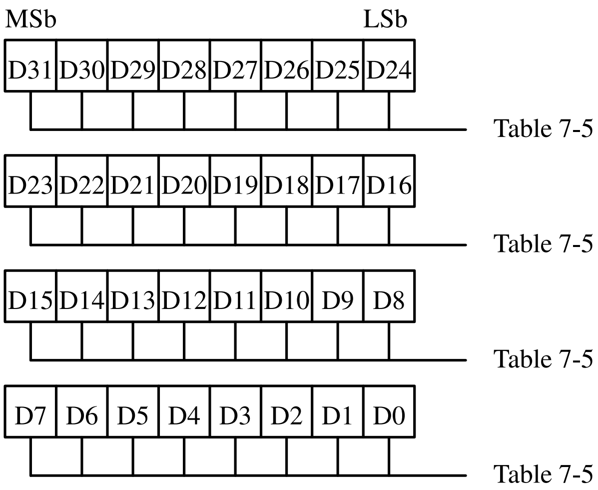
Direct Access BFFF FFF0	Read Only	Reset: Undefined

CPU accesses mapped to this virtual BCR cause the 660 Bridge to execute a PCI interrupt acknowledge transaction. The interrupt vector, which is supplied by the responding PCI agent, is forwarded to the CPU bus as the data read from the BCR. See section 3.4.6.

10.2.2.15 ROM Write Bridge Control BCR

Direct Access FFFF FFF0h	Write Only	Reset NA

This 32-bit, write-only register is used to program the ROM in direct-attach ROM systems (see section 7.1.2). This register must be written by means of a 4-byte transfer. Bits are shown with little-endian labels.

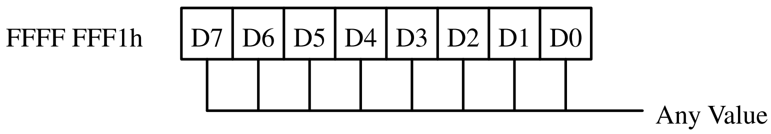


BCR Byte	Content in Little-Endian System	Content in Big-Endian System
MSB	ROM Data	ROM Address low byte
	ROM Address high byte	ROM Address mid byte
	ROM Address mid byte	ROM Address high byte
LSB	ROM Address low byte	ROM Data

10.2.2.16 ROM Lockout BCR

Direct Access FFFF FFF1h	Write Only	Reset NA

After it has been written once, this 8-bit, write-only register prevents ROM writes. This register is only used in direct-attach ROM systems.



Bits 7:0 Writing any value to the register prevents all future writes to a ROM that is connected directly to the 660 through the PCI_AD lines.

10

10.3 Indexed BCRs

The 660 Bridge features a large indexed BCR set, which allows extensive programmability of the Bridge and the system. Indexed BCR operations allow CPU bus masters to perform the following operations:

- Access the 660 Bridge internal indexed BCRs
- Initiate PCI type 0 configuration transactions
- Initiate PCI type 1 configuration transactions

The target registers of PCI agents that claim the PCI configuration transactions appear to the CPU to be virtual indexed BCRs.

10.3.1 Indexed BCR Access

The indexed BCRs shown in Table 10-2 are accessed by means of pairs of direct-access BCR accesses. The method of accessing them follows that described by the *PCI Local Bus Specification (Revision 2.0),* defined as Configuration Mechanism #1.

The first access of the pair, a direct-access BCR access (see Section 10.2.1) to the PCI/BCR configuration address BCR (8000 0CF8h), defines the destination of the transaction.

The second access of the pair, to the PCI/BCR configuration data BCR (8000 0CFCh), reads or writes the data and initiates the bus sequence. There is no physical PCI/BCR configuration data BCR. When the CPU accesses this BCR address, the bridge arbitrates for the PCI bus. After being granted the bus (or if the bus is parked on the CPU) the bridge executes either a PCI BCR transaction to an indexed BCR or a PCI configuration transaction to a PCI agent, depending on the contents of the bus and device fields of the PCI/BCR configuration address BCR.

Data written to the data BCR is forwarded to the actual target device or BCR over the PCI_AD lines during the data phase of the transaction. During reads, data supplied by the target device or BCR as sampled from the PCI_AD lines during the data phase of the transaction is returned to the CPU.

The 660 Bridge indexed BCRs should only be accessed individually during a PCI BCR transaction. For example, the 660 Bridge PCI command BCR (index 04h to 05h) can be accessed as a 2-byte object at index 04h, but the 660 Bridge revision ID BCR (1-byte) and the 660 Bridge PCI standard programming interface (1-byte) BCR should not be read together as a 2-byte object.

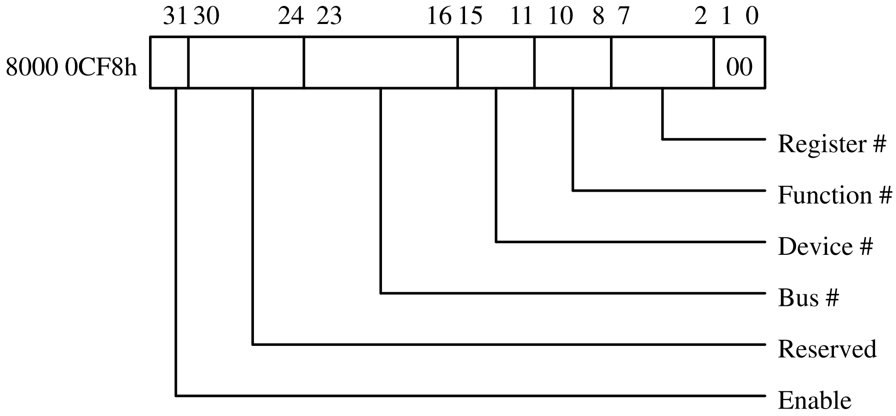
PCI configuration transactions can also be generated using these two BCRs and the appropriate configuration address information. Note that the 660 Bridge places no transfer size or alignment restrictions on PCI configuration transactions, other that they do not cross a 4-byte boundary.

It is often useful to use byte-swap instructions when accessing the indexed BCRs in BE mode.

10.3.1.1 PCI/BCR Configuration Address BCR

Direct Access 8000 0CF8	Read/Write	Reset to 0000 0000

This 32-bit read/write register is used as a pointer to PCI configuration registers during PCI configuration transactions and as a pointer to the selected indexed BCR during indexed BCR configuration transactions. The CPU must perform 4-byte transfers when accessing this BCR. Transfers of less than 4 bytes are mapped to PCI I/O space instead of this BCR.



Bits 1:0 Reserved, always 00b. The 664 drives 00b onto PCI_AD[1:0] during type 0 PCI configuration transactions, and drives 01b onto PCI_AD[1:0] during type 1 PCI configuration transactions.

Bits 7:2 Register Number: During PCI configuration transactions, these bits form the upper six bits (doubleword address) of the PCI register number, which specifies which of the 256 possible bytes within the PCI configuration space of a selected device is addressed. The least significant two bits of the BCR number are determined by the two least significant bits of the address of the accessed PCI/BCR configuration data BCR:

PCI/BCR Configuration Data BCR		Bits 3 and 2 of PCI/BCR Configuration Address BCR							
BCR Address	Bits 1:0 of Address	00		01		10		11	
8000 0CFC	00	0000	B0	0100	B4	1000	B8	1100	BC
8000 0CFD	01	0001	B1	0101	B5	1001	B9	1101	BD
8000 0CFE	10	0010	B2	0110	B6	1010	BA	1110	BE
8000 0CFF	11	0011	B3	0111	B7	1011	BB	1111	BF

For example, if bits [7:2] of the configuration address BCR are 1011 00b, and the configuration data BCR is 8000 0CFEh (...10b), then indexed register B2 (1011 0010) is accessed. During indexed BCR accesses, the BCR index is also determined in this manner.

Bits 10:8 Function Number: During PCI configuration transactions, these bits specify which of eight functions within the PCI configuration space of a selected device is to be addressed. These bits are not used (don't cares) during indexed BCR accesses.

Bits 15:11 Device Number: During PCI configuration transactions, these bits specify which of 21 devices (or slots) is selected. As shown in the table, only one of the PCI_AD[31:11] bits is asserted for each valid value of bits 15:11. These bits are not used during indexed BCR accesses.

Bits 15:11	Asserted PCI_AD[]	Bits 15:11	Asserted PCI_AD[]	Bits 15:11	Asserted PCI_AD[]	Bits 15:11	Asserted PCI_AD[]
00000	none	00110	[16]	01100	[22]	10010	[28]
00001	[11]	00111	[17]	01101	[23]	10011	[29]
00010	[12]	01000	[18]	01110	[24]	10100	[30]
00011	[13]	01001	[19]	01111	[25]	10101	[31]
00100	[14]	01010	[20]	10000	[26]	other	none
00101	[15]	01011	[21]	10001	[27]		

Bits 23:16 Bus Number: During PCI configuration transactions, these bits specify which of 256 buses is selected. The PCI bus attached to the 660 Bridge is bus number 0.

If bus = 0 and device = 0, the bridge executes a BCR access to one of the indexed BCRs when the PCI/BCR configuration data BCR is accessed.

If bus = 0 and device > 0, the bridge masters a standard PCI type 0 configuration transaction when the PCI/BCR configuration data BCR is accessed.

If bus > 0, the bridge masters a standard PCI type 1 configuration transaction when the PCI/BCR configuration data BCR is accessed.

Bit 31 Enable: This bit must be a 1 to enable accesses to either the the PCI configuration registers or the indexed BCRs.

10.3.1.2 PCI/BCR Configuration Data BCR

Direct Access 8000 0CFC	Read/Write	Reset: Undefined

	31	0
8000 0CFCh		

This 32-bit read/write virtual register is used to access the register pointed to by the PCI/BCR configuration address BCR. When the CPU accesses this BCR, the bridge arbitrates for the PCI bus. After being granted the bus (or if the bus is parked on the CPU) the bridge executes either a BCR access or a PCI configuration transaction, depending on the contents of the bus and device fields of the PCI/BCR Configuration Address BCR.

During write transactions, the data written to this BCR is driven onto the PCI_AD lines during the data phase. During read transactions, the data that the CPU receives from this BCR is the data sampled during the data phase of the bus transaction. There is no physical PCI/BCR configuration data BCR. This BCR is accessed as a 1-, 2-, or 4-byte BCR. The least significant 2 bits of the address of the Configuration Data BCR accessed become the 2 LSBs of the accessed PCI or BCR configuration register.

10.3.2 Indexed BCR Summary

Table 10-2 contains a summary listing of the indexed BCRs. Accesses to these registers are described in sections 10.2.1 and 10.3.1. There are BCRs in the 660 other than the ones listed. These BCRs are not supported for customer use. Do not access indexed BCRs other than the ones listed in Table 10-2.

Table 10-2. Indexed BCR Listing

Bridge Control Register	Index	R/W	Bytes	See
PCI Vendor ID	Index 00 – 01	R	2	10.3.3
PCI Device ID	Index 02 – 03	R	2	10.3.4
PCI Command	Index 04 – 05	R/W	2	10.3.5
PCI Device Status	Index 06 – 07	R/W	2	10.3.6
Revision ID	Index 08	R	1	10.3.7
PCI Standard Programming Interface	Index 09	R	1	10.3.8
PCI Subclass Code	Index 0A	R	1	10.3.9
PCI Class Code	Index 0B	R	1	10.3.10
PCI Cache Line Size	Index 0C	R	1	10.3.11
PCI Latency Timer	Index 0D	R	1	10.3.12
PCI Header Type	Index 0E	R	1	10.3.13
PCI Built-in Self-Test (BIST) Control	Index 0F	R	1	10.3.14
PCI Interrupt Line	Index 3C	R	1	10.3.15
PCI Interrupt Pin	Index 3D	R	1	10.3.16
PCI MIN_GNT	Index 3E	R	1	10.3.17
PCI MAX_LAT	Index 3F	R	1	10.3.18
PCI Bus Number	Index 40	R	1	10.3.19
PCI Subordinate Bus Number	Index 41	R	1	10.3.20
PCI Disconnect Counter	Index 42	R/W	1	10.3.21
PCI Special Cycle Address BCR	Index 44 –45	R	2	10.3.22
Memory Bank 0 Starting Address	Index 80	R/W	1	10.3.23
Memory Bank 1 Starting Address	Index 81	R/W	1	10.3.23
Memory Bank 2 Starting Address	Index 82	R/W	1	10.3.23
Memory Bank 3 Starting Address	Index 83	R/W	1	10.3.23
Memory Bank 4 Starting Address	Index 84	R/W	1	10.3.23
Memory Bank 5 Starting Address	Index 85	R/W	1	10.3.23
Memory Bank 6 Starting Address	Index 86	R/W	1	10.3.23
Memory Bank 7 Starting Address	Index 87	R/W	1	10.3.23
Memory Bank 0 Extended Starting Address	Index 88	R/W	1	10.3.24

Table 10-2. Indexed BCR Listing (Continued)

Bridge Control Register	Index	R/W	Bytes	See
Memory Bank 1 Ext Starting Address	Index 89	R/W	1	10.3.24
Memory Bank 2 Ext Starting Address	Index 8A	R/W	1	10.3.24
Memory Bank 3 Ext Starting Address	Index 8B	R/W	1	10.3.24
Memory Bank 4 Ext Starting Address	Index 8C	R/W	1	10.3.24
Memory Bank 5 Ext Starting Address	Index 8D	R/W	1	10.3.24
Memory Bank 6 Ext Starting Address	Index 8E	R/W	1	10.3.24
Memory Bank 7 Ext Starting Address	Index 8F	R/W	1	10.3.24
Memory Bank 0 Ending Address	Index 90	R/W	1	10.3.25
Memory Bank 1 Ending Address	Index 91	R/W	1	10.3.25
Memory Bank 2 Ending Address	Index 92	R/W	1	10.3.25
Memory Bank 3 Ending Address	Index 93	R/W	1	10.3.25
Memory Bank 4 Ending Address	Index 94	R/W	1	10.3.25
Memory Bank 5 Ending Address	Index 95	R/W	1	10.3.25
Memory Bank 6 Ending Address	Index 96	R/W	1	10.3.25
Memory Bank 7 Ending Address	Index 97	R/W	1	10.3.25
Memory Bank 0 Extended Ending Address	Index 98	R/W	1	10.3.26
Memory Bank 1 Ext Ending Address	Index 99	R/W	1	10.3.26
Memory Bank 2 Ext Ending Address	Index 9A	R/W	1	10.3.26
Memory Bank 3 Ext Ending Address	Index 9B	R/W	1	10.3.26
Memory Bank 4 Ext Ending Address	Index 9C	R/W	1	10.3.26
Memory Bank 5 Ext Ending Address	Index 9D	R/W	1	10.3.26
Memory Bank 6 Ext Ending Address	Index 9E	R/W	1	10.3.26
Memory Bank 7 Ext Ending Address	Index 9F	R/W	1	10.3.26
Memory Bank Enable	Index A0	R/W	1	10.3.27
Memory Timing 1	Index A1	R/W	1	10.3.28
Memory Timing 2	Index A2	R/W	1	10.3.29
Memory Bank 0 & 1 Addressing Mode	Index A4	R/W	1	10.3.30
Memory Bank 2 & 3 Addressing Mode	Index A5	R/W	1	10.3.30
Memory Bank 4 & 5 Addressing Mode	Index A6	R/W	1	10.3.30
Memory Bank 6 & 7 Addressing Mode	Index A7	R/W	1	10.3.30
Cache Status	Index B1	R/W	1	10.3.31
RAS# Watchdog Timer	Index B6	R/W	1	10.3.32
Single-Bit Error Counter	Index B8	R/W	1	10.3.33

Table 10-2. Indexed BCR Listing (Continued)

Bridge Control Register	Index	R/W	Bytes	See
Single-Bit Error Trigger Level	Index B9	R/W	1	10.3.34
Bridge Options 1	Index BA	R/W	1	10.3.35
Bridge Options 2	Index BB	R/W	1	10.3.36
Error Enable 1	Index C0	R/W	1	10.3.37
Error Status 1	Index C1	R/W	1	10.3.38
CPU Bus Error Status	Index C3	R	1	10.3.39
Error Enable 2	Index C4	R/W	1	10.3.40
Error Status 2	Index C5	R/W	1	10.3.41
PCI Bus Error Status	Index C7	R/W	1	10.3.42
CPU/PCI Error Address	Index C8 – CB	R/W	4	10.3.43
Single-Bit ECC Error Address	Indx CC – CF	R/W	4	10.3.44
Refresh Timer Divisor	Index D0 – D1	R/W	2	10.3.45
Bridge Chip Set Options 3	Index D4	R/W	1	10.3.46

10.3.3 PCI Vendor ID Register

Index 00 to 01h	Read Only	Reset to 1014h

The vendor ID register is a 16-bit, read-only BCR used to identify the manufacturer of the 660 Bridge. Reading this register always returns 1014h (index 00h = 14h, index 01h = 10h). This is the vendor ID assigned for all IBM–produced PCI devices.

10.3.4 PCI Device ID Register

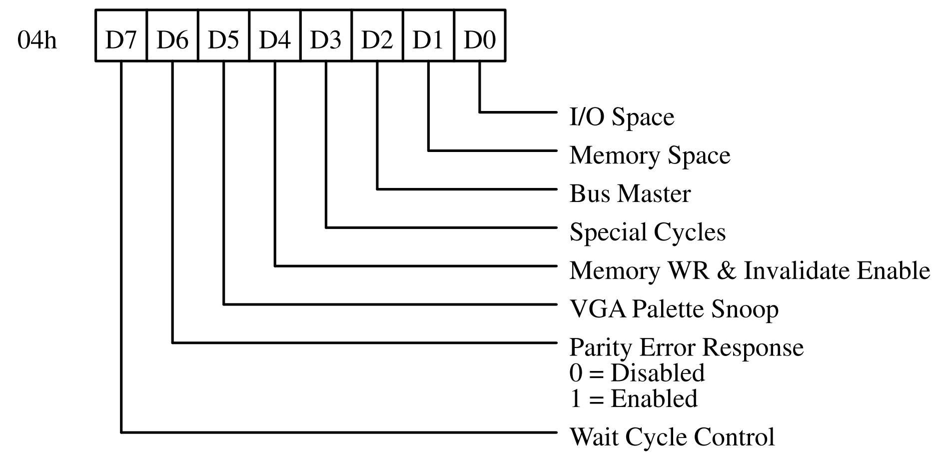
Index 02 to 03h	Read Only	Reset to 0037h

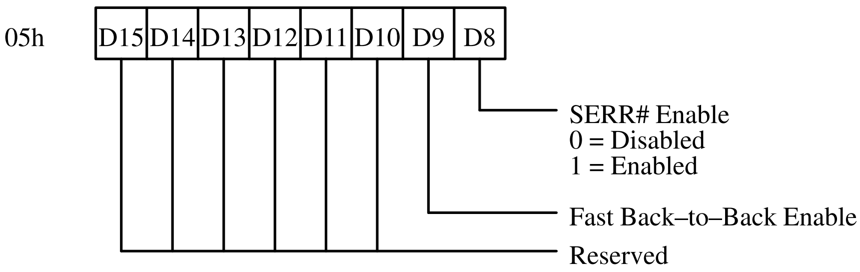
The device ID register is a 16-bit, read-only BCR used to identify the 660 Bridge. Reading this register always returns 0037h (index 02h = 37h, index 03h = 00h).

10.3.5 PCI Command Register

Index 04 to 05	Read/Write	Reset to 0006h

The PCI command register is a 16-bit, read/write BCR used to control the operation of the 664 on the PCI bus.





Bit 0 Controls the response of the 660 Bridge to I/O space accesses. The 660 Bridge does not respond to the I/O address space; therefore, this bit is read-only and always returns 0 when read.

Bit 1 Controls the response of the 660 Bridge to memory space accesses. The 660 Bridge always responds to the memory address space. This bit is read-only.

Bit 2 Enables the 660 Bridge to master cycles on the PCI bus. The 660 Bridge is always enabled as a PCI bus master; therefore, this bit is read-only and returns 1 when read.

Bit 3 Enable special cycle operations. The 660 Bridge does not respond to special cycle operations; therefore, this bit is read-only and returns 0 when read.

Bit 4 Enable memory write and invalidate command support. The 660 Bridge does not support caching on the PCI bus; therefore, this bit is read-only and returns 0 when read.

Bit 5 Enable special palette snooping. The 660 Bridge is not a VGA device; therefore, this bit is read-only and returns 0 when read.

Bit 6 Parity error response. This bit is enabled for all types of PCI bus parity errors, including the following:
1) PCI data bus parity errors while PCI master (reads) (see Section 8.3.3)
2) PCI data bus parity errors while PCI target (writes) (see Section 8.3.3)
3) PCI address bus parity errors

When parity error response is disabled (set to 0) detection of these errors is masked.

Bit 7 Address stepping wait states. The 660 Bridge does not require the use of AD stepping; therefore, this bit is read-only and returns 0 when read.

Bit 8 Enable PCI_SERR#. Enables driving PCI_SERR# when PCI bus address parity error is detected. PCI command register bit 6 must also be enabled.

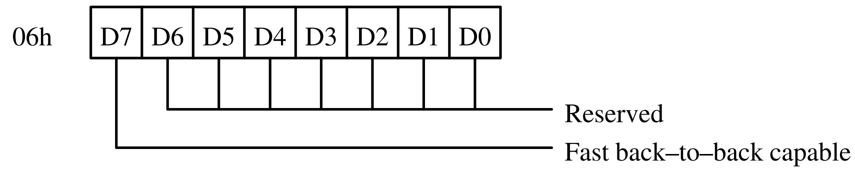
Bit 9 Fast back-to-back write enable. The 660 Bridge does not support fast back-to-back write cycles to different devices on the PCI bus; therefore, this bit is read-only and returns 0 when read.

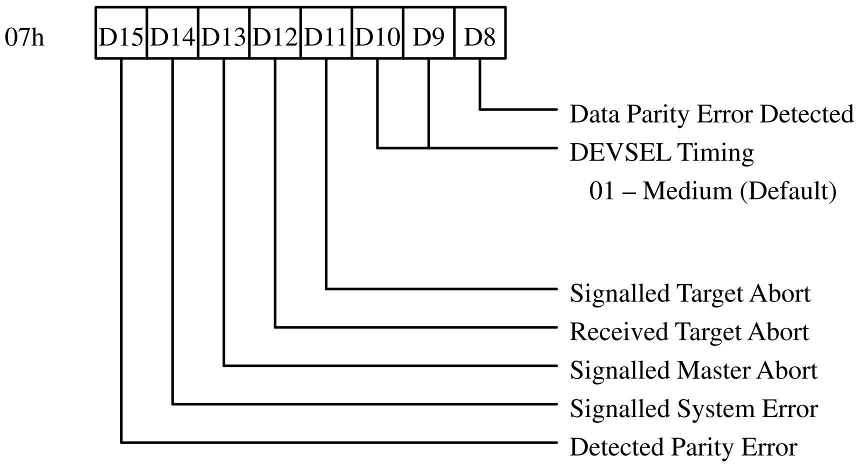
Bits 15:10 Reserved, returns 000000b when read.

10.3.6 PCI Device Status Register

Index 06 to 07h	Read/Bit reset	Reset to 0200h

This 16-bit, read/bit–reset BCR records status information for PCI-related events. Bits in this register can only be set as a result of a specific event occurring on the PCI bus, and they can only be reset by the CPU. To reset any bit, a 1 is written to the specific bit location. A 0 written to a specific bit location leaves that bit unaltered.





Bits 6:0	Reserved, returns 0000000b when read.
Bit 7	Fast back-to-back capable. This bit is read-only and is a 0 to indicate that fast back-to-back cycles are not supported when the 660 Bridge is the PCI target. For example, when a PCI master accesses system memory.
Bit 8	Data parity error detected. This bit is set if both of the following conditions are true: 1. The 660 Bridge is the master when a data bus parity error is detected during a PCI read cycle, or the 660 Bridge is the master when PCI_PERR# is sampled active during a PCI write cycle. 2. PCI command register bit 6 is set to 1. If this bit is set, a TEA#/MCP# to the processor is generated if enabled. Writing a 1 to this bit clears this bit to 0.

Bits 10:9 PCI_DEVSEL# response timing. The 660 Bridge asserts PCI_DEVSEL# in the second clock following a PCI_FRAME# generated by a PCI bus master attempting to access memory. These bits always return 01. These bits are read-only.

Bit 11 This bit is set whenever the 660 Bridge terminates a PCI cycle for which it is the target with target abort.

Bit 12 This bit is set whenever a PCI cycle for which the 660 Bridge is the master is terminated with target abort.

Writing a 1 to this bit clears this bit to 0.

Bit 13 Signalled master abort. This bit is set whenever the 660 Bridge terminates a PCI cycle for which it is the master with master abort during a CPU to PCI memory or I/O cycle.

Writing a 1 to this bit clears this bit to 0.

Bit 14 Signalled system error. If this bit is set, the 660 Bridge asserts PCI_SERR#. This bit is set if both of the following conditions are true:
1) The PCI_SERR# enable bit is set (PCI command register bit 8).
2) Certain errors are detected during a PCI transaction while the Bridge is the PCI target. Not all errors cause this bit to be set.

Writing a 1 to this bit clears this bit to 0.

Bit 15 Detected parity error. This bit is set whenever the 660 Bridge detects a PCI bus parity error. This bit is not maskable—if an error is detected, this bit is set regardless of any control bits. The following events set this bit:
1) PCI address bus parity error detected when an external PCI master accesses system memory.
2) PCI data bus parity error detected when an external PCI master writes to system memory.
3) PCI data bus parity error detected when the 660 bridge masters a PCI read cycle.

Writing a 1 to this bit clears this bit to 0.

10.3.7 Revision ID

Index 08h	Read Only	Reset to 02h

The revision ID register is an 8-bit, read-only BCR used to hold the current incremental revision number of the 664. For revision 1.2 of the 664 (used in revision 2.2 of the 660) this register returns 02h.

10

10.3.8 PCI Standard Programming Interface

Index 09h	Read Only	Reset to 00h

The PCI standard programming interface register is an 8-bit, read-only BCR used to hold programing interface information. Since the 660 Bridge does not support a programming interface, this register returns 00h.

10.3.9 PCI Subclass Code

Index 0Ah	Read Only	Reset to 00h

The PCI subclass code register is an 8-bit, read-only BCR used to hold device class information. This register returns 00h, indicating that the 660 Bridge is a host bridge.

10.3.10 PCI Class Code

Index 0Bh	Read Only	Reset to 06h

The PCI class code register is an 8-bit, read-only BCR used to hold device class information. This register returns 06h, indicating that the 660 Bridge is a bridge between the PCI bus and the 60X CPU bus.

10.3.11 PCI Cache Line Size

Index 0Ch	Read Only	Reset to 00h

The PCI cache line size register is an 8-bit, read-only BCR used to hold the size of a PCI cache line. Since the 660 Bridge does not support a PCI cache, this register returns 00h.

10.3.12 PCI Latency Timer

Index 0Dh	Read Only	Reset to 00h

The PCI latency timer register is an 8-bit, read-only BCR used to hold the value of the PCI latency timer. Because the 660 Bridge can only burst two PCI cycles as master (only with the 604 store multiple instruction), this timer is not implemented and this register returns 00h.

10.3.13 PCI Header Type

Index 0Eh	Read Only	Reset to 00h

The PCI header type register is an 8-bit, read-only BCR used to describe the header region of PCI configuration space. The 660 Bridge implements a standard PCI header and this register returns 00h.

10.3.14 PCI Built-in Self-Test (BIST) Control

Index 0Fh	Read Only	Reset to 00h

The PCI built–in self-test control register is an 8-bit, read-only BCR used for control and status of BIST. The 660 Bridge does not implement BIST and this register returns 00h.

10.3.15 PCI Interrupt Line

Index 3Ch	Read Only	Reset to 00h

The PCI interrupt line register is an 8-bit, read-only BCR used for PCI interrupt routing information. The 660 Bridge does not implement this feature and this register returns 00h.

10.3.16 PCI Interrupt Pin

Index 3Dh	Read Only	Reset to 00h

The PCI interrupt pin register is an 8-bit, read-only BCR used to indicate the PCI interrupt pin to be used. The 660 Bridge does not generate PCI interrupts and this register returns 00h.

10.3.17 PCI MIN_GNT

Index 3Eh	Read Only	Reset to 00h

The PCI MIN_GNT register is an 8-bit, read-only BCR used to specify the minimum setting for the PCI latency timer. The 660 Bridge does not implement the PCI latency timer and this register returns 00h.

10.3.18 PCI MAX_LAT

Index 3Fh	Read Only	Reset to 00h

The PCI MAX_LAT register is an 8-bit, read-only BCR used to specify the maximum setting for the PCI latency timer. The 660 Bridge does not implement the PCI latency timer and this register returns 00h.

10.3.19 PCI Bus Number

Index 40h	Read Only	Reset to 00h

The PCI bus number register is an 8-bit, read-only BCR used to identify the number of the bus controlled by the 660 Bridge. This register returns 00h indicating that the 660 Bridge controls PCI bus number 0. Peer bridges are not supported (for example, a second 60X to PCI bridge on the 60X CPU bus); therefore, this register value is read-only. All CPU accesses to CONFIG_DATA when CONFIG_ADDRESS specifies the bus number as 0 result in the 660 Bridge running a type 0 configuration cycle on its PCI bus.

10.3.20 PCI Subordinate Bus Number

Index 41h	Read Only	Reset to 00h

The PCI subordinate bus number register is an 8-bit, read-only BCR. Since the 660 Bridge does not support peer PCI bridges (only hierarchical), this register is unused. All CPU accesses to CONFIG_DATA when the CONFIG_ADDRESS specifies a bus number other than 0 cause the 660 Bridge to run a type 1 configuration cycle on its PCI bus.

10

10.3.21 PCI Disconnect Counter

Index 42h	Read/Write	Reset to 00

This BCR determines the maximum number of PCI clocks (from 1 to 255) that a PCI master can burst access system memory before a target disconnect is initiated. Loading the counter with 00 disables it. The counter begins to count down when FRAME# is sampled active, and is reloaded between PCI to memory accesses. If the counter reaches 0, the 660 Bridge target disconnects the PCI bus master.

10.3.22 PCI Special Cycle Address Register

Index 44 to 45h	Read Only	Reset to 0000h

The PCI special cycle address register is a 16-bit, read-only BCR. It is not implemented and returns 0000h.

10.3.23 Memory Bank Starting Address

Index 80 to 87h	Read/Write	Reset to 00h (each BCR)

This array of eight BCRs (along with the eight extended starting address registers) contains the starting address for each memory bank. Each pair of registers maps to the corresponding RAS# decode. For example, RAS[4]# corresponds to the BCRs at index 84h and 8Ch. The eight least-significant bits of the bank starting address are contained in the starting address register, and the most-significant bits come from the corresponding extended starting address register. The starting address of the bank is entered with the least significant 20 bits truncated. These BCRs must be programmed in conjunction with the ending address and extended ending address registers.

Program the banks in ascending order, such that (for n = 0 to 6) the starting address of bank n+1 is higher than the starting address of bank n. Each bank must be located in the 0 to 1G address range (see section 5.3.13).

(Start address in MBs)

Bit	Description
D7	A27 of start address (128MB+)
D6	A26 of start address (64MB+)
D5	A25 of start address (32MB+)
D4	A24 of start address (16MB+)
D3	A23 of start address (8MB+)
D2	A22 of start address (4MB+)
D1	A21 of start address (2MB+)
D0	A20 of start address (1MB+)

10.3.24 Memory Bank Extended Starting Address

Index 88 to 8Fh	Read/Write	Reset to 00h (each BCR)

This array of eight BCRs (along with the eight starting address registers) contains the starting address for each memory bank. These BCRs contain the most-significant address bits of the starting address of the corresponding bank.

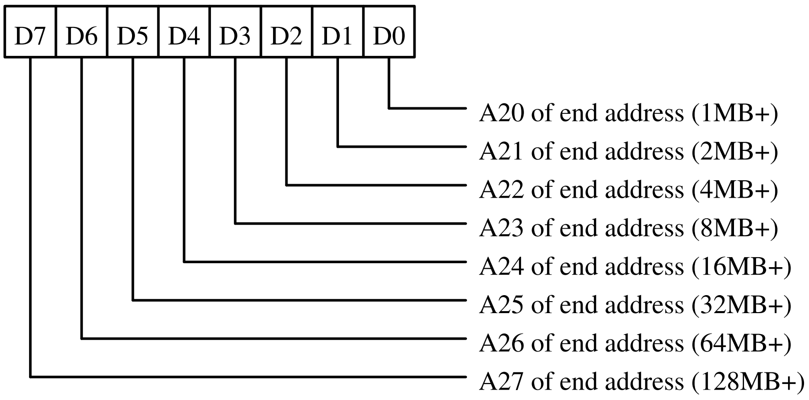
(Start address in MBs)

Bit	Description
D7–D2	Reserved
D1	A29 of start address (512MB+)
D0	A28 of start address (256MB+)

10.3.25 Memory Bank Ending Address

Index 90 to 97h	Read/Write	Reset to 00h (each BCR)

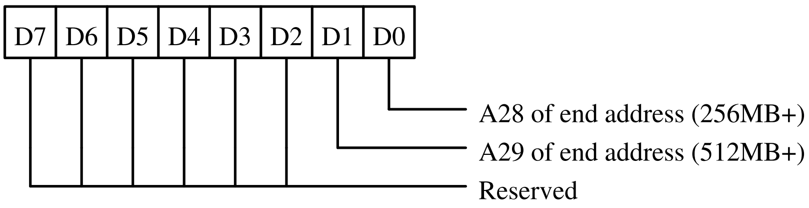
This array of eight BCRs (along with the eight extended starting address registers) contains the ending address for each memory bank. Each pair of registers maps to the corresponding RAS# decode. For example, RAS[4]# corresponds to the BCRs at index 94h and 9Ch. The eight least-significant bits of the bank ending address are contained in the ending address register, and the most-significant bits come from the corresponding extended ending address register. The ending address of the bank is entered as the address of the next highest memory location minus 1, with the least significant 20 bits truncated. For an xMB bank program, the end address is equal to the start address (from Section 5.3.9) + x–1. Each bank must be located in the 0 to 1G address range. These BCRs must be programmed in conjunction with the ending address and extended ending address registers. See section 5.3.13.



10.3.26 Memory Bank Extended Ending Address

Index 98 to 9Fh	Read/Write	Reset to 00 (each BCR)

This array of eight 8-bit, read/write registers (along with the eight ending address registers) contains the ending address for each memory bank. These BCRs contain the most-significant address bits of the ending address of its bank.

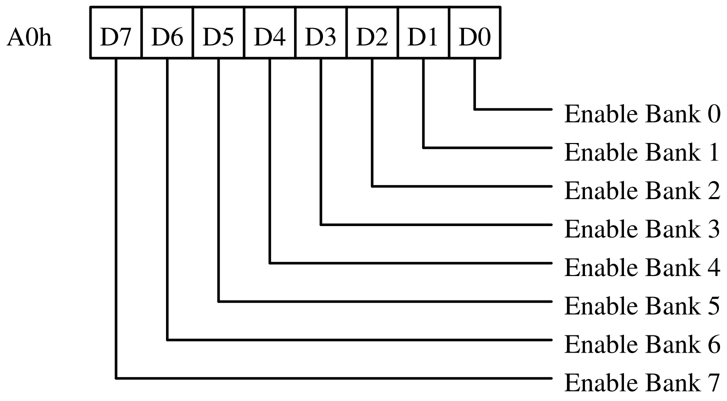


10.3.27 Memory Bank Enable

Index A0h	Read/Write	Reset to 00h

This BCR contains a control enable for each bank of memory. Each bank of memory must be enabled for proper refreshing. For each bit, a 0 disables that bank of memory and a 1 enables it.

This register must be programmed in conjunction with the starting address and ending address registers. If a bank is disabled by this register, the corresponding starting and ending address register entries become don't cares.



10.3.28 Memory Timing Register 1

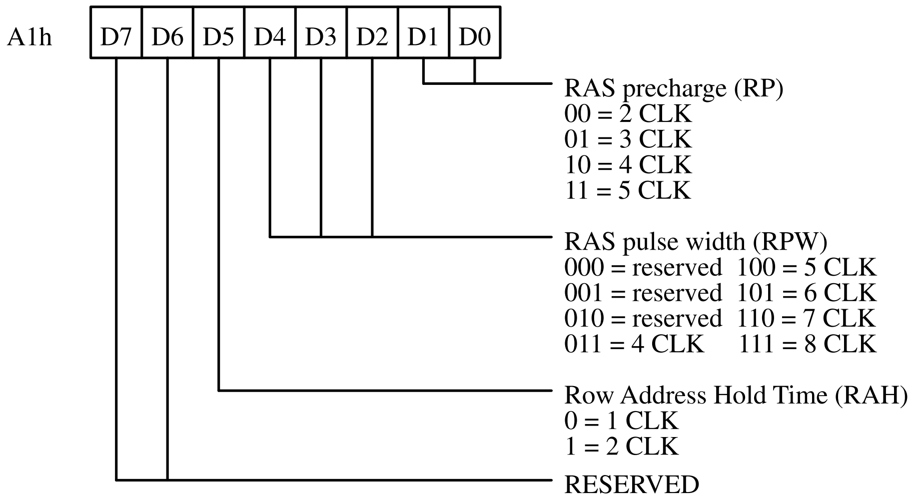
Index A1h	Read/Write	Reset to 3Fh

This BCR determines the timing of RAS# signal assertion for memory cycles. RAS# timing must support the worst-case timing for the slowest SIMM installed in the system. See Section 5.2.1.

Bits 1:0 These bits control the number of CPU clocks for RAS# precharge.

Bits 4:2 These bits control RAS# pulse width except on refresh. For refresh, the RAS# pulse width is hard-coded to three PCI clocks.

Bit 5 This bit controls the number of CPU clocks that the row address is held following the assertion of RAS#.

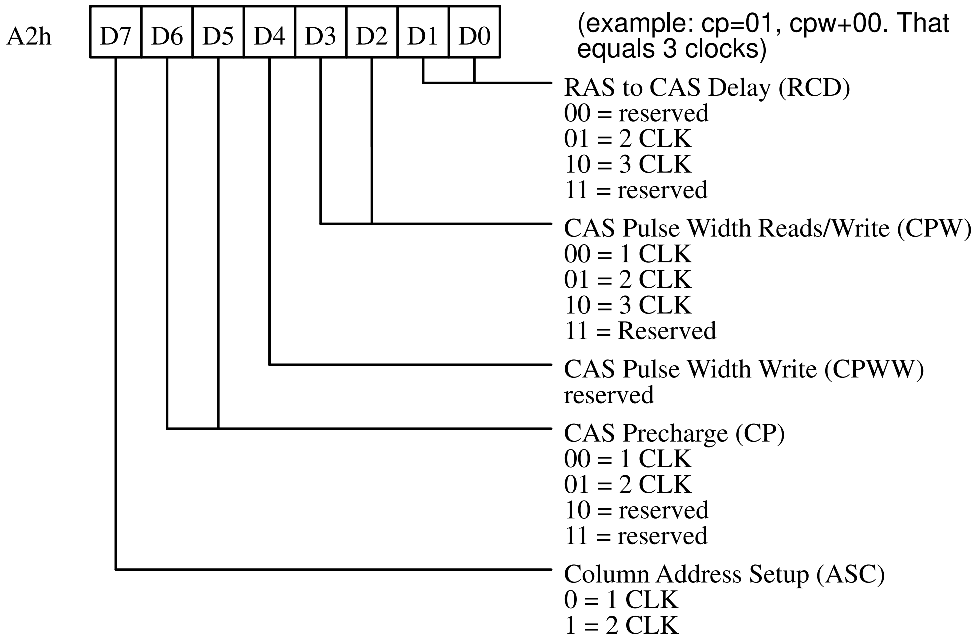


10.3.29 Memory Timing Register 2

Index A2h	Read/Write	Reset to AEh

This BCR determines the timing of CAS# signal assertion for memory cycles. CAS# timing must support the worst-case timing for the slowest DRAM installed in the system. See Section 5.2.1.

When using the 660 with a CPU:PCI bus frequency ratio of 1:1, then CP plus CPW must be set less than or equal to 3 clocks total (not the number of settings).

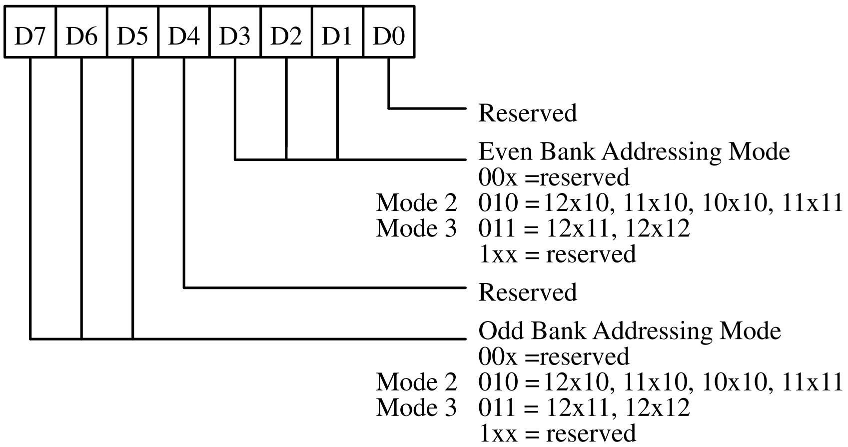


10

10.3.30 Memory Bank Addressing Mode Registers

Index A4 to A7h	Read/Write	Reset to 44h (each BCR)

This array of four 8-bit, read/write BCRs defines the format of the row and column addressing of each DRAM memory bank.

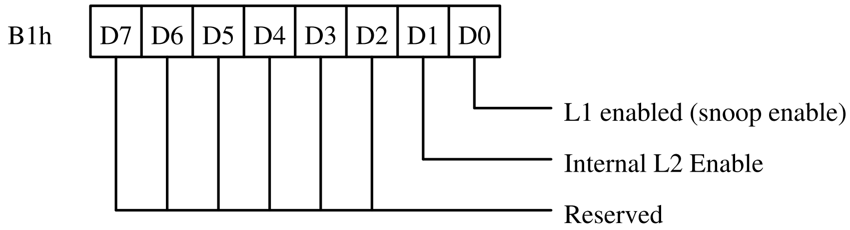


Register	Bits	Memory Bank	Bits	Memory Bank
A4h	3:0	0	7:4	1
A5h	3:0	2	7:4	3
A6h	3:0	4	7:4	5
A7h	3:0	6	7:4	7

10.3.31 Cache Status Register

Index B1h	Read/Write	Reset to 43h

This 8-bit, read/write BCR describes the status of the level one (L1) and level two (L2) caches and other features.



Bit 0 L1 enabled: Snooping is always enabled and this bit is hardcoded to a 1.
0 : L1 disabled
1 : L1 enabled

Bit 1 Internal L2 Enable: This bit is used with the L2 Cache Enable bit (bit 6 of the system control 81C BCR, section 10.2.2.3) to enable the internal L2 cache. The internal L2 is enabled only when both bits are set to 1.
0 : Internal L2 disabled.
1 : Internal L2 enabled (iff L2 Cache Enable bit is set).

Bit 2:7 Reserved

10.3.32 RAS# Watchdog Timer Register

Index B6h	Read/Write	Reset to 53h

This BCR limits the maximum RAS# active pulse width. The value of this BCR represents the maximum amount of time that any RAS# can remain active in units of eight CPU bus clocks. The timer (down-counter) associated with this BCR is reloaded on the assertion of any RAS# line. On expiration of the timer, the 660 Bridge drops out of page mode to deassert the RAS# lines.

In response to the RESET# signal, this register is reset to 53h. This value results in a maximum RAS# active time of just under 10us at 66MHz. This is the value required by most 4-byte and 8-byte SIMMs. The value of the BCR must be reprogrammed if the CPU bus frequency is not 66MHz or a different RAS# pulse width is required.

10.3.33 Single-Bit Error Counter Register

Index B8h	Read/Write	Reset to 00h

This 8-bit, read/write BCR contains the count of the number of ECC single-bit errors that have occurred. This register can be written by the CPU to set or clear the counter value. If a value greater than the trigger level register (see Section 10.3.34) is written, the TEA# or MCP# error is asserted.

The bits of this register are in reversed significance.

D7	D6	D5	D4	D3	D2	D1	D0
LSB							MSB

The order of the bits should be reversed before interpreting the byte as a number.

D0	D1	D2	D3	D4	D5	D6	D7
MSB							LSB

10.3.34 Single-Bit Error Trigger Level Register

Index B9h	Read/Write	Reset to 00h

This 8-bit, read/write BCR contains the threshold value for generating an error to the CPU on ECC single-bit errors. When the single-bit error counter register value equals the value in this register, a single-bit ECC error (trigger exceeded) is generated to the CPU.

If the single-bit error trigger level register is set to 00h, no single-bit ECC error is ever generated to the CPU. Setting this BCR to 0 does not inhibit the reporting of multiple-bit ECC error.

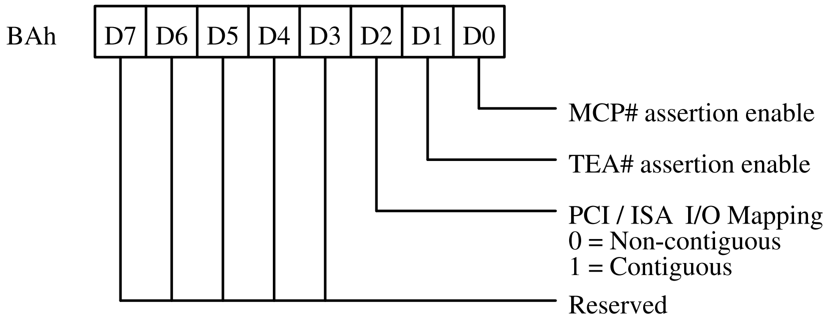
If, during DRAM read, a single-bit error is detected, the 660 corrects the data supplied to the reading agent, and increments the single-bit error counter BCR. The 660 does not write the corrected data back to the DRAM. Repeated reads of a location that has a single-bit error will eventually cause an MCP# when the single-bit error trigger level is exceeded, even though the data is corrected before being supplied to the requestor. Therefore some applications may wish to set the error trigger level BCR to 0 to prevent this scenario.

The bit significance of this BCR is not reversed as is BCR(B8), but uses the same bit significance as the other BCRs.

10.3.35 Bridge Chip Set Options 1

Index BAh	Read/Write	Reset to 04h

This 8-bit, read/write BCR controls various operating parameters of the 660 Bridge.



Bit 0 MCP# assertion enabled: When set, the 664 will assert the machine check pin to the 603/604 CPU or drive an interrupt to the 601 CPU with intent to error terminate. The state of this bit does not affect the operation of the error detection and handling logic in any other way.

0 : MCP# assertion disabled: In response to a system error, the 664 will neither assert MCP# (603/604) nor generate an interrupt (601) with the intent to error terminate. Setting this bit to 0 does not automatically cause the 660 to assert TEA#.

1 : MCP# assertion enabled: In response to a system error, the 664 is allowed to assert MCP# (603/604) or generate an interrupt (601) with the intent to error terminate.

Bit 1 TEA# assertion enabled: When enabled, the 664 is allowed to assert TEA# when an error condition is detected.
XATS# assertion will cause a bus hang if the cycle is not terminated so the 664 will still terminate an XATS# cycle with TEA# to prevent a bus hang condition, even if this bit is set to 0. Note that this bit can also be written by means of the system control 8000 081Ch BCR.

1 : TEA# assertion enabled.

Bit 2 PCI/ISA I/O mapping. This bit can also be written by means of the I/O map type register (address 8000 0850h).

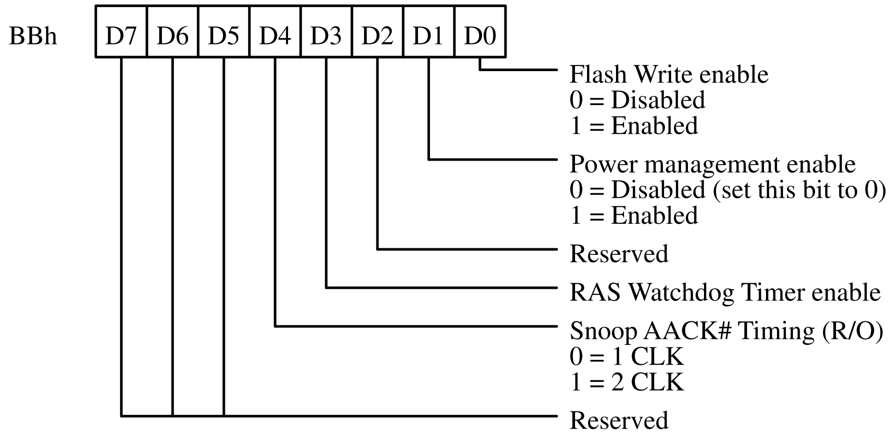
0 : Non–contiguous PCI / ISA address mapping is used. CPU addresses from 2G to 2G + 8M are translated where each 4K of CPU address space is mapped to 32 bytes. The 8M address space is compressed into 64K.

1 : Contiguous PCI/ISA address mapping is used. CPU addresses from 2G to 2G + 8M are not remapped.

Note that the state of bits 0 and 1 do not control MCP# mode. These bits merely enable or disable the assertion of particular outputs. For more information on MCP mode, see section 10.3.46 Bridge Chipset Options 3.

10.3.36 Bridge Chip Set Options 2

Index BBh	Read/Write	Reset to 4Fh



Bit 0 Flash write enable: When the ROM is remotely attached, this bit controls write access to the flash ROM address space (4G – 2M to 4G). When enabled, writes to this space are forwarded to the PCI memory space at the same address. When disabled, writes to this space are treated as no-ops and an error is signalled. After the bit is set to 0 (disabled), it cannot be reset to 1 (enabled).

Bit 1 Power management enable: Power management is not supported. Set this bit to 0.

Bit 2 Reserved. This bit is hardcoded to a 1.

Bit 3 RAS# watchdog timer enable: The RAS# max pulse width is always checked and this bit is hardcoded to 1.

Bit 4 AACK# timing: Controls the minimum number of cycles between assertion of TS# (transfer start) and assertion of AACK# (address acknowledge) by the 660 Bridge for any cycle. The implementation of a 603 processor with its internal logic clocked at the same rate as the 60X CPU bus (1:1) requires this bit to be set. This bit is read-only and is set to the value defined by the configuration strapping options. (Reset value is determined by strapping pin.)

Bit 5 Reserved.

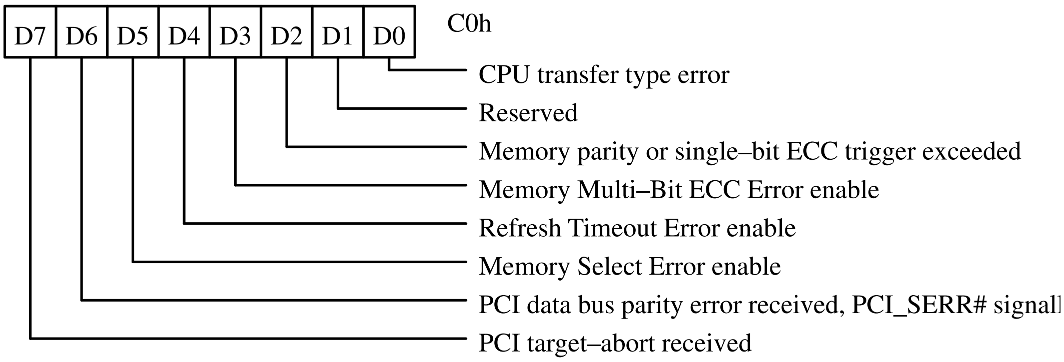
Bit 6 Reserved. This bit is hardcoded to a 1.

Bit 7 Reserved.

10.3.37 Error Enable 1

Index C0h	Read/Write	Reset to 01h

The error report enable 1 register is an 8-bit read/write BCR that selects the system error conditions to which the 660 Bridge responds. A bit set in this register enables the associated error condition to be detected. If the bit is 0, the error will not be detected or reacted to by the 660.



Bit 0 CPU transfer type error enable: This bit enables the detection of unsupported transfer types and XATS# assertion.
0 : Disabled
1 : Enabled

Bit 2 Memory parity or single-bit ECC error enable. Detection of parity or single-bit ECC errors is:
0 : Disabled
1 : Enabled

Bit 3 Memory multi-bit ECC error enable:
0 : Disabled
1 : Enabled

Bit 4 Refresh timeout error enable: The 660 Bridge does not allow refresh timeout errors to occur and this bit is hardcoded as a 0.

Bit 5 Memory select error enable:
0 : Disabled
1 : Enabled

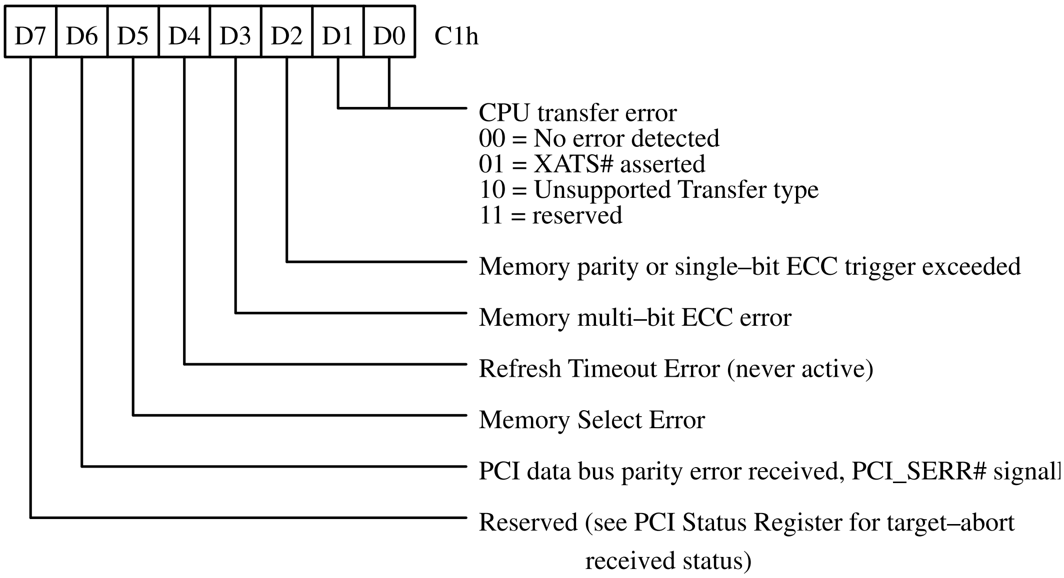
Bit 6 PCI data bus parity error received, PCI_SERR# signalled. This bit enables the 660 Bridge to drive PCI_SERR# in addition to PCI_PERR# when a PCI data bus parity error is detect when the PCI target (PCI write to system memory).
0 : Disabled
1 : Enabled

Bit 7 PCI target abort received. This bit enables detection of target aborts that occur while the 660 Bridge is the PCI master (CPU accesses to PCI).
0 : Disabled
1 : Enabled

10.3.38 Error Status 1

Index C1h	Read/Write	Reset to 00h

The error status 1 register is an 8-bit read/write BCR that contains status on error conditions that have been detected. Bits in this register can only be set to 1 as a result of a system error occurring. To reset any bit, the CPU can write a 1 to that bit location. Writing a 0 to a specific bit location leaves that bit unaltered. (Writing a 1 to bit 0 or bit 1 clears bit 0 and bit 1.) Bits cannot be reset if the error condition persists.



Bits 1:0 CPU transfer error:
00 : No error detected
01 : XATS# sampled asserted.
10 : Illegal transfer attribute encoding detected on the CPU bus.
11 : Reserved
Note: These bits can also be reset by means of the unsupported transfer type read and clear register.

Bit 2 If parity memory is implemented, this bit is set on a parity error. If ECC memory has been implemented, this bit is set in response to exceeding the single-bit ECC error trigger threshold. This bit can also be reset by means of the memory parity error status BCR (8000 0840h).

Bit 3 If ECC memory has been implemented, this bit is set in response to the detection of a multi–bit ECC error.

Bit 4 Refresh timeout errors never occur; therefore, hardcoded 0.

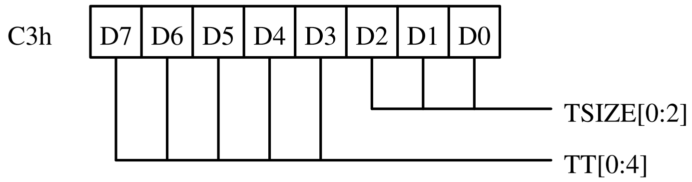
Bit 5 Memory select error: The CPU has run a memory cycle from 0 to 2G, but no memory is populated at that address.

Bit 6 PCI_SERR# has been asserted is response to the detection of a data bus parity error during a PCI master write to system memory.

10.3.39 CPU Bus Error Status

Index C3h	Read Only	Reset: Undefined

The CPU bus error status BCR contains information regarding the CPU bus when an error is detected during a CPU cycle. This register is only valid if register C7 bit 4 is cleared.



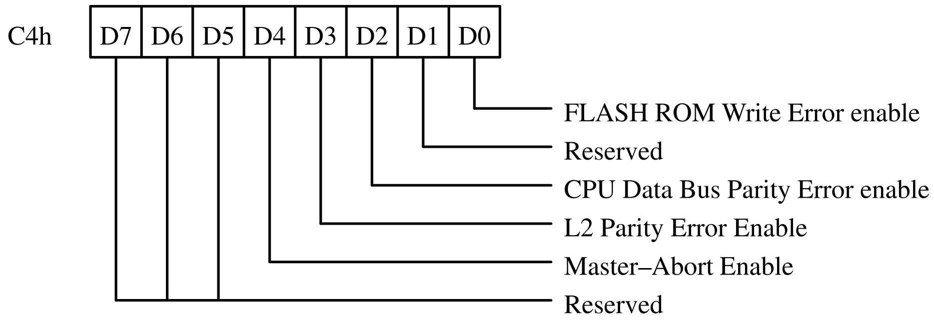
Bits 2:0 TSIZ[0:2]: These bits hold the value of TSIZ when the system error occurred.

Bits 7:3 TT[0:4]: These bits hold the value of TT when the system error occurred.

10.3.40 Error Enable 2

Index C4h	Read/Write	Reset to 00h

The error report enable 2 BCR selects which additional system error conditions get reported. A bit set in this register allows errors in the error status 2 register to generate TEA# or MCP#.



10

Bit 0 Flash/ROM write error enable: When this bit is set, attempts to write to the flash/ROM when it is locked result in an error.

Bit 2 CPU data bus parity error enable: When this bit is set, parity errors on the CPU data bus are detected. See sections 1.11.9 and 6.4.

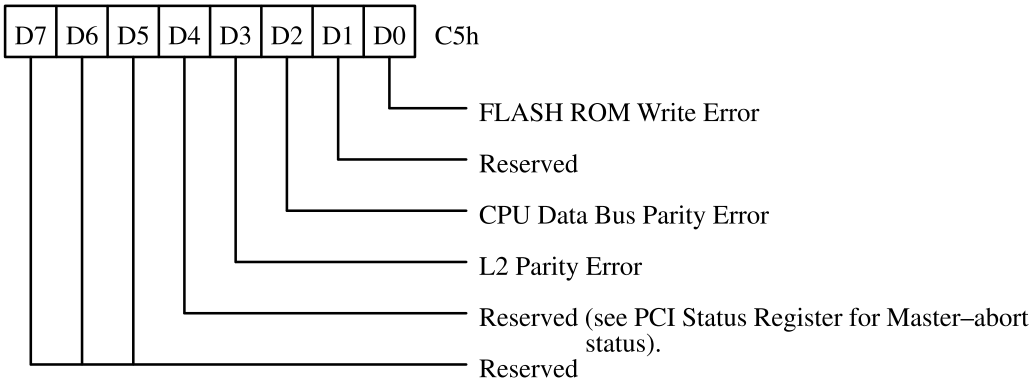
Bit 3 L2 cache parity error enable: When this bit is set, parity errors during CPU reads from the L2 cache are detected. If the L2 is not present or does not implement parity, this bit should not be enabled. See sections 1.11.9 and 6.4.

Bit 4 Master Abort Enable: When this bit is set, PCI master aborts that occur while the 660 is the PCI master for CPU to PCI access are detected as errors. Note that this bit should not be enabled when running an operating system that does not consider master aborts to be errors.

10.3.41 Error Status 2

Index C5h	Read/Write	Reset to 00h

The error status 2 register is an 8-bit read/write BCR that contains additional error conditions status information. Bits in this register can only be set to 1 as a result of a system error occurring. A 1 written to a specific bit location resets the bit to 0. A 0 written to a specific bit location leaves that bit unaltered.

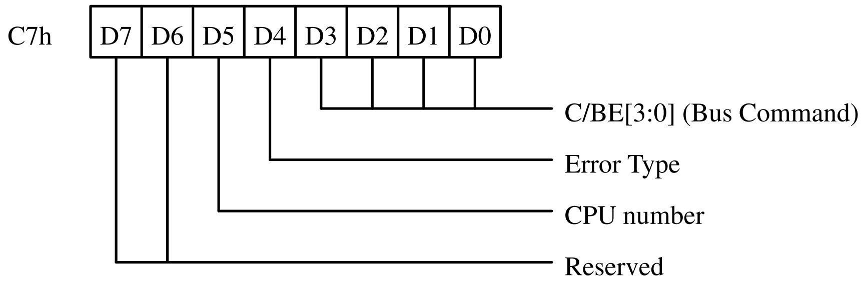


Bit 0	Flash ROM write error: When set, the 664 has detected an attempt to write to the flash/ROM after the flash/ROM write lock–out bit has been set.
Bit 1	Reserved
Bit 2	CPU data bus parity error: When set, a parity error on the CPU data bus has been detected during a transfer between the CPU and the 660 Bridge. Note that L2 parity errors also cause the CPU bus data parity error bit to be set.
Bit 3	L2 cache parity error: This bit is set if a parity error is detected on a CPU read from the L2 cache. Note that this bit can also be reset by means of the L2 cache parity error read and clear register.
Bit 7:4	Reserved

10.3.42 PCI Bus Error Status

Index C7h	Read Only	Reset: Undefined

The PCI bus error status register is an 8-bit read-only BCR that contains information regarding the status of the PCI bus and other status information when an error is detected.



Bits 3:0 — PCI_C/BE#[3:0] : These bits hold the value of the PCI_C/BE#[3:0] bits during the address tenure (bus command) of the transaction that caused a PCI bus error. These bits are only valid if an error occurred during a PCI to memory transaction (BCR(C7) bit 4 = 1)

Bit 4 — Error Type: This bit is set if an error occurred while the 660 is the PCI target. This bit is cleared if an error occurred during a CPU cycle.

Bit 5 — CPU Number: This bit is 0 if CPU #1 (the parked CPU) is the CPU bus master when an error occurs. This bit is 1 if CPU #2 is the CPU bus master when an error occurs. This bit is only valid if the error occurred during a CPU to PCI transaction (BCR(C7) bit 4 = 0).

10.3.43 CPU/PCI Error Address

Index C8 to CBh	Read Only	Reset: Undefined (each BCR)

The CPU/PCI bus error address registers are four 8-bit read/write BCRs that contain the address of the CPU or PCI transaction that generated an error. Once an address has been latched, it remains in these registers until the corresponding error detect bit(s) in the error status 1 register (C1h) or the PCI device status registers (06h, 07h) have been reset.

C8h	CPU_ADDR[24:31] or PCI_AD[7:0]
C9h	CPU_ADDR[16:23] or PCI_AD[15:8]
CAh	CPU_ADDR[8:15] or PCI_AD[23:16]
CBh	CPU_ADDR[0:7] or PCI_AD[31:24]

10.3.44 Single-Bit ECC Error Address

Index CC to CFh	Read Only	Reset: Undefined (each BCR)

The single-bit ECC error address registers are four 8-bit read/write BCRs that contain the address of the memory transaction that generated a single-bit ECC error. This register is updated each time a single-bit ECC error is detected while the single-bit error trigger level is exceeded, and it is not updated unless the trigger level has been exceeded and the transaction caused a single-bit ECC error.

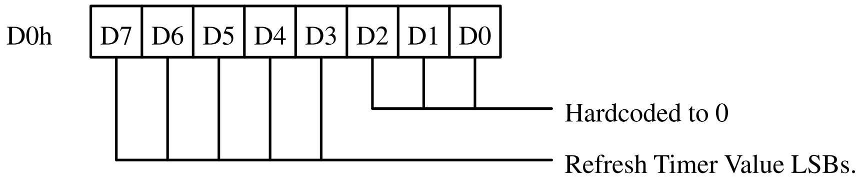
The value stored in this register is the offset into system memory. This means that if the error occurred during a CPU access, then the CPU address is saved. If the error occurred during a PCI access, then the PCI address minus 2G is saved.

CCh	Most-significant byte of address
CDh	Second-most-significant byte of address
CEh	Third-most-significant byte of address
CFh	Least-significant byte of address

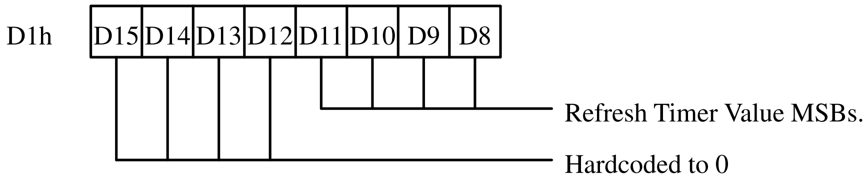
10.3.45 Refresh Timer Divisor Register

Index D0 to D1	Read/Write	Reset to F8 (D0) and 01 (D1)

The refresh timer register is a 16-bit BCR that determines the memory refresh rate. Typical refresh rates are 15.1 to 15.5 microseconds. If all DRAM in the system supports extended (slow) refresh, the refresh rate can be slower. The refresh timer is clocked by the PCI clock input to the 664. The reset value of 01F8h provides a refresh rate of 15.1 microseconds while the PCI clock is 33MHz. (01F8h equals 504 times 30ns equals 15.12us.) Bits 3–11 of the timer allow timer values from 8 to 4096.



Bits 7:3 Refresh timer (7:3) : These are the five least-significant bits of the refresh timer value.

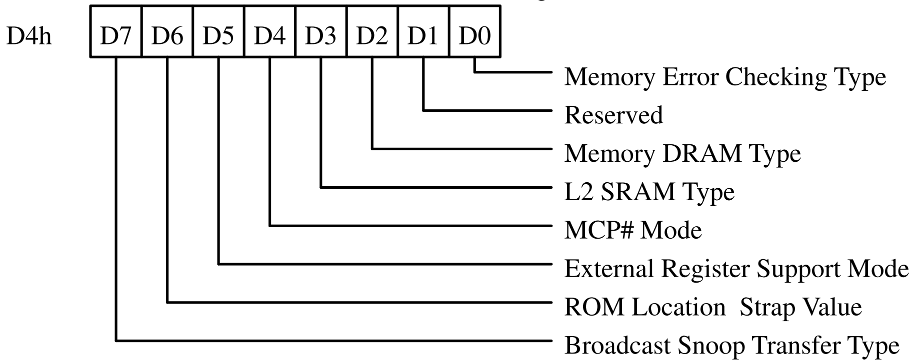


Bits 11:8 Refresh timer (11:8) : These are the four most-significant bits of the refresh timer value.

10.3.46 Bridge Chip Set Options 3 Register

Index D4h	Read/Write	Reset to 00h

This 8-bit read/write BCR controls various 660 Bridge functions.



Bit 0 Memory error checking type.
0 = parity mode (does not enable or disable error checking)
1 = ECC mode (setting this bit changes the DRAM access timing, even if ECC is disabled)

Bit 1 Reserved

Bit 2 DRAM memory type
0 = standard page mode DRAM
1 = hyper page mode (extended data out EDO) DRAM

Bit 3 Data SRAM type
0 = asynchronous
1 = synchronous

Bit 4 MCP# mode. This bit selects either the 604/603 MCP# error reporting mode or the 601 TEA# and interrupt error reporting mode. This is the mode select, not the output enable bit for the signal. See section 10.3.35.
0 = MCP# mode selected. Use with 603/604.
1 = MCP# mode deselected. Errors are reported using the 601 protocol.

Bit 5 External register support mode.
1 = External register mode enabled. Reads to 8000 0814h, 8000 081Ch, and 8000 0850h are forwarded to the PCI bus and ignored by the internal BCRs. Writes are forwarded to the PCI bus and also written to the internal BCRs.
0 = External register mode disabled. Accesses to these three registers go only to the internal BCRs, and are not forwarded to the PCI bus.

Bit 6 Value of the ROM location strapping pin (read–only)
0 = Direct (PCI based).
1 = Remote.

Bit 7 Broadcast snoop transfer type. To maintain memory coherence, set this bit correctly before allowing PCI to memory accesses. See section 4.5.
0 = 601/604 transfer type encodings used for broadcasting PCI to memory accesses to the CPU bus.
1 = 603 transfer type encodings used.

10

Appendix A Timing

A.1 Timing Conventions

Unless otherwise noted, all timing information is given for 660 Bridge operation within the envelope defined by the Recommended Operating Conditions.

Unless otherwise noted, all specifications in this section apply equally to the 663 and the 664.

A.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. Allow for delays between components while constructing timing diagrams for the design of an actual system.

A.1.2 Terms and Definitions

Signal range names used without range indicators refer to the entire group of signals. For example, CPU_DATA refers to the 663 signals CPU_DATA[0:63]. Ranges are expressed as [most-significant bit : least-significant bit].

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

Valid refers to a voltage level above V_H(min) or below V_L(max). Valid does not imply logically true or false, asserted or negated.

A.1.3 Signal Switching Levels for Timing Analysis

Figure A-1 shows typical timing analysis signal switching levels, where V_H and V_L 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 both inputs and outputs for timing analysis only, and do not imply anything about the DC characteristics of the device.

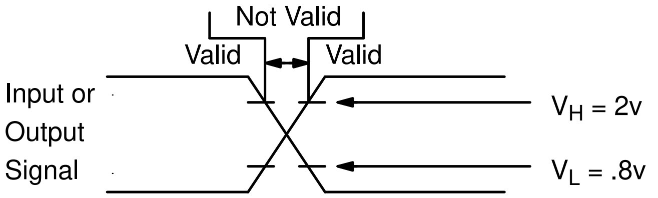


Figure A-1. Switching Levels

A.1.4 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 A-2) 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).

A.1.5 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 A-2) 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.

A.1.6 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 A-2) 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.

A

A.1.7 Output Delay Time

Output delay time is the amount of time required for an output signal to change to a stable valid state following an event. Output delay time (T_{OD} in Figure A-2) 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.

A.1.8 Output Enable Time

Output enable time is the amount of time required for an output signal driver to drive a tristated line to a valid logic level. This time does not take into account the effects of possible pullup resistors connected to the net. Such resistors may improve or degrade the actual time, depending on the situation. Output enable time 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). There are no asynchronous output enables in the 660 Bridge.

A.1.9 Output Tristate 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 tristate hold time (T_{3SH} in Figure A-2) 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 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 is no longer guaranteed to be actively driven to a valid logic level. Note that this specification deals with the time that the output driver remains active following an event which may turn it off, and is computed from the minimum output tristate delay time. The actual output signal may remain valid for some time after this, depending on other conditions.

A.1.10 Output Tristate Delay Time

Output tristate 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 tristate delay time (T_{3SD} in Figure A-2) 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.

A

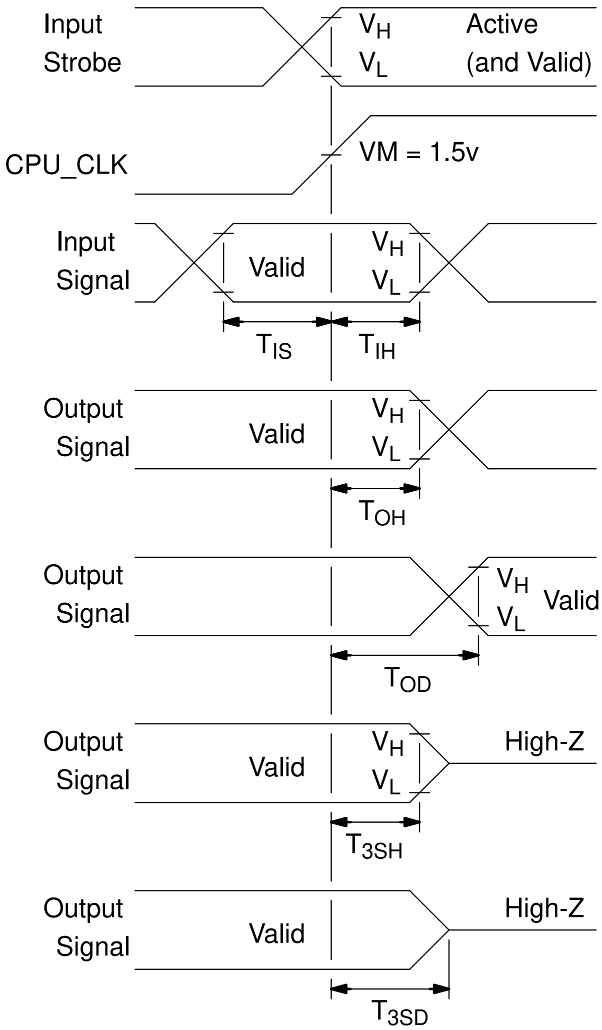


Figure A-2. Signal Timing Conventions

A.2 Clock Considerations

To maintain synchronization between the 660 Bridge and the CPU, certain constraints are placed on the CPU_CLK signal. Unless otherwise noted, all references to CPU_CLK timing refer to the point in time at which the CPU_CLK crosses the VM level, 1.5v.

In general, both the 663 and the 664 are synchronous machines. Inputs are sampled on the rising edge of CPU_CLK, and outputs are updated on the rising edge of CPU_CLK.

A.2.1 660 Bridge CPU_CLK Skew to the Processor SYSCLK

As shown in Figure A-3, the allowed skew between the 664 CPU_CLK and the processor SYSCLK is +/– 1ns. The allowed skew between the 663 CPU_CLK and the processor SYSCLK is +/– 1ns.

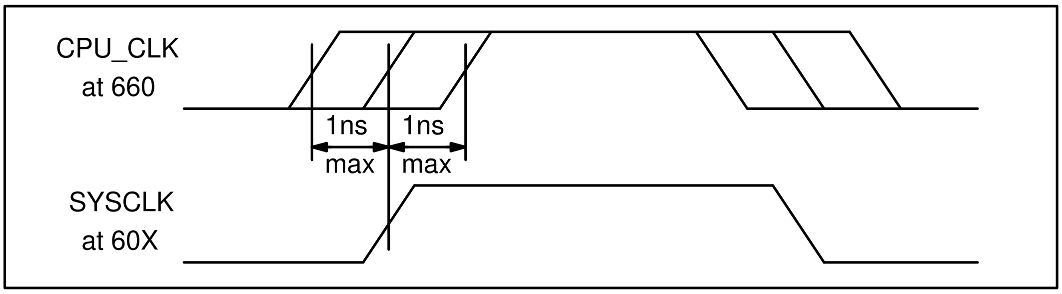


Figure A-3. CPU_CLK to SYSCLK Skew

A.2.2 663 Buffer CPU_CLK Skew to 664 Controller CPU_CLK

The allowed skew between the CPU_CLK at the input to the 664 and the CPU_CLK at the input to the 663 is also +/– 1ns. Thus the skew between any two of the three devices (663, 664, and 60x) is a maximum of +/– 1ns.

A

A.2.3 CPU_CLK Duty Cycle

The CPU_CLK is shown (at the CPU_CLK pin of the 664) in Figure A-4, where T_{CH} is the time that CPU_CLK is high, and T_{CL} is the time that CPU_CLK is low. The allowed duty cycle of CPU_CLK is from 35% to 65%. If the period of CPU_CLK is 15ns, then T_{CH} may range from 5.25ns to 9.75ns.

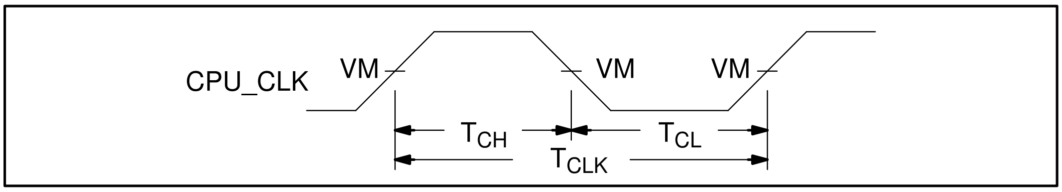


Figure A-4. CPU_CLK Duty Cycle

A.2.4 CPU_CLK to PCI_CLK Skew

The allowed skew of the PCI_CLK at any point in the system to the CPU_CLK at the 660 Bridge is +/– 2ns, as shown in Figure A-5.

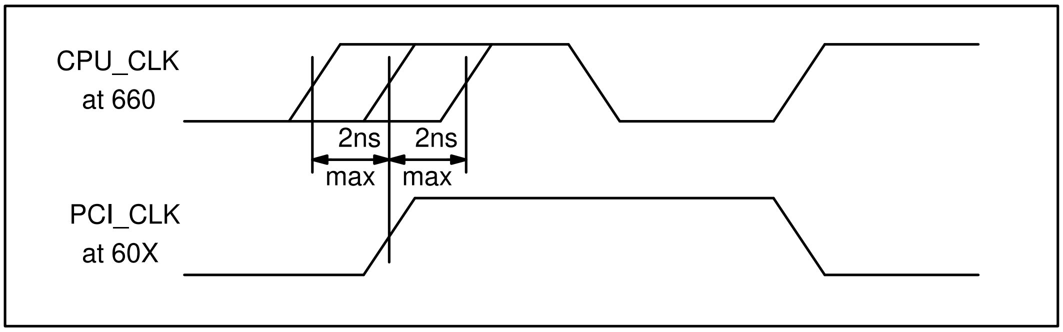


Figure A-5. CPU_CLK to PCI_CLK Skew

A.3 Asynchronous Paths

The 660 Bridge is in general a synchronous device. There are however some asynchronous paths which are only active to speed up the assertion of particular signals. The paths speculatively assert certain signals. Once the clock edge latches in the correct input conditions and the synchronous logic verifies the speculative assertion, the asynchronous path is deactivated and no longer has any effect on the Bridge (until next time).

The even numbered CPU_DATA lines are driven synchronously by the 660. To improve signal quality by reducing simultaneous switching noise, the odd numbered CPU_DATA lines are typically driven asynchronously, in advance of the clock, so that they will switch at a different instant in time. When the clock arrives, the odd numbered CPU_DATA lines (as well as the even numbered lines) are latched onto the 660 outputs.

A

A.4 Power-On Considerations

The 660 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 660 works properly in a system designed to correctly support the 60X CPU and the PCI bus.

The 660 Bridge requires RESET# to be asserted at power-on for a minimum of 1us 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 Bridge are not required to be in any special state during the reset period, but the Bridge 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.

A.5 663 Buffer Timing By Signal

Table A-1. 663 Buffer Timing By Signal

Signal	Pin	I/O	Input (ns)		Output (ns)				
			Setup (min)	Hold (min)	Valid Delay (Max)	Enable Delay (Max)	Hold Time (min)	Hold (to 3S) (min)	Tristate Delay (Max)
AOS_RR_MMRS	166	I	*	*	—	—	—	—	—
C2P_WRL_OPEN	154	I	*	*	—	—	—	—	—
CPU_DATA[00:63][1]	App B	I/O	5.4	0	8.7	10.1	4.7	2.7	8.5
CPU_DATA_OE#	146	I	*	*	—	—	—	—	—
CPU_DPAR[0:7]	App B	I/O	5.4	0	8.7	10.1	4.7	2.7	8.5
CPU_PAR_ERR#	174	O	—	—	*	—	*	—	—
CPU_RDL_OPEN	148	I	*	*	—	—	—	—	—
CRS_C2PWXS	151	I	*	*	—	—	—	—	—
DUAL_CTRL_REF	170	I	*	*	—	—	—	—	—
ECC_LE_SEL	149	I	*	*	—	—	—	—	—
MEM_BE[3:0]	App B	I	*	*	—	—	—	—	—
MEM_CHECK[0:7]	App B	I/O	2.4	0.4	16.8	17.2	4.6	2.2	7.6
MEM_DATA[63:0]	App B	I/O	6.0	0.2	16.8	17.5	4.5	2.2	8.0
MEM_DATA_OE#	145	I	*	*	—	—	—	—	—
MEM_ERR#	171	O	—	—	*	—	*	—	—
MEM_RD_SMPL	147	I	*	*	—	—	—	—	—
MEM_WRL_OPEN	150	I	*	*	—	—	—	—	—
MWS_P2MRXS	152	I	*	*	—	—	—	—	—
PCI_AD[31:0][2]	App B	I/O	7	0	11	13.9	2	2	28
PCI_AD_OE#	144	I	*	*	—	—	—	—	—
PCI_EXT_SEL	153	I	*	*	—	—	—	—	—
PCI_IRDY#	167	I	7	0	—	—	—	—	—
PCI_OL_OPEN	165	I	*	*	—	—	—	—	—
PCI_OUT_SEL	169	I	*	*	—	—	—	—	—
PCI_TRDY#	168	I	7	0	—	—	—	—	—
ROM_LOAD	160	I	*	*	—	—	—	—	—
SBE#	175	O	—	—	*	—	*	—	—

A

Notes for Table A-1 :

* These signals interconnect the 663 and the 664. Keep these lines point to point and as short as possible to maintain minimum flight time between the 663 and the 664.

— These timing specifications are not applicable to the signal.

1. The 663 samples CPU_DATA[0:63] on the second clock following their assertion by the CPU busmaster, so the effective input setup time is 5.7ns–T [CPU clock]. The 663 enables the CPU_DATA output drivers 1 CPU clock before setting the outputs to the correct state, so the net effective output enable delay is negative.

   The even numbered CPU_DATA lines follow the times shown and are switched and enabled by the CPU clock. The odd numbered CPU_DATA lines are switched asynchronously to improve the signal quality of the system. The timing of the asynchronously switched signals will always be better than that shown for the synchronously switched set.

2. The 663 enables the PCI_AD[0:31] output drivers one PCI clock before setting the outputs to the correct state, so the effective output enable delay is 13.9–T [PCI clock].

Where required for timing purposes, the 663 enables the PCI_AD lines one PCI clock before asserting them.

A.6 664 Controller Timing By Signal

Table A-2. 664 Controller Timing By Signal

Signal	Pin	I/O	Input (ns)		Output (ns)				
			Setup (min)	Hold (min)	Valid Delay (Max)	Enable Delay (Max)	Hold Time (min)	Hold (to 3S) (min)	Tristate Delay (Max)
AACK#[1]	109	I/O	2.9	0.07	8.5	9.7	2.3	2.9	8.9
AOS_RR_MMRS	69	O	—	—	*	—	*	—	—
ARTRY#[1]	110	I/O	3.9	1.3	8.7	10.3	2.3	3.2	10.0
C2P_WRL_OPEN	61	O	—	—	*	—	*	—	—
CAS[7:0]#	—	O	—	—	13.2	—	2.9	—	—
CPU_ADDR[0:31][1,2]	—	I/O	1.8	2.4	12.9	19.5	3.6	4.1	17.2
CPU_BUS_CLAIM#	132	I	3.0	1.3	—	—	—	—	—
CPU_DATA_OE#	197	O	—	—	*	—	*	—	—
CPU_GNT1#	134	O	—	—	9.0	—	2.5	—	—
CPU_GNT2#	135	O	—	—	9.0	—	2.5	—	—
CPU_PAR_ERR#	192	I	*	*	—	—	—	—	—
CPU_RDL_OPEN	50	O	—	—	*	—	*	—	—
CPU_REQ1#	127	I	3.1	0.6	—	—	—	—	—
CPU_REQ2#	128	I	3.0	0.6	—	—	—	—	—
CRS_C2PWXS	65	O	—	—	*	—	*	—	—
DBG#	140	O	—	—	13.2	—	3.3	—	—
DPE#	133	I	1.9	0	—	—	—	—	—
DUAL_CTRL_REF	205	O	—	—	*	—	*	—	—
ECC_LE_SEL	2	O	—	—	*	—	*	—	—
GBL#[1,2]	120	O	—	—	12.7	12.7	3.2	3.2	13.8
IGN_PCI_AD31	57	I	1.2	0.2	—	—	—	—	—
INT_CPU#	139	O	—	—	14.1	—	3.5	—	—
INT_REQ	55	I	4.6	0	—	—	—	—	—
MA[11:0]	—	O	—	—	13.6	—	3.0	—	—
MCP#	138	O	—	—	14.7	—	4.5	—	—
MEM_BE[3:0]	206	O	—	—	*	—	*	—	—
MEM_DATA_OE#	196	O	—	—	*	—	*	—	—
MEM_ERR#	194	I	*	*	—	—	—	—	—
MEM_RD_SMPL	49	O	—	—	*	—	*	—	—
MEM_WRL_OPEN	51	O	—	—	*	—	*	—	—
MWS_P2MRXS	66	O	—	—	*	—	*	—	—
NMI_REQ	56	I	1.5	0.2	—	—	—	—	—
PCI_AD[31:0]		I/O	7.0	0	11.0	11.0	3.4	2.9	8.8
PCI_AD_OE#	195	O	—	—	*	—	*	—	—
PCI_C/BE[3:0]#	—	I/O	7.0	0	11.0	11.0	3.3	2.7	8.3
PCI_DEVSEL#	204	I/O	7.0	0	11.0	11.0	3.3	2.8	8.4
PCI_EXT_SEL	67	O	—	—	*	—	*	—	—

A

Table A-2. 664 Controller Timing By Signal (Continued)

Signal	Pin	I/O	Input (ns)		Output (ns)				
			Setup (min)	Hold (min)	Valid Delay (Max)	Enable Delay (Max)	Hold Time (min)	Hold (to 3S) (min)	Tristate Delay (Max)
PCI_FRAME#	200	I/O	7.0	0	11.0	11.0	3.3	2.8	8.5
PCI_GNT#	54	I	7.0	0	—	—	—	—	—
PCI_IRDY#	201	I/O	7.0	0	11.0	11.0	3.3	2.8	8.6
PCI_LOCK#	53	I	7.0	0	—	—	—	—	—
PCI_OL_OPEN	64	O	—	—	*	—	*	—	—
PCI_OUT_SEL	68	O	—	—	*	—	*	—	—
PCI_PAR	7	I/O	7.0	0	11.0	11.0	3.3	2.8	8.6
PCI_PERR#	10	I/O	7.0	0	11.0	11.0	3.3	2.7	8.1
PCI_REQ#	58	O	—	—	11.9	—	3.4	—	—
PCI_SERR#	71	O	—	—	11.0	11.0	2.0	2.9	8.8
PCI_STOP#	203	I/O	7.0	0	11.0	11.0	3.3	2.0	8.4
PCI_TRDY#	202	I/O	7.0	0	11.0	11.0	3.3	2.8	8.4
RAS[7:0]#	—	O	—	—	13.2	—	2.9	—	—
RESET#	156	I	2.0	0	—	—	—	—	—
ROM_LOAD	70	O	—	—	*	—	*	—	—
ROM_OE#	47	O	—	—	15.7	—	3.9	—	—
ROM_WE#	60	O	—	—	16.2	—	3.9	—	—
SBE#	193	I	*	*	—	—	—	—	—
SHD#[1]	141	O	—	—	9.5	9.6	—	2.2	10.5
SRAM_ADS#/ADDR0	124	O	—	—	10.8	—	3.1	—	—
SRAM_ALE	119	O	—	—	13.6	—	3.0	—	—
SRAM_CNT_EN#/ ADDR1	125	O	—	—	10.8	—	3.1	—	—
SRAM_OE#[3]	117	O	—	—	16.0	—	3.5	—	—
SRAM_WE#	118	O	—	—	11.2	—	3.3	—	—
STOP_CLK_EN#	151	I	1.9	0.5	—	—	—	—	—
TA#[1]	111	I/O	4.7	0.4	8.6	16.7	2.2	3.3	15.7
TAG_CLR#	116	O	—	—	14.2	—	4.4	—	—
TAG_MATCH	142	I	2.3	0.7	—	—	—	—	—
TAG_VALID	115	O	—	—	12.7	—	3.5	—	—
TAG_WE#	114	O	—	—	12.9	—	3.6	—	—
TBST#[1,2]	144	I	2.0	0	—	13.7	—	2.6	11.8
TEA#[1]	137	O	2.4	1.4	0.8	16.7	2.2	3.3	15.7
TS#[1]	143	I/O	3.1	0	8.6	16.4	2.3	4.8	15.2
TSIZE[0:2][1,2]	—	I/O	2.4	0.3	13.8	19.2	2.5	4.0	16.9
TT[0:4][1,2]	—	I/O	3.9	0.15	14.5	19.2	4.0	4.0	16.9

A

Table A-2. 664 Controller Timing By Signal (Continued)

Signal	Pin	I/O	Input (ns)		Output (ns)				
			Setup (min)	Hold (min)	Valid Delay (Max)	Enable Delay (Max)	Hold Time (min)	Hold (to 3S) (min)	Tristate Delay (Max)
WE[1:0]#	—	O	—	—	11.7	—	2.8	—	—
XATS#	129	I	2.2	0	11.8	—	2.8	—	—

Notes:

* These signals interconnect the 663 and the 664. Keep these lines point to point and as short as possible to maintain minimum flight time between the 663 and the 664.

— These timing specifications are not applicable to the signal.

1. The 660 enables these signals at least one CPU clock before setting the outputs to the correct state, so the net effective output enable delay is T(enable delay) – T(CPU clock).
2. The 660 drives these signals to the correct state one CPU clock before asserting TS#; thus, the net effective delay is T(enable delay) – T(CPU clock).
3. The 660 drives this signal to the correct state at least one CPU clock before it is required.

Typically asynchronous signals, such as RESET#, INT_REQ, and NMI_REQ are metastable-hardened. If their assertion timing violates the input setup or hold times for a given clock cycle, these signals will be recognized on a subsequent clock.

Transitions on the RAS[7:0]# and CAS[7:0]# lines are timed from the rising edge of the CPU clock during CPU and PCI accesses to the DRAM, and are timed from the rising edge of PCI_CLK during DRAM refresh cycles. The output timing in each case is the same. For example, RAS# output valid delay is a maximum of 12.8ns from either the CPU or PCI clock, whichever one provoked the transition.

A

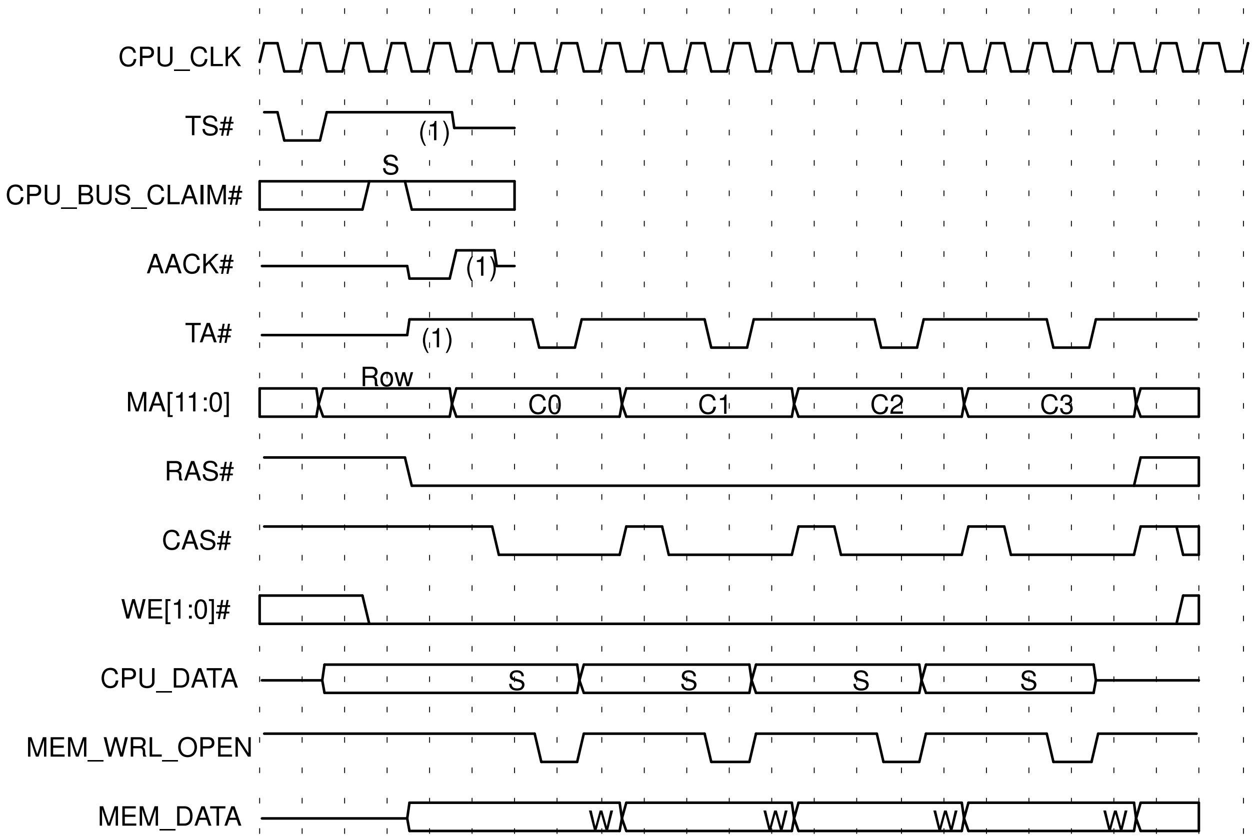
A.7 Detailed Timing Diagrams

The following timing diagrams in Figure A-6 through Figure A-22 illustrate some fundamental operations of the 660 Bridge.

In CPU to memory read burst-mode timing diagrams where asynchronous SRAMs are used with the internal L2 controller, an additional clock cycle must be added to the fourth beat of any CPU to memory read burst that causes a cache miss. For example after a read:

> A pipelined page hit, cache miss, burst read with EDO DRAM requires –3–3–3–3 CPU clocks when the L2 uses burst SRAMs and –3–3–3–4 CPU clocks when the L2 uses asynchronous SRAMs. The extra beat is caused by delaying the final TA# by one CPU_CLK to allow the asynchronous SRAM sufficient data hold time for the fourth beat.

A.7.1 CPU to Memory Write (Page DRAM) From Bus Idle



(1) These signals are tristated as shown only while the internal cache is disabled. While the internal cache is enabled, these signals are always driven except in response to a CPU_BUS_CLAIM# from a CPU bus agent.

Figure A-6. CPU to Memory Write (Page DRAM) From Bus Idle

A.7.2 CPU to Memory Write (Page DRAM) Followed by Write Hit

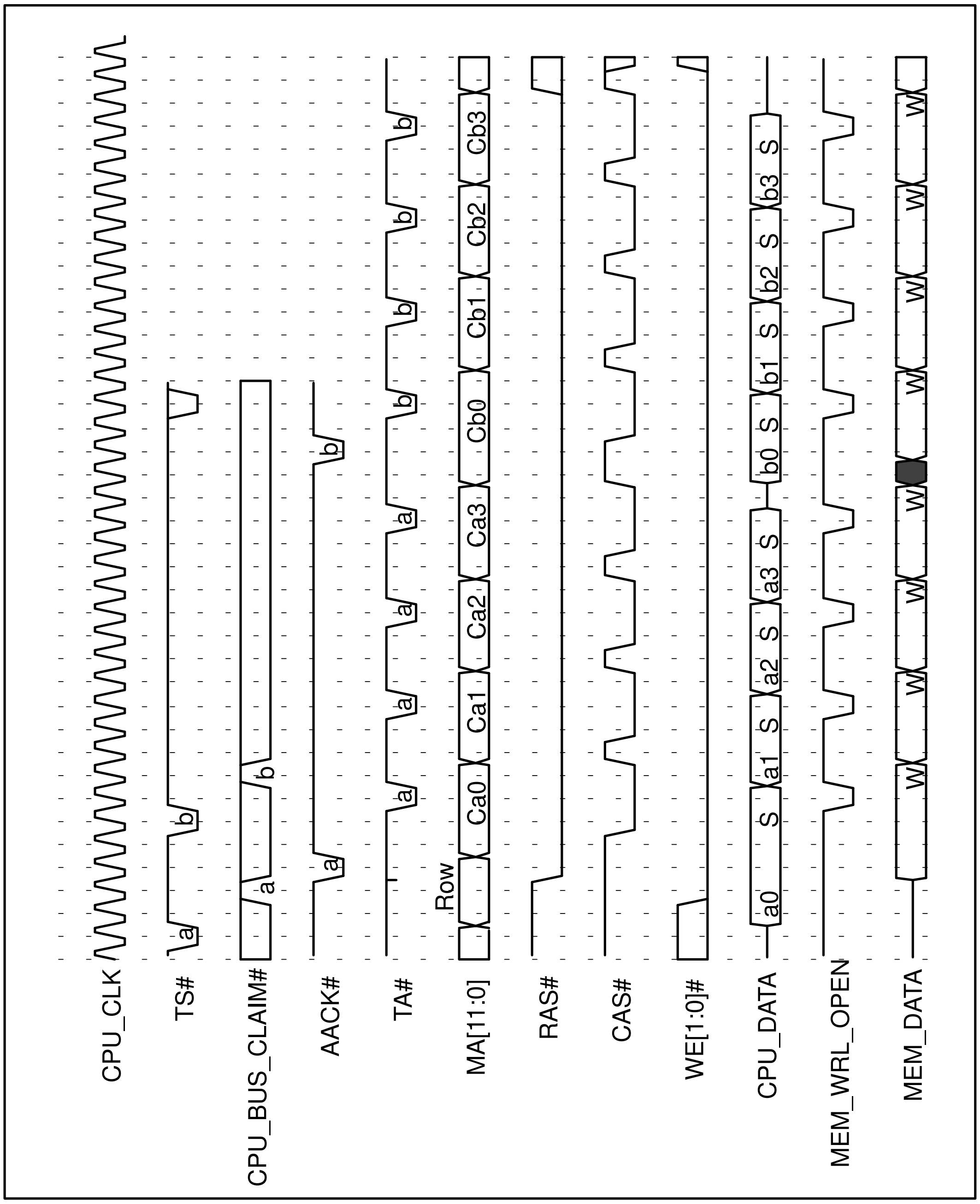


Figure A-7. CPU to Memory Write (Page DRAM) Followed by Write Hit

A.7.3 CPU to Memory Write (Page DRAM) Followed by Write Page Miss and Bank Miss

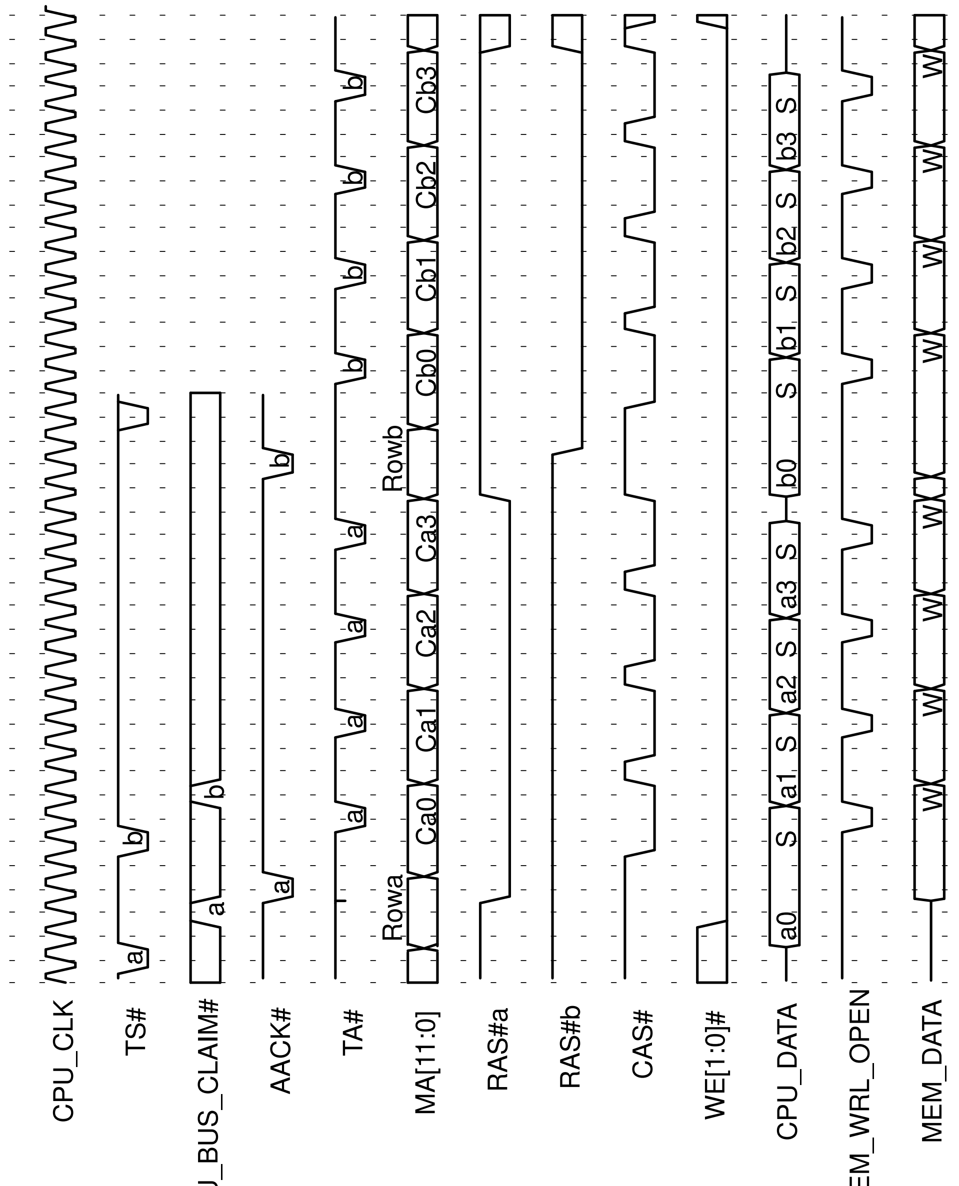


Figure A-8. CPU to Memory Write (Page DRAM) Followed by Write Page Miss and Bank Miss

A.7.4 CPU to Memory Write (Page DRAM) Followed by Read Hit

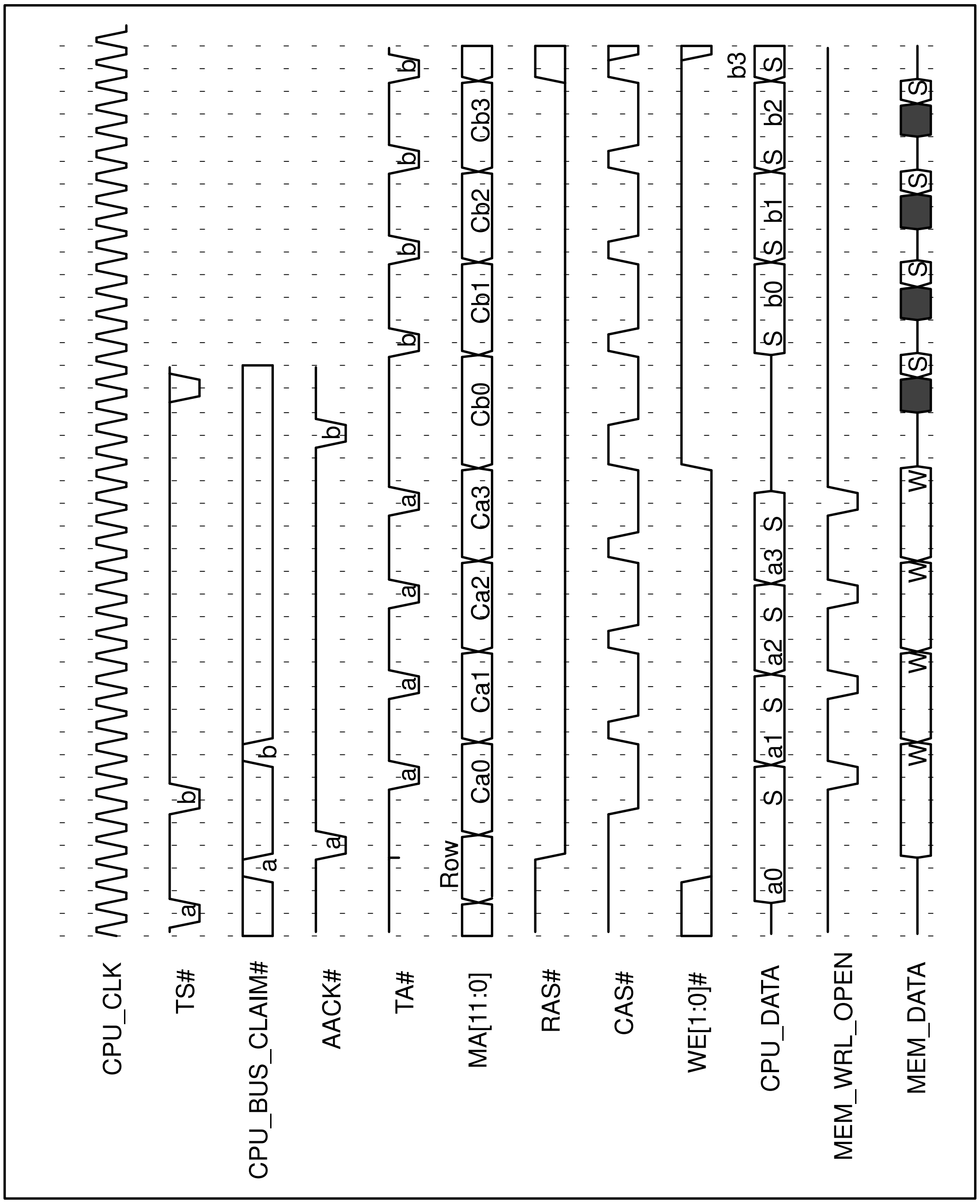


Figure A-9. CPU to Memory Write (Page DRAM) Followed by Read Hit

A.7.5 CPU to Memory Write (Page DRAM) Read/Modify/Write From Bus Idle

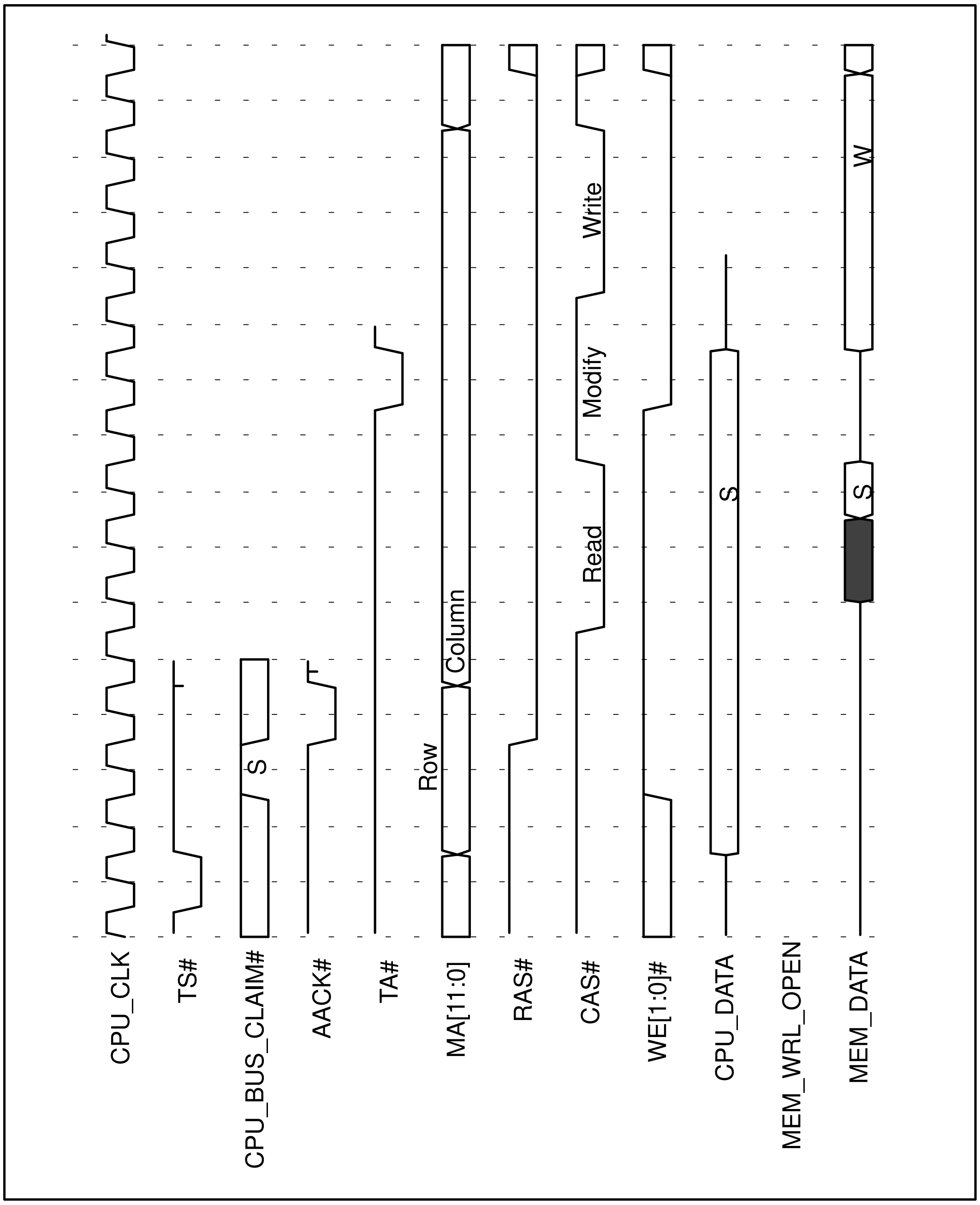


Figure A-10. CPU to Memory Write (Pg. DRAM) Read/Modify/Write From Bus Idle

A.7.6 CPU to Memory Read (Page DRAM) From Bus Idle

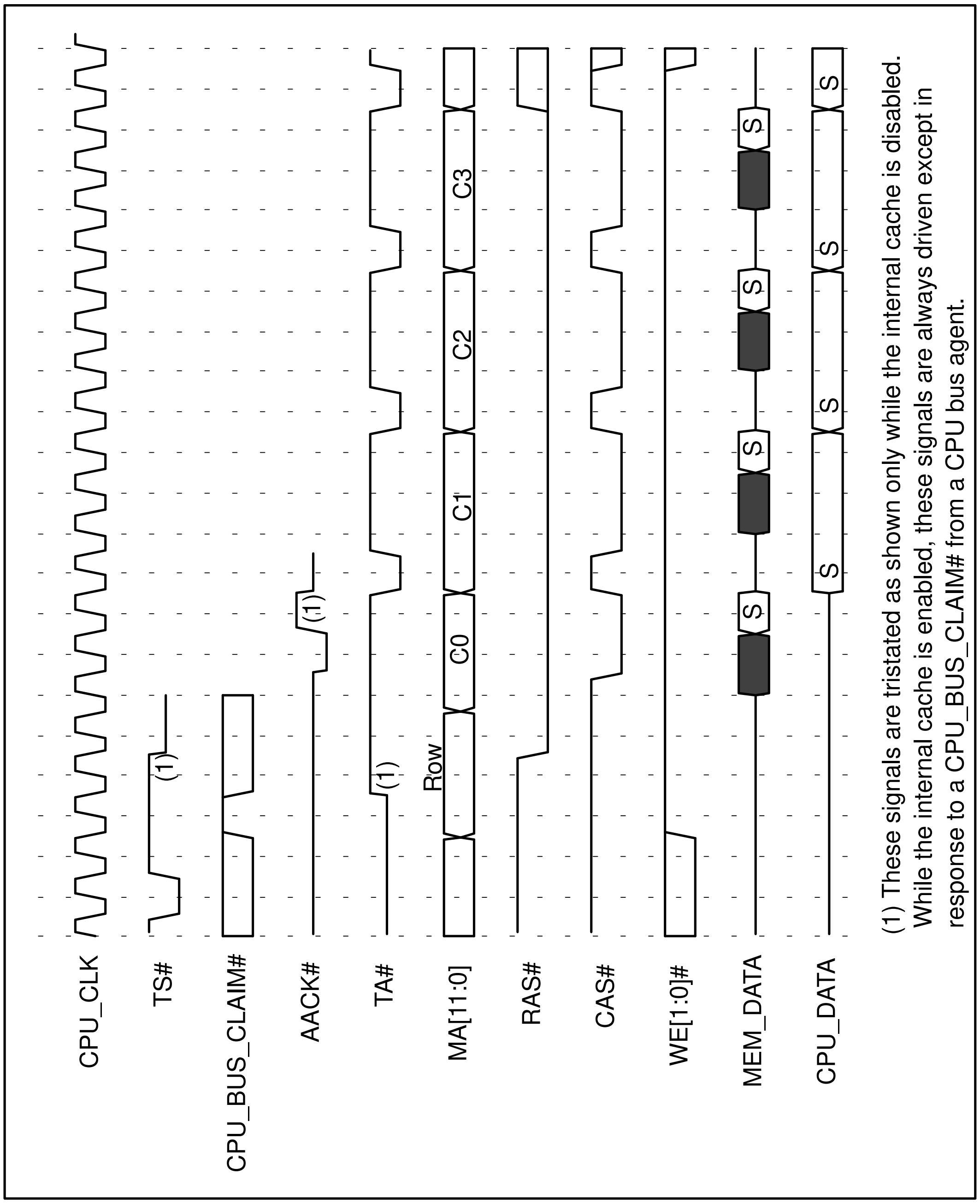


Figure A-11. CPU to Memory Read (Page DRAM) From Bus Idle

A.7.7 CPU to Memory Read (Page DRAM) Followed by Read Hit

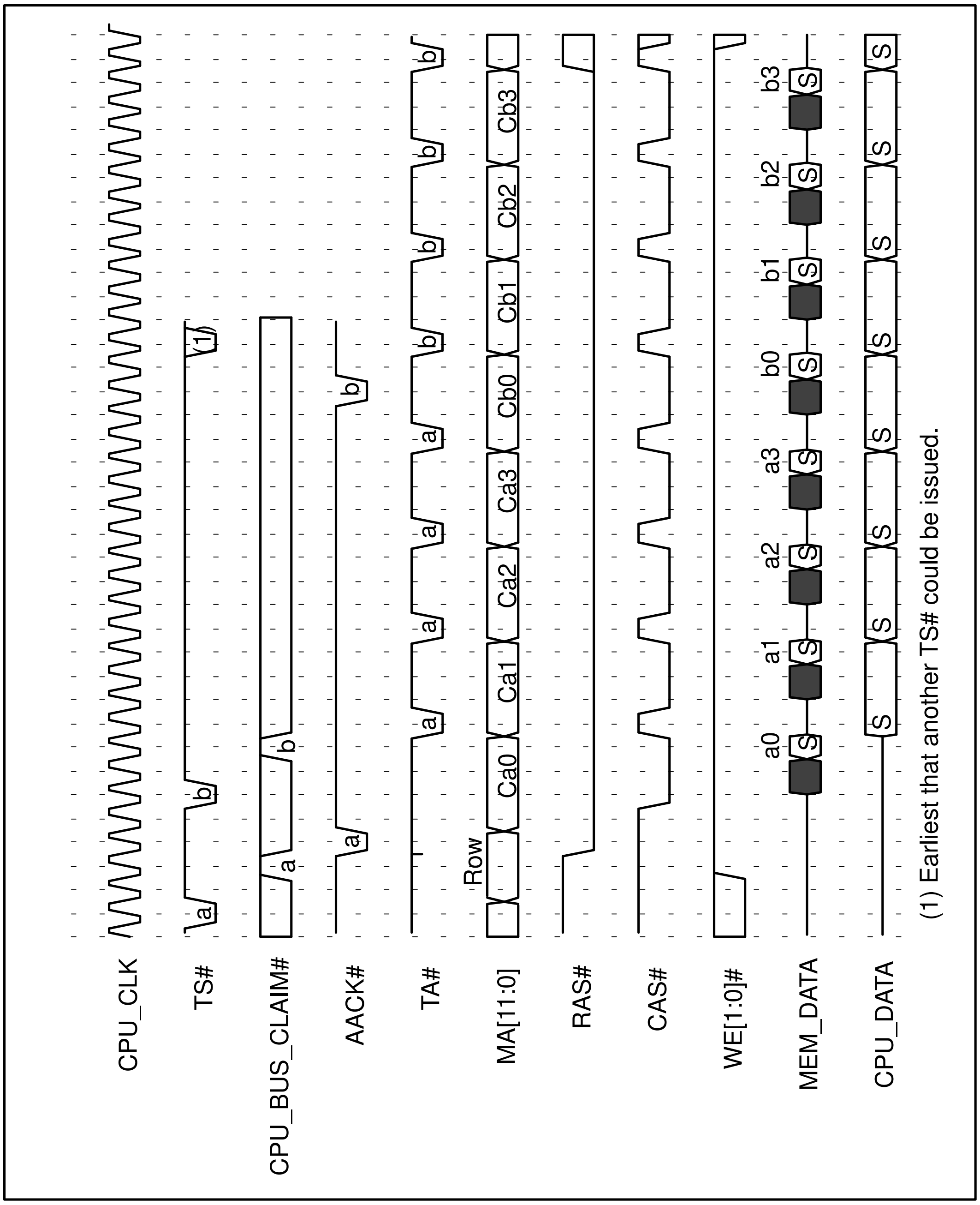


Figure A-12. CPU to Memory Read (Page DRAM) Followed by Read Hit

A.7.8 CPU to Memory Read (Page DRAM) Followed by Read Miss and Bank Miss

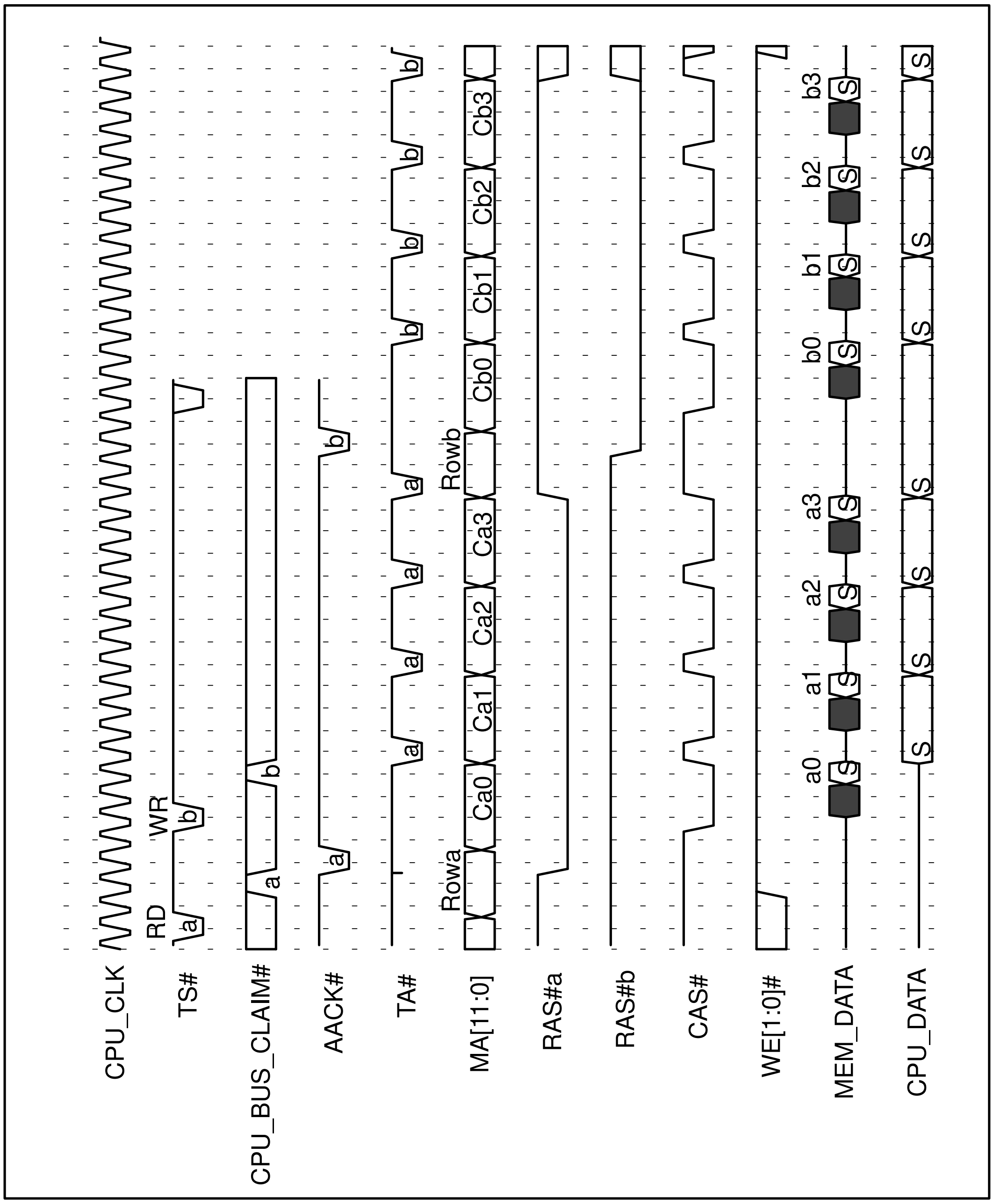


Figure A-13. CPU to Memory Read (Page DRAM) Followed by Read Miss and Bank Miss

A.7.9 CPU to Memory Read (Page DRAM) Followed by Write Hit

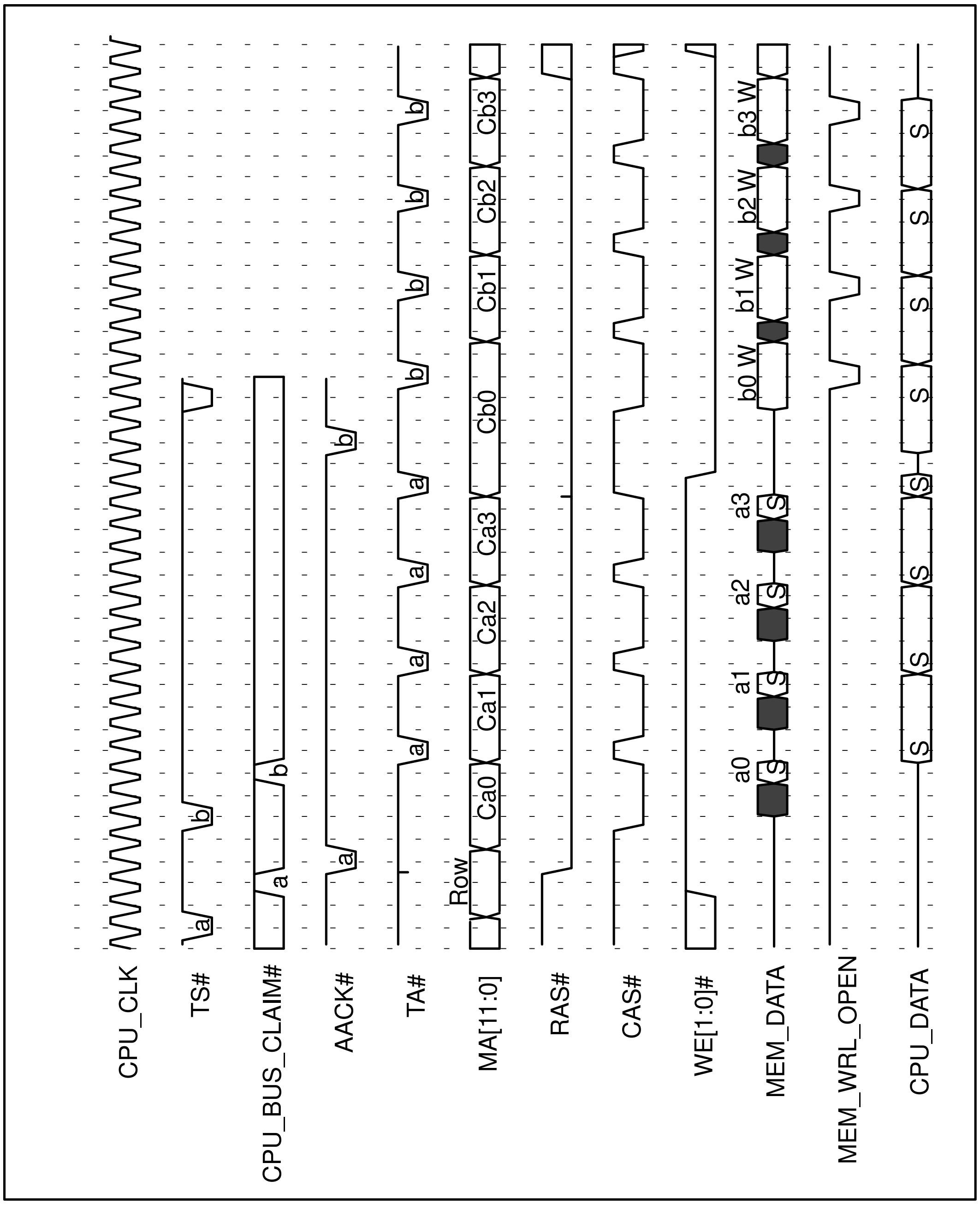
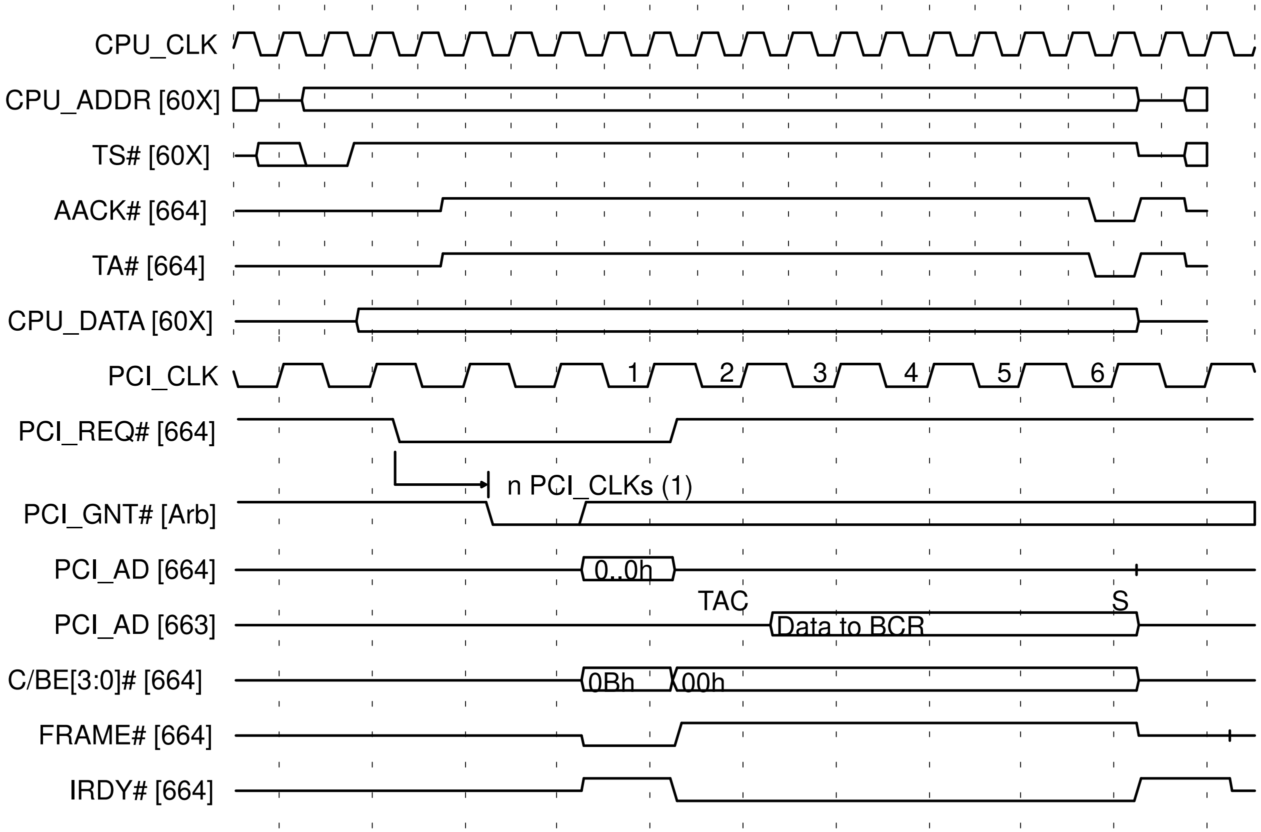


Figure A-14. CPU to Memory Read (Page DRAM) Followed by Write Hit

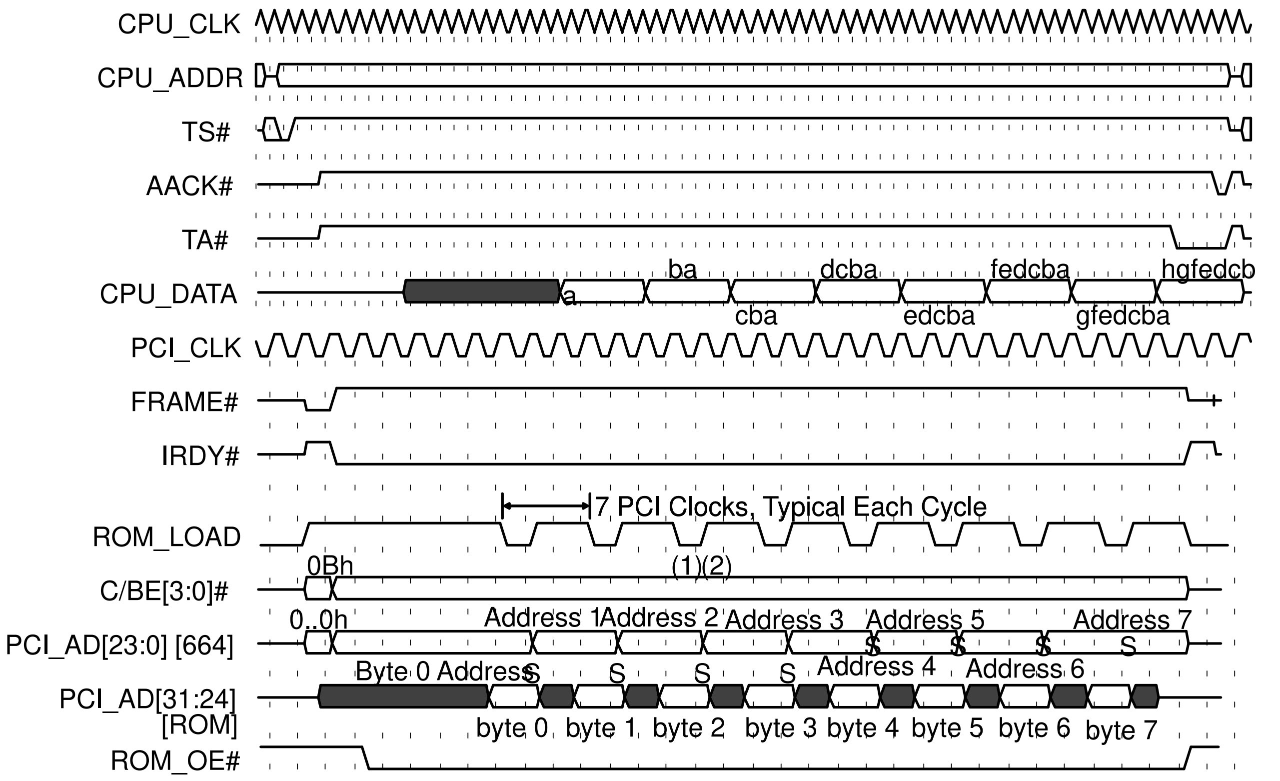
A.7.10 CPU to Bridge Write of Bridge Control Register



Note that: DEVSEL#, IDSEL#, TRDY#, and STOP# are not asserted during the transaction.
(1) This delay controlled by the system arbiter.

Figure A-15. CPU to Bridge Write of Bridge Control Register

A.7.11 CPU to Bridge Read of ROM



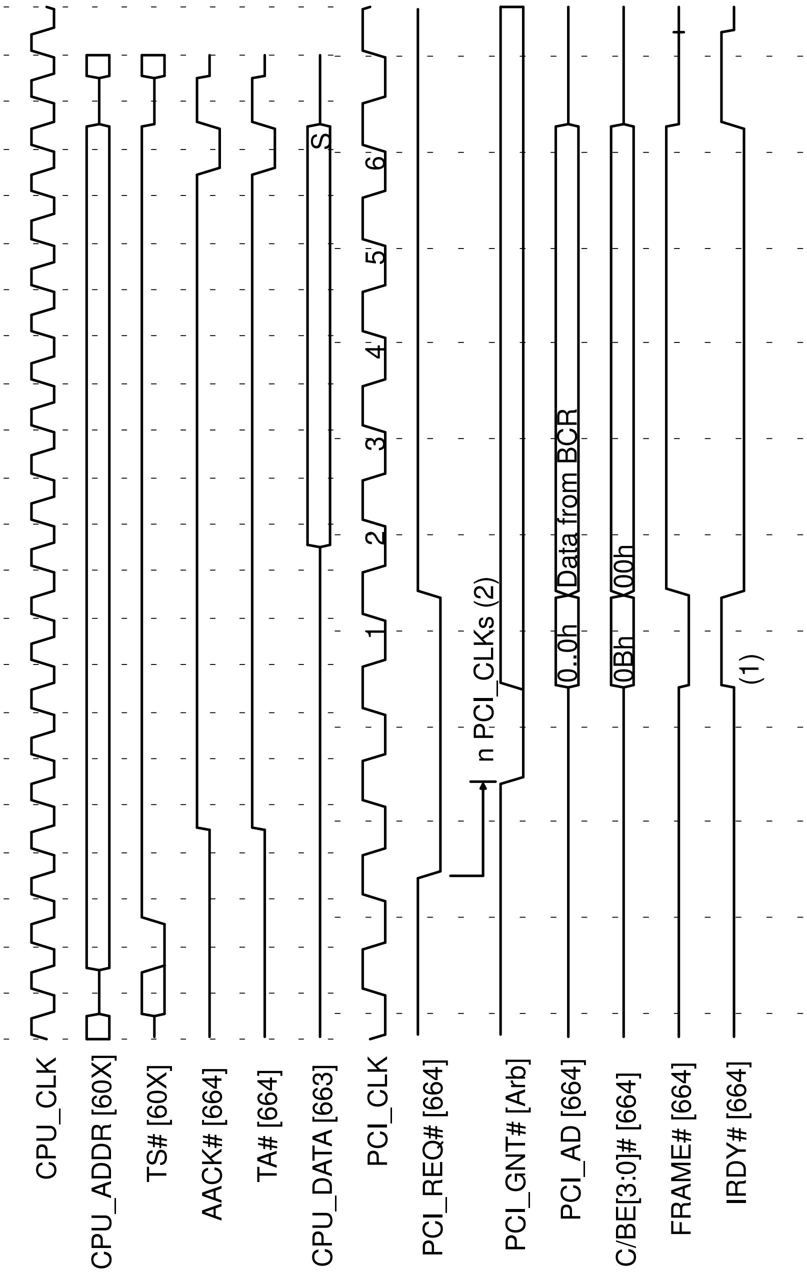
Note that: DEVSEL#, IDSEL#, TRDY#, STOP#, and ROM_WE# are not asserted during the transaction.

1 Diagram assumes that the PCI bus is parked on the CPU.

2 (1) shift (2) load

Figure A-16. CPU to Bridge Read of ROM

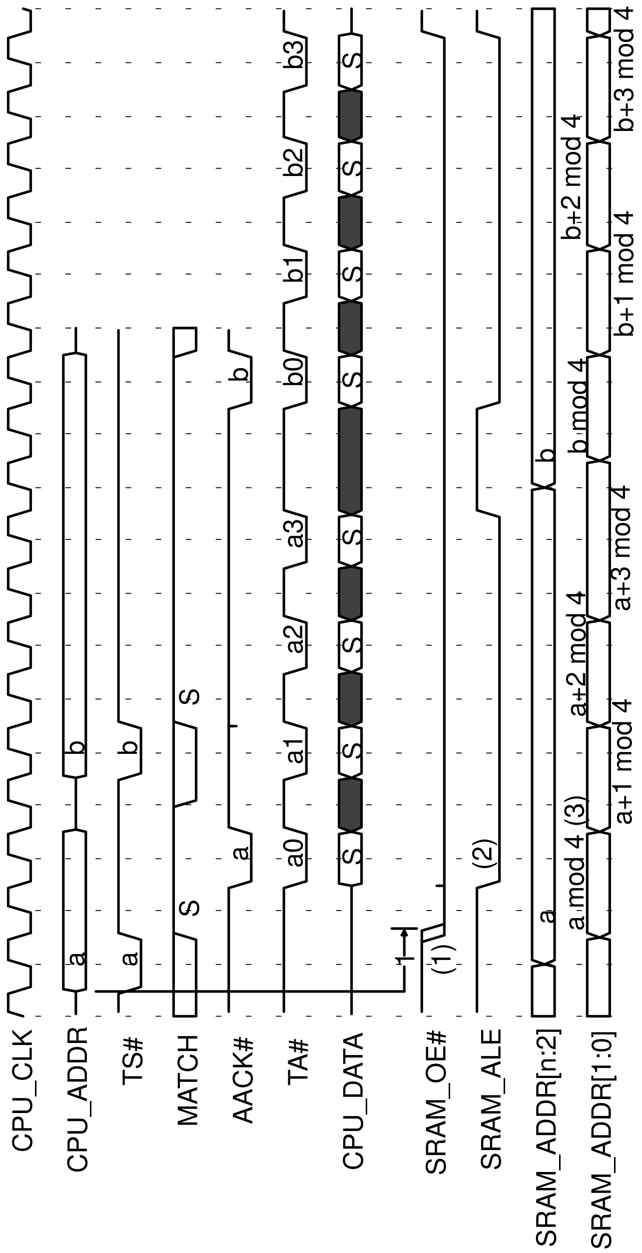
A.7.12 CPU to Bridge Read of Bridge Control Register



Note that: DEVSEL#, IDSEL#, TRDY#, and STOP# are not asserted during the transaction.
(1) The Bridge takes control of the PCI bus at this point.
(2) This delay controlled by the system arbiter.

Figure A-17. CPU to Bridge Read of Bridge Control Register

A.7.13 CPU to Memory Read, L2 Cache w/Async SRAMS, Hit



(1) On cacheable CPU to memory reads, SRAM_OE# is speculatively asserted.

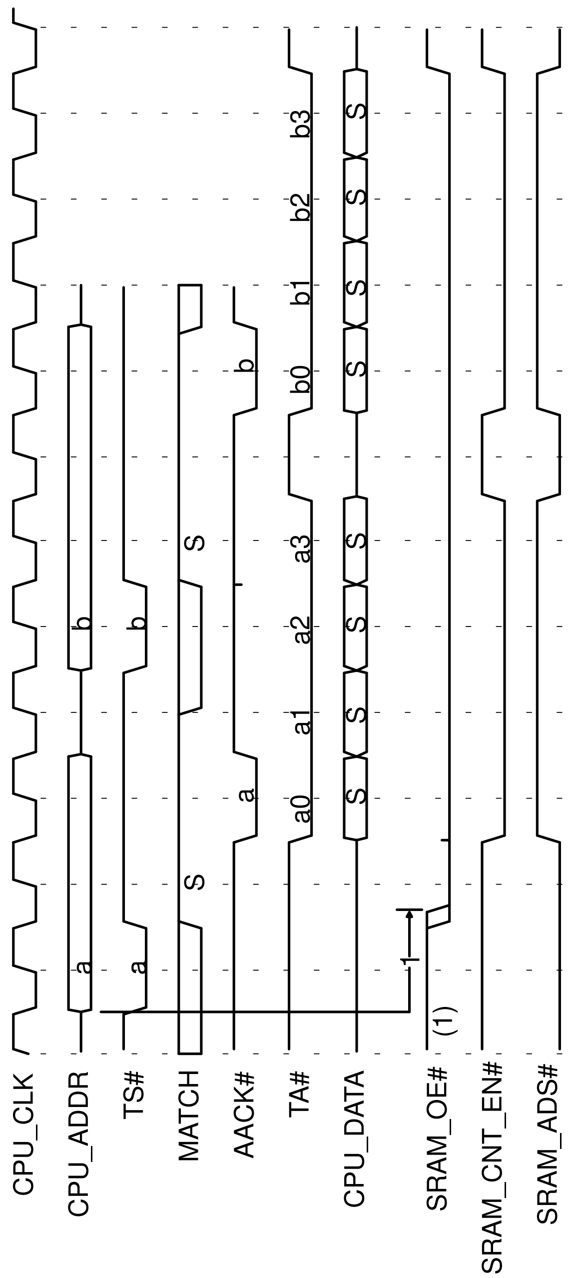
If MATCH is sampled deasserted, SRAM_OE# is deasserted at the tick mark.

(2) SRAM_ADDR[n:2] are held by a transparent external latch while ALE is low.

(3) SRAM_A[1:0] is incremented in a circular manner (00, 01, 10, 11, 00,...) and may begin in any sta

Figure A-18. CPU to Memory Read, L2 Cache w/Async SRAMS, Hit

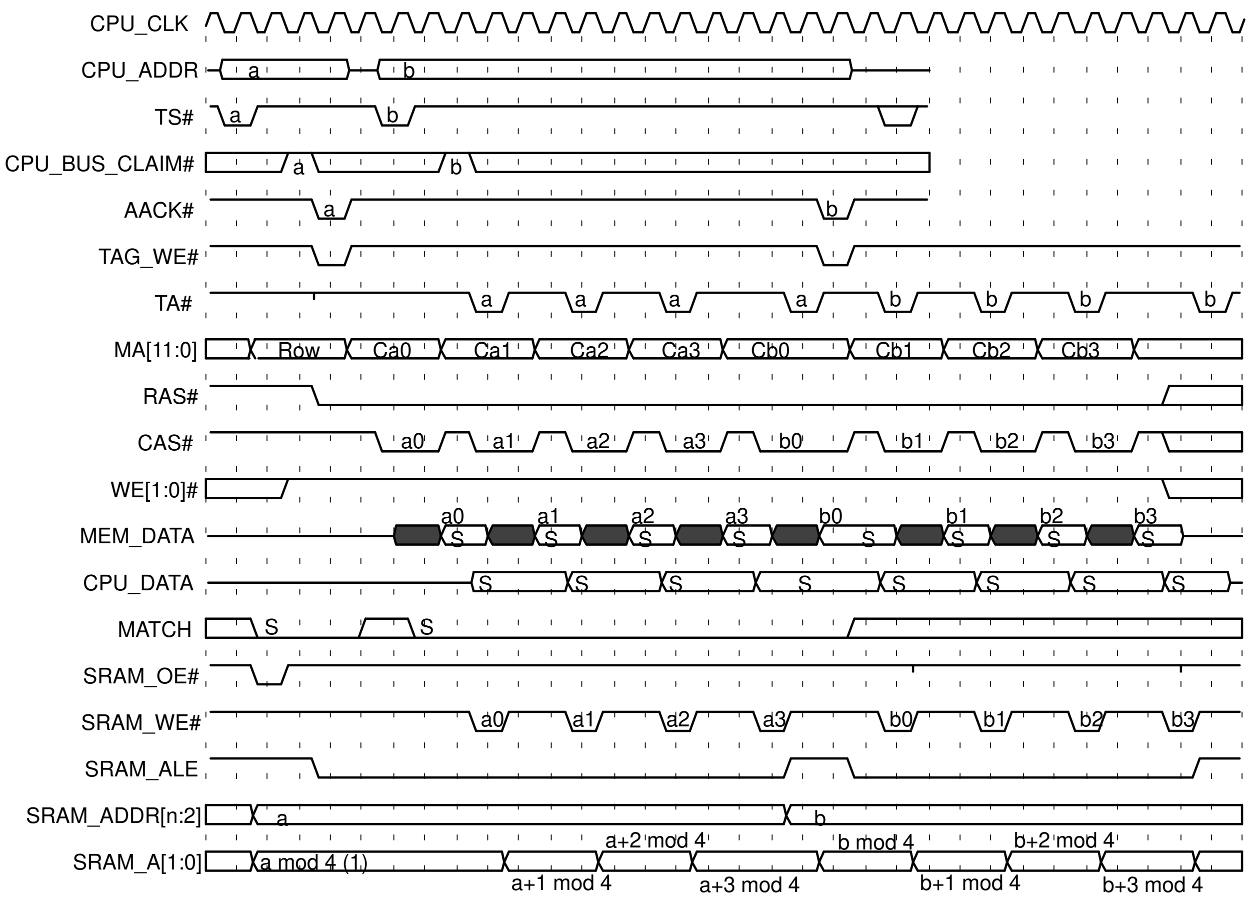
A.7.14 CPU to Memory Read, L2 Cache w/Burst SRAMs, Hit



(1) On cacheable CPU to memory reads, SRAM_OE# is speculatively asserted.
If MATCH is sampled deasserted, SRAM_OE# is deasserted at the tick mark.

Figure A-19. CPU to Memory Read, L2 Cache w/Burst SRAMs, Hit

A.7.15 CPU to Memory Read (EDO DRAM) Cache Miss Followed by Read Hit Cache Miss w/Async SRAMs



(1) SRAM_A[1:0] is incremented in a circular manner (00, 01, 10, 11, 00,...) and may begin in any state.

Figure A-20. CPU to Memory Read (EDO DRAM) Cache Miss Followed by Read Hit Cache Miss w/Async SRAMs

A.7.16 CPU to Memory Read (EDO DRAM) Cache Miss Followed by Read Hit Cache Miss w/Burst SRAMs

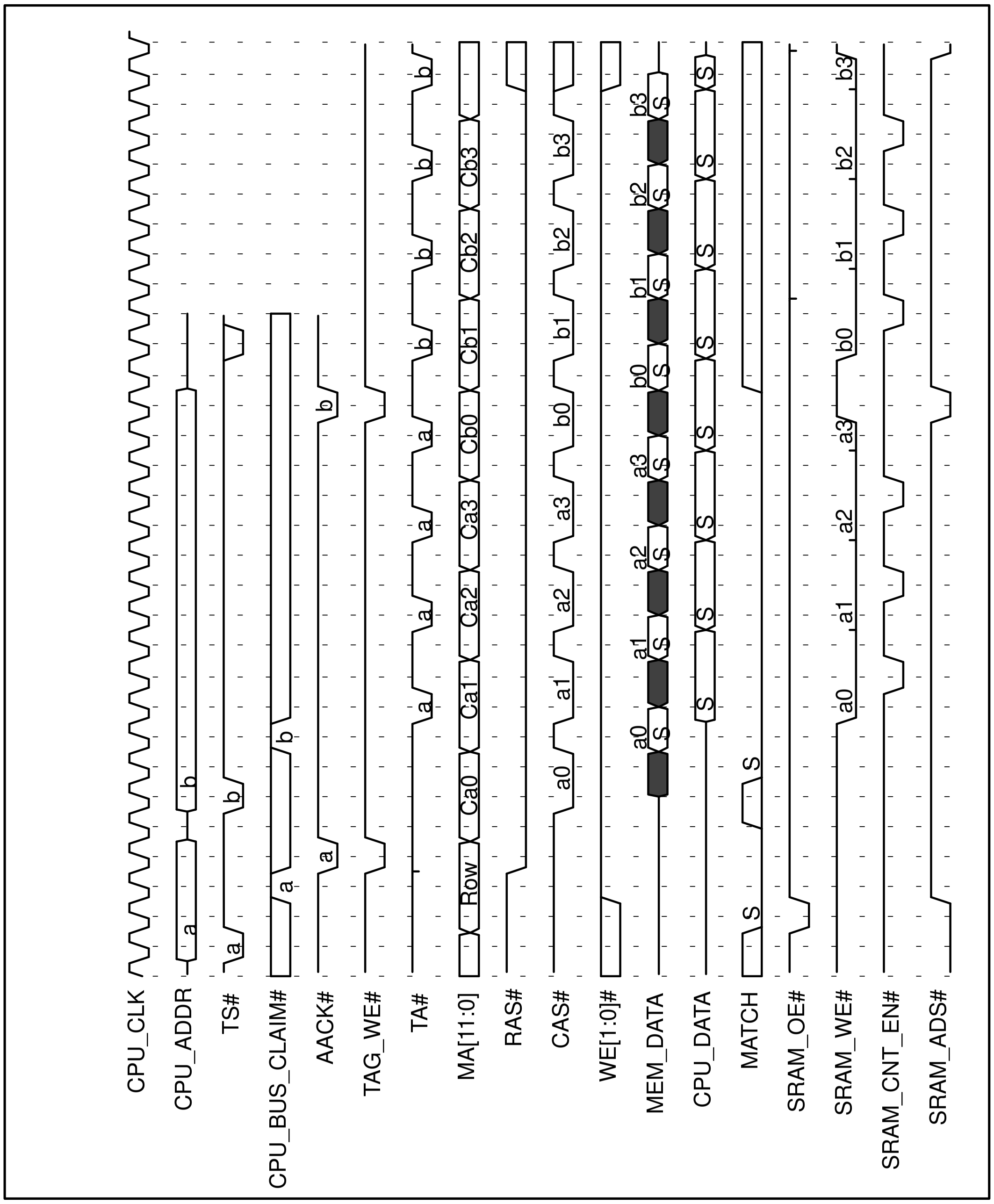


Figure A-21. CPU to Memory Read (EDO DRAM) Cache Miss Followed by Read Hit Cache Miss w/Burst SRAMs

A.7.17 CPU to Memory Read (EDO DRAM) Followed by Read Hit

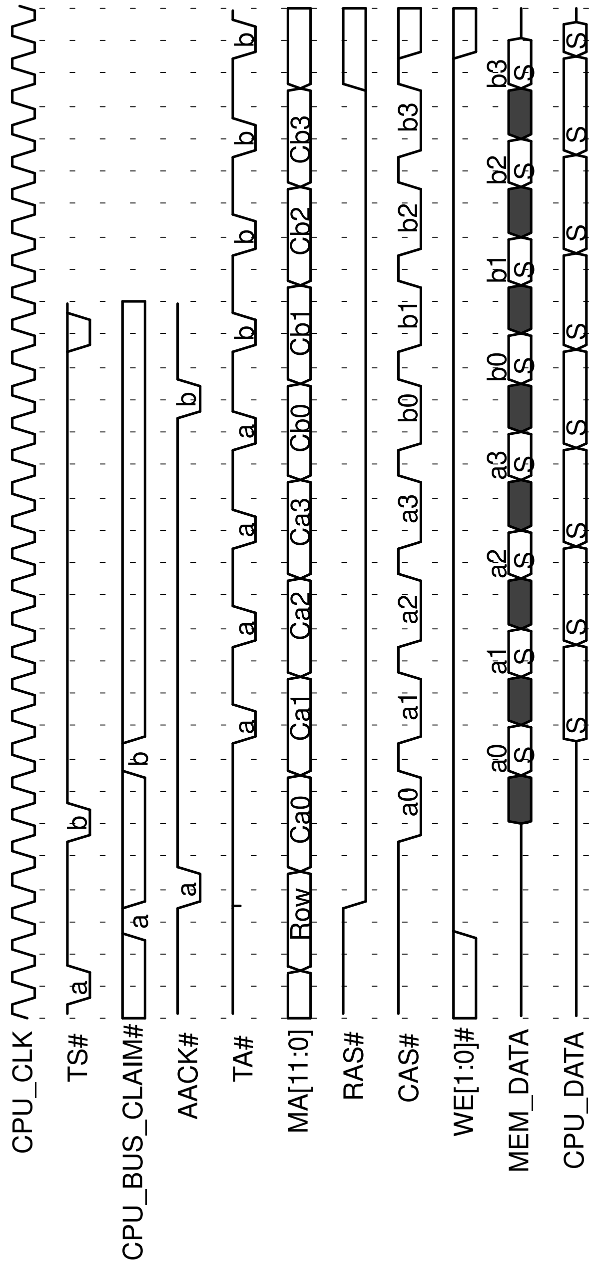


Figure A-22. CPU to Memory Read (EDO DRAM) Followed by Read Hit

Appendix B
Electrical and Mechanical

Unless otherwise noted, all specifications in this section apply to both the 663 and to the 664.

B.1 Absolute Maximum Ratings

Stresses in excess of those listed in Table B-1 may damage and/or decrease the reliability of the 660 Bridge. Additionally, stressing the 660 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 B-1. Absolute Maximum Ratings, 660 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

B.2 Recommended Operating Conditions

Table B-2 lists the conditions under which the 660 Bridge is intended to operate.

Table B-2. Recommended Operating Conditions, 660 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	(1)
Top	Junction Temperature, Operating	10	85	deg C	

Notes ForTable B-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.

B.3 Power Dissipation and Thermal Characteristics

B.3.1 Power Dissipation

Table B-3 shows typical power dissipation numbers for the 663 and the 664. These values were measured during heavy bus traffic periods, using devices with typical process variables.

Table B-3. 660 Power Dissipation

Parameter	663	664	Unit
Typical Power Dissipation, Vdd = 3.3v, Heavy Bus Traffic	0.5	1.0	W
Typical Power Dissipation, Vdd = 3.6v, Heavy Bus Traffic	0.6	1.3	W

B.3.2 Thermal Characteristics

B

B.3.2.1 Typical Thermal Resistance from Junction to Ambient

Table B-4 shows the typical thermal resistances associated with the 663 and the 664. 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.

Table B-4. Typical Thermal Resistance, Junction to Ambient, No Heat Sink

Airflow	Θj–a, 663	Θj–a, 664	Units
Convection	43	52	deg C/W
.25 M/s (50fpm)	37	45	deg C/W
.5 M/s (100fpm)	34	42	deg C/W
1 M/s (200fpm)	31	38	deg C/W

B.3.2.2 Typical Thermal Resistance from Junction to Case

Different cooling paths predominate at different heat flows. Typically:

TR_{j-c} ≈ 10° C/W with a good heat sink, and
≈ 2° – 5° C/W without a heat sink

B.4 Common Characteristics

The specifications shown in Table B-5 are common to both the 663 and the 664, while within the recommended operating conditions envelope.

Table B-5. Common Characteristics

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 B-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.

B.5 Package and Pin Electrical Characteristics Model

The electrical model of the effects of package and pin parasitic effects on the 660 Bridge is shown in Figure B-1. The corresponding ranges of values shown in Table B-6 are nominal only, are not guaranteed, and vary somewhat 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 Figure B-1 (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.

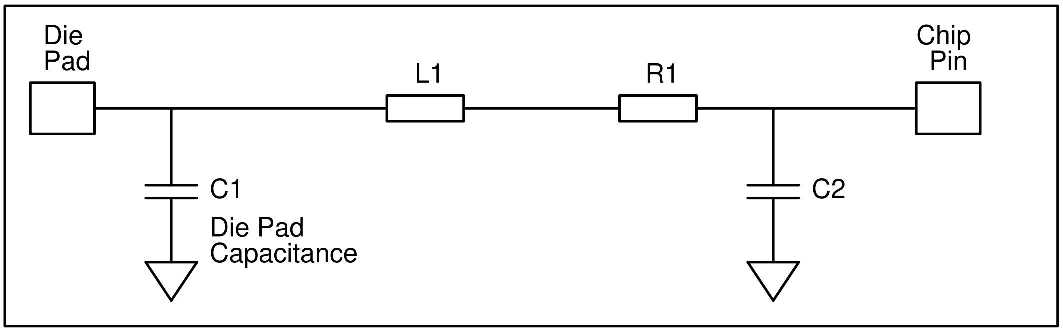


Figure B-1. 653 Package/Pin Electrical Model

Table B-6. Electrical Model Range of Values

Symbol	Condition	663	664	Unit
Inductance (L1)	Shortest lead	12	11	nH
	Longest lead	17	14	nH
Resistance (R1)	Shortest lead	.17	.13	Ω
	Longest lead	.21	.17	Ω
Capacitance (C2)	Shortest lead	1.4	1.2	pf
	Longest lead	2.4	1.9	pf

B

B.6 663 DC Characteristics By Signal

Table B-7. 663 DC Characteristics By Signal (See Note 1)

Signal	Pin	I/O	Book, Pad (2)	Type (3)	P/L (2)	I_{OL} (mA)	I_{OH} (mA)	SPICE Card (4)	DPC (5) Min	DPC (5) Max
AOS_RR_MMRS	166	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
C2P_WRL_OPEN	154	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
CPU_CLK	157	I	CBB4, B0	TTL	A	—	—	cbb4 a b0	.8	1.0
CPU_DATA[00:63]	—	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
CPU_DATA_OE#	146	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
CPU_DPAR[0:7]	App B	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
CPU_PAR_ERR#	174	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
CPU_RDL_OPEN	148	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
CRS_C2PWXS	151	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
DUAL_CTRL_REF	170	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
ECC_LE_SEL	149	I	CBJE, BO	TTL	F	—	—	cbjd a b0	.8	1.0
MEM_BE[0:1]	App B	I	CBSX, B0	TTL	A	—	—	cbsx a b0	.8	1.0
MEM_BE[2:3]	App B	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
MEM_CHECK[0:7]	App B	I/O	CBNU, 10	TTL	B	12	8	cbns b 10	3.7	4.3
MEM_DATA[63:0]	App B	I/O	CBNU, 10	TTL	B	12	8	cbns b 10	3.7	4.3
MEM_DATA_OE#	145	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
MEM_ERR#	171	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
MEM_RD_SMPL	147	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
MEM_WRL_OPEN	150	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
MIO_TEST	156	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
MWS_P2MRXS	152	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
PCI_AD[31:0]	App B	I/O	CBUM, 10	PCI	A	6	6	cbuk a 10	3.1	3.7
PCI_AD_OE#	144	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
PCI_EXT_SEL	153	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
PCI_IRDY#	167	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
PCI_OL_OPEN	165	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
PCI_OUT_SEL	169	I	CBJE, BO	TTL	F	—	—	cbjd a b0	.8	1.0
PCI_TRDY#	168	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
ROM_LOAD	160	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
SBE#	175	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
TEST#	155	I	CBSW, B3	TTL	A	—	—	cbsw a b3	.8	1.0

Notes:

1) Values apply over recommended operating conditions.

B

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 B.4, Common Characteristics.

4) Use this SPICE card to model this signal. The 660 SPICE model package is available from IBM under Non-Disclosure Agreement. Contact your IBM technical representative for more information.

5) 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 B.5.

B.7 664 DC Characteristics By Signal

Table B-8. 664 DC Characteristics By Signal (See Note 1)

Signal	Pin	I/O	Book, Pad (2)	Type (3)	P/L (2)	I_{OL} (mA)	I_{OH} (mA)	SPICE Card (4)	DPC (5)	
									Min	Max
AACK#	109	I/O	CBUM, 10	PCI	C	6	6	cbuk c 10	3.1	3.7
AOS_RR_MMRS	69	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
ARTRY#	110	I/O	CBUM, 10	PCI	C	6	6	cbuk c 10	3.1	3.7
C2P_WRL_OPEN	61	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
CAS[7:0]#	—	O	CBNQ, 13	TTL	C	12	8	cbno c 13	3.5	4.2
CPU_ADDR[0:31]	—	I/O	CBNZ, 11	TTL	B	14	12	cbnw b 11	2.0	2.2
CPU_BUS_CLAIM#	132	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.5	3.7
CPU_CLK	121	I	CBB4, B0	TTL	A	—	—	cbb4 a b0	.8	1.0
CPU_DATA_OE#	197	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
CPU_GNT1#	134	O	CBUI, 13	PCI	C	6	6	cbug c 10	2.8	3.5
CPU_GNT2#	135	O	CBUI, 13	PCI	C	6	6	cbug c 10	2.8	3.5
CPU_PAR_ERR#	192	I	CBJE, BO	TTL	F	—	—	cbjd a b0	.8	1.0
CPU_RDL_OPEN	50	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
CPU_REQ1#	127	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
CPU_REQ2#	128	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
CRS_C2PWXS	65	O	CBNZ, 11	TTL	C	14	12	cbnw c 11	2.0	2.2
DBG#	140	O	CBNQ, 15	TTL	C	4	4	cbno c 15	2.9	3.5
DPE#	133	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
DUAL_CTRL_REF	205	O	CBNX, 16	TTL	B	14	12	cbnw b 11	2.0	2.2
ECC_LE_SEL	2	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
GBL#	120	O	CBNX, 11	TTL	B	14	12	cbnw b 11	2.0	2.2
IGN_PCI_AD31	57	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
INT_CPU#	139	O	CBNQ, 15	TTL	C	4	4	cbno c 15	2.9	3.5
INT_REQ	55	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
MA[11:0]	—	O	CBNQ, 13	TTL	C	12	8	cbno c 13	3.5	4.2
MCP#	138	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
MEM_BE[3:0]	206	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
MEM_DATA_OE#	196	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2

Table B-8. 664 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)	SPICE Card (4)	DPC (5) Min	DPC (5) Max
MEM_ERR#	194	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
MEM_RD_SMPL	49	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
MEM_WRL_OPEN	51	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
MIO_TEST	154	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
MWS_P2MRXS	66	O	CBNZ, 11	TTL	C	14	12	cbnw c 11	2.0	2.2
NMI_REQ	56	I	CBSX, B0	TTL	A	—	—	cbsx a b0	.8	1.0
PCI_AD[31:0]		I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
PCI_AD_OE#	195	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
PCI_C/BE[3:0]#	—	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
PCI_CLK	123	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
PCI_DEVSEL#	204	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
PCI_EXT_SEL	67	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
PCI_FRAME#	200	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
PCI_GNT#	54	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
PCI_IRDY#	201	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
PCI_LOCK#	53	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
PCI_OL_OPEN	64	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
PCI_OUT_SEL	68	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
PCI_PAR	7	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
PCI_PERR#	10	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
PCI_REQ#	58	O	CBUI, 13	PCI	B	6	6	cbug b 10	2.8	3.5
PCI_SERR#	71	O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
PCI_STOP#	203	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
PCI_TRDY#	202	I/O	CBUM, 10	PCI	B	6	6	cbuk b 10	3.1	3.7
RAS[7:0]#	—	O	CBNQ, 13	TTL	C	12	8	cbno c 13	3.5	4.1
RESET#	156	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
ROM_LOAD	70	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
ROM_OE#	47	O	CBNQ, 15	TTL	C	4	4	cbno c 15	2.9	3.5
ROM_WE#	60	O	CBNQ, 15	TTL	C	4	4	cbno c 15	2.9	3.5
SBE#	193	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0
SHD#	141	O	CBNX, 11	TTL	B	14	12	cbnw b 11	2.0	2.2
SRAM_ADS#/ ADDR0	124	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
SRAM_ALE	119	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
SRAM_CNT_EN#/ ADDR1	125	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
SRAM_OE#	117	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
SRAM_WE#	118	O	CBNX, 16	TTL	C	14	12	cbnw c 11	2.0	2.2
STOP_CLK_EN#	151	I	CBSZ, BO	TTL		—	—	cbsz a b0	.8	1.0
TA#	111	I/O	CBUM, 10	PCI	C	6	6	cbuk c 10	3.1	3.7

B

Table B-8. 664 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)	SPICE Card (4)	DPC (5)	
									Min	Max
TAG_CLR#	116	O	CBNX, 17	TTL	C	12	10	cbnw c 12	1.6	1.8
TAG_MATCH	142	I	CBUM, 10	PCI	B	—	—	cbuk b 10	3.1	3.7
TAG_VALID	115	O	CBNX, 17	TTL	C	12	10	cbnw c 12	1.6	1.8
TAG_WE#	114	O	CBNX, 17	TTL	C	12	10	cbnw c 12	1.6	1.8
TBST#	144	I/O	CBNZ, 11	TTL	B	14	12	cbnw b 11	2.0	2.2
TEA#	137	O	CBUM, 10	PCI	C	6	6	cbuk c 10	3.1	3.7
TEST#	155	I	CBSW, B3	TTL	A	—	—	cbsw a b3	.8	1.0
TS#	143	I/O	CBUM, 10	PCI	C	6	6	cbuk c 10	3.1	3.7
TSIZE[0:2]	—	I/O	CBNZ, 11	TTL	B	14	12	cbnw b 11	2.0	2.2
TT[0:4]	—	I/O	CBNZ, 11	TTL	B	14	12	cbnw b 11	2.0	2.2
WE[1:0]#	—	O	CBNQ, 13	TTL	C	12	8	cbno c 13	3.5	4.1
XATS#	129	I	CBJE, B0	TTL	F	—	—	cbjd a b0	.8	1.0

Notes:

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 D.4, Common Characteristics.

4) Use this SPICE card to model this signal. The 660 SPICE model package is available from IBM under Non-Disclosure Agreement. Contact your IBM technical representative for more information.

5) 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 B.5.

B

B.8 Package Drawings

B.8.1 663 Buffer Package Drawing

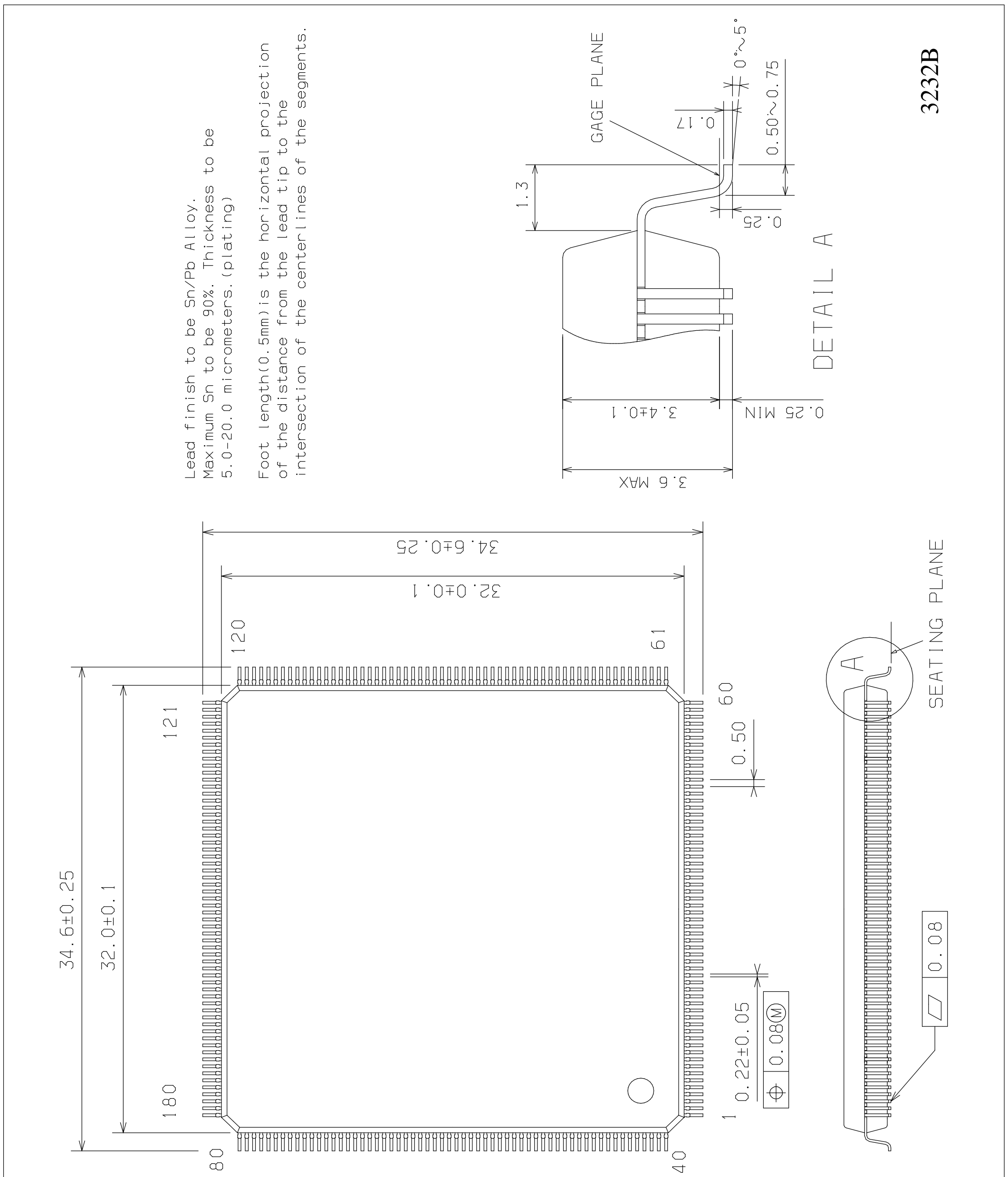


Figure B-2. 663 Buffer Package Drawing

See section 1.9 for package marking.

B.8.2 664 Controller Package Drawing

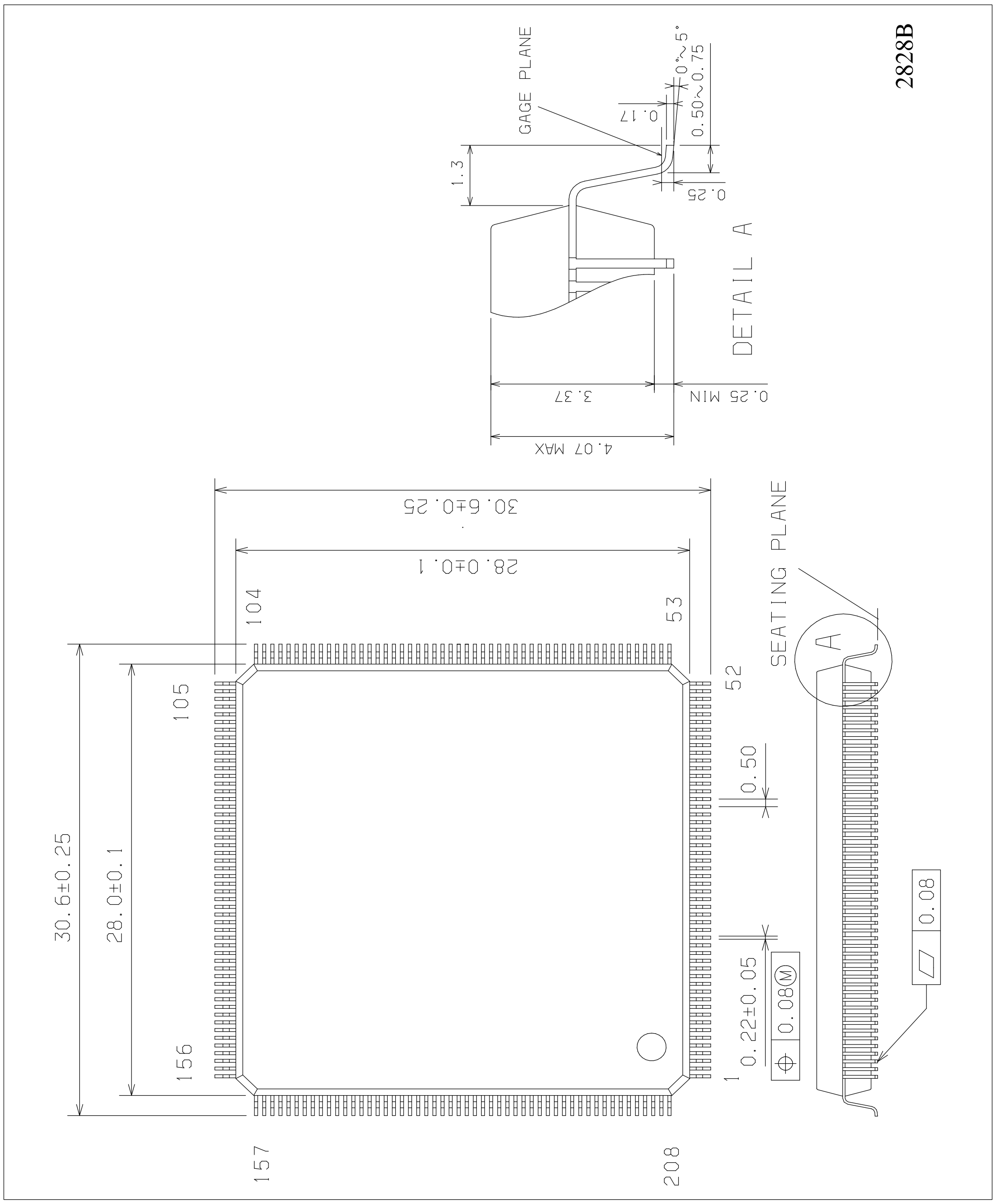


Figure B-3. 664 Controller Package Drawing

See Section 1.9 for package marking.

B

Appendix C
Pin Lists

Appendix C contains alphabetic pin lists and numeric pin lists for the 663 and the 664.

C.1 663 Buffer Alphabetic Pin List

Pin	663 Signal Name
166	AOS_RR_MMRS
154	C2P_WRL_OPEN
157	CPU_CLK
176	CPU_DATA[00]
177	CPU_DATA[01]
178	CPU_DATA[02]
179	CPU_DATA[03]
185	CPU_DATA[04]
186	CPU_DATA[05]
187	CPU_DATA[06]
188	CPU_DATA[07]
197	CPU_DATA[08]
198	CPU_DATA[09]
199	CPU_DATA[10]
207	CPU_DATA[11]
208	CPU_DATA[12]
209	CPU_DATA[13]
210	CPU_DATA[14]
218	CPU_DATA[15]
220	CPU_DATA[16]
221	CPU_DATA[17]
226	CPU_DATA[18]
227	CPU_DATA[19]
228	CPU_DATA[20]

Pin	663 Signal Name
229	CPU_DATA[21]
1	CPU_DATA[22]
2	CPU_DATA[23]
4	CPU_DATA[24]
24	CPU_DATA[25]
25	CPU_DATA[26]
26	CPU_DATA[27]
27	CPU_DATA[28]
31	CPU_DATA[29]
32	CPU_DATA[30]
33	CPU_DATA[31]
54	CPU_DATA[32]
55	CPU_DATA[33]
56	CPU_DATA[34]
57	CPU_DATA[35]
67	CPU_DATA[36]
68	CPU_DATA[37]
69	CPU_DATA[38]
70	CPU_DATA[39]
78	CPU_DATA[40]
79	CPU_DATA[41]
80	CPU_DATA[42]
86	CPU_DATA[43]
87	CPU_DATA[44]

Pin	663 Signal Name
88	CPU_DATA[45]
89	CPU_DATA[46]
94	CPU_DATA[47]
98	CPU_DATA[48]
99	CPU_DATA[49]
104	CPU_DATA[50]
105	CPU_DATA[51]
106	CPU_DATA[52]
107	CPU_DATA[53]
114	CPU_DATA[54]
115	CPU_DATA[55]
117	CPU_DATA[56]
124	CPU_DATA[57]
125	CPU_DATA[58]
126	CPU_DATA[59]
127	CPU_DATA[60]
134	CPU_DATA[61]
135	CPU_DATA[62]
136	CPU_DATA[63]
146	CPU_DATA_OE#
196	CPU_DPAR[0]
219	CPU_DPAR[1]
3	CPU_DPAR[2]
34	CPU_DPAR[3]

C

663 Buffer Alphabetic Pin List (Continued)

Pin	663 Signal Name
77	CPU_DPAR[4]
95	CPU_DPAR[5]
116	CPU_DPAR[6]
137	CPU_DPAR[7]
174	CPU_PAR_ERR#
148	CPU_RDL_OPEN
151	CRS_C2PWXS
170	DUAL_CTRL_REF
149	ECC_LE_SEL
9	GND
23	GND
39	GND
53	GND
66	GND
71	GND
85	GND
97	GND
120	GND
129	GND
143	GND
159	GND
173	GND
191	GND
205	GND
217	GND
235	GND
240	GND
161	MEM_BE[0]
162	MEM_BE[1]
163	MEM_BE[2]
164	MEM_BE[3]
141	MEM_CHECK[0]
122	MEM_CHECK[1]
103	MEM_CHECK[2]
82	MEM_CHECK[3]
37	MEM_CHECK[4]
234	MEM_CHECK[5]
214	MEM_CHECK[6]
195	MEM_CHECK[7]

Pin	663 Signal Name
180	MEM_DATA[00]
182	MEM_DATA[01]
183	MEM_DATA[02]
184	MEM_DATA[03]
189	MEM_DATA[04]
190	MEM_DATA[05]
193	MEM_DATA[06]
194	MEM_DATA[07]
200	MEM_DATA[08]
201	MEM_DATA[09]
202	MEM_DATA[10]
203	MEM_DATA[11]
206	MEM_DATA[12]
211	MEM_DATA[13]
212	MEM_DATA[14]
213	MEM_DATA[15]
215	MEM_DATA[16]
222	MEM_DATA[17]
223	MEM_DATA[18]
224	MEM_DATA[19]
225	MEM_DATA[20]
231	MEM_DATA[21]
232	MEM_DATA[22]
233	MEM_DATA[23]
5	MEM_DATA[24]
6	MEM_DATA[25]
7	MEM_DATA[26]
28	MEM_DATA[27]
29	MEM_DATA[28]
30	MEM_DATA[29]
35	MEM_DATA[30]
36	MEM_DATA[31]
58	MEM_DATA[32]
59	MEM_DATA[33]
60	MEM_DATA[34]
73	MEM_DATA[35]
74	MEM_DATA[36]
75	MEM_DATA[37]
76	MEM_DATA[38]

Pin	663 Signal Name
81	MEM_DATA[39]
83	MEM_DATA[40]
90	MEM_DATA[41]
91	MEM_DATA[42]
92	MEM_DATA[43]
93	MEM_DATA[44]
100	MEM_DATA[45]
101	MEM_DATA[46]
102	MEM_DATA[47]
108	MEM_DATA[48]
109	MEM_DATA[49]
111	MEM_DATA[50]
112	MEM_DATA[51]
113	MEM_DATA[52]
118	MEM_DATA[53]
119	MEM_DATA[54]
121	MEM_DATA[55]
123	MEM_DATA[56]
130	MEM_DATA[57]
131	MEM_DATA[58]
132	MEM_DATA[59]
133	MEM_DATA[60]
138	MEM_DATA[61]
139	MEM_DATA[62]
140	MEM_DATA[63]
145	MEM_DATA_OE#
171	MEM_ERR#
147	MEM_RD_SMPL
150	MEM_WRL_OPEN
156	MIO_TEST
152	MWS_P2MRXS
236	PCI_AD[00]
237	PCI_AD[01]
238	PCI_AD[02]
239	PCI_AD[03]
10	PCI_AD[04]
11	PCI_AD[05]
12	PCI_AD[06]
13	PCI_AD[07]

663 Buffer Alphabetic Pin List (Continued)

Pin	663 Signal Name
14	PCI_AD[08]
15	PCI_AD[09]
16	PCI_AD[10]
17	PCI_AD[11]
18	PCI_AD[12]
19	PCI_AD[13]
20	PCI_AD[14]
21	PCI_AD[15]
40	PCI_AD[16]
41	PCI_AD[17]
42	PCI_AD[18]
43	PCI_AD[19]
44	PCI_AD[20]
45	PCI_AD[21]
46	PCI_AD[22]
47	PCI_AD[23]
48	PCI_AD[24]

Pin	663 Signal Name
49	PCI_AD[25]
50	PCI_AD[26]
51	PCI_AD[27]
62	PCI_AD[28]
63	PCI_AD[29]
64	PCI_AD[30]
65	PCI_AD[31]
144	PCI_AD_OE#
153	PCI_EXT_SEL
167	PCI_IRDY#
165	PCI_OL_OPEN
169	PCI_OUT_SEL
168	PCI_TRDY#
160	ROM_LOAD
175	SBE#
155	TEST#
8	V_{DD}

Pin	663 Signal Name
22	V_{DD}
38	V_{DD}
52	V_{DD}
61	V_{DD}
72	V_{DD}
84	V_{DD}
96	V_{DD}
110	V_{DD}
128	V_{DD}
142	V_{DD}
158	V_{DD}
172	V_{DD}
181	V_{DD}
192	V_{DD}
204	V_{DD}
216	V_{DD}
230	V_{DD}

C

C.2 663 Buffer Numeric Pin List

Pin	663 Signal Name
1	CPU_DATA[22]
2	CPU_DATA[23]
3	CPU_DPAR[2]
4	CPU_DATA[24]
5	MEM_DATA[24]
6	MEM_DATA[25]
7	MEM_DATA[26]
8	V_{DD}
9	GND
10	PCI_AD[04]
11	PCI_AD[05]
12	PCI_AD[06]
13	PCI_AD[07]
14	PCI_AD[08]
15	PCI_AD[09]
16	PCI_AD[10]
17	PCI_AD[11]
18	PCI_AD[12]
19	PCI_AD[13]
20	PCI_AD[14]
21	PCI_AD[15]
22	V_{DD}
23	GND
24	CPU_DATA[25]
25	CPU_DATA[26]
26	CPU_DATA[27]
27	CPU_DATA[28]
28	MEM_DATA[27]
29	MEM_DATA[28]
30	MEM_DATA[29]
31	CPU_DATA[29]
32	CPU_DATA[30]
33	CPU_DATA[31]
34	CPU_DPAR[3]
35	MEM_DATA[30]
36	MEM_DATA[31]
37	MEM_CHECK[4]
38	V_{DD}
39	GND
40	PCI_AD[16]

Pin	663 Signal Name
41	PCI_AD[17]
42	PCI_AD[18]
43	PCI_AD[19]
44	PCI_AD[20]
45	PCI_AD[21]
46	PCI_AD[22]
47	PCI_AD[23]
48	PCI_AD[24]
49	PCI_AD[25]
50	PCI_AD[26]
51	PCI_AD[27]
52	V_{DD}
53	GND
54	CPU_DATA[32]
55	CPU_DATA[33]
56	CPU_DATA[34]
57	CPU_DATA[35]
58	MEM_DATA[32]
59	MEM_DATA[33]
60	MEM_DATA[34]
61	V_{DD}
62	PCI_AD[28]
63	PCI_AD[29]
64	PCI_AD[30]
65	PCI_AD[31]
66	GND
67	CPU_DATA[36]
68	CPU_DATA[37]
69	CPU_DATA[38]
70	CPU_DATA[39]
71	GND
72	V_{DD}
73	MEM_DATA[35]
74	MEM_DATA[36]
75	MEM_DATA[37]
76	MEM_DATA[38]
77	CPU_DPAR[4]
78	CPU_DATA[40]
79	CPU_DATA[41]
80	CPU_DATA[42]

Pin	663 Signal Name
81	MEM_DATA[39]
82	MEM_CHECK[3]
83	MEM_DATA[40]
84	V_{DD}
85	GND
86	CPU_DATA[43]
87	CPU_DATA[44]
88	CPU_DATA[45]
89	CPU_DATA[46]
90	MEM_DATA[41]
91	MEM_DATA[42]
92	MEM_DATA[43]
93	MEM_DATA[44]
94	CPU_DATA[47]
95	CPU_DPAR[5]
96	V_{DD}
97	GND
98	CPU_DATA[48]
99	CPU_DATA[49]
100	MEM_DATA[45]
101	MEM_DATA[46]
102	MEM_DATA[47]
103	MEM_CHECK[2]
104	CPU_DATA[50]
105	CPU_DATA[51]
106	CPU_DATA[52]
107	CPU_DATA[53]
108	MEM_DATA[48]
109	MEM_DATA[49]
110	V_{DD}
111	MEM_DATA[50]
112	MEM_DATA[51]
113	MEM_DATA[52]
114	CPU_DATA[54]
115	CPU_DATA[55]
116	CPU_DPAR[6]
117	CPU_DATA[56]
118	MEM_DATA[53]
119	MEM_DATA[54]
120	GND

663 Buffer Numeric Pin List (Continued)

Pin	663 Signal Name	Pin	663 Signal Name	Pin	663 Signal Name
121	MEM_DATA[55]	161	MEM_BE[0]	201	MEM_DATA[09]
122	MEM_CHECK[1]	162	MEM_BE[1]	202	MEM_DATA[10]
123	MEM_DATA[56]	163	MEM_BE[2]	203	MEM_DATA[11]
124	CPU_DATA[57]	164	MEM_BE[3]	204	V_{DD}
125	CPU_DATA[58]	165	PCI_OL_OPEN	205	GND
126	CPU_DATA[59]	166	AOS_RR_MMRS	206	MEM_DATA[12]
127	CPU_DATA[60]	167	PCI_IRDY#	207	CPU_DATA[11]
128	V_{DD}	168	PCI_TRDY#	208	CPU_DATA[12]
129	GND	169	PCI_OUT_SEL	209	CPU_DATA[13]
130	MEM_DATA[57]	170	DUAL_CTRL_REF	210	CPU_DATA[14]
131	MEM_DATA[58]	171	MEM_ERR#	211	MEM_DATA[13]
132	MEM_DATA[59]	172	V_{DD}	212	MEM_DATA[14]
133	MEM_DATA[60]	173	GND	213	MEM_DATA[15]
134	CPU_DATA[61]	174	CPU_PAR_ERR#	214	MEM_CHECK[6]
135	CPU_DATA[62]	175	SBE#	215	MEM_DATA[16]
136	CPU_DATA[63]	176	CPU_DATA[00]	216	V_{DD}
137	CPU_DPAR[7]	177	CPU_DATA[01]	217	GND
138	MEM_DATA[61]	178	CPU_DATA[02]	218	CPU_DATA[15]
139	MEM_DATA[62]	179	CPU_DATA[03]	219	CPU_DPAR[1]
140	MEM_DATA[63]	180	MEM_DATA[00]	220	CPU_DATA[16]
141	MEM_CHECK[0]	181	V_{DD}	221	CPU_DATA[17]
142	V_{DD}	182	MEM_DATA[01]	222	MEM_DATA[17]
143	GND	183	MEM_DATA[02]	223	MEM_DATA[18]
144	PCI_AD_OE#	184	MEM_DATA[03]	224	MEM_DATA[19]
145	MEM_DATA_OE#	185	CPU_DATA[04]	225	MEM_DATA[20]
146	CPU_DATA_OE#	186	CPU_DATA[05]	226	CPU_DATA[18]
147	MEM_RD_SMPL	187	CPU_DATA[06]	227	CPU_DATA[19]
148	CPU_RDL_OPEN	188	CPU_DATA[07]	228	CPU_DATA[20]
149	ECC_LE_SEL	189	MEM_DATA[04]	229	CPU_DATA[21]
150	MEM_WRL_OPEN	190	MEM_DATA[05]	230	V_{DD}
151	CRS_C2PWXS	191	GND	231	MEM_DATA[21]
152	MWS_P2MRXS	192	V_{DD}	232	MEM_DATA[22]
153	PCI_EXT_SEL	193	MEM_DATA[06]	233	MEM_DATA[23]
154	C2P_WRL_OPEN	194	MEM_DATA[07]	234	MEM_CHECK[5]
155	TEST#	195	MEM_CHECK[7]	235	GND
156	MIO_TEST	196	CPU_DPAR[0]	236	PCI_AD[00]
157	CPU_CLK	197	CPU_DATA[08]	237	PCI_AD[01]
158	V_{DD}	198	CPU_DATA[09]	238	PCI_AD[02]
159	GND	199	CPU_DATA[10]	239	PCI_AD[03]
160	ROM_LOAD	200	MEM_DATA[08]	240	GND

C

C.3 664 Controller Alphabetic Pin Lists

664 Signal Name	Pin
AACK#	109
ADDR[00]	72
ADDR[01]	73
ADDR[02]	74
ADDR[03]	75
ADDR[04]	76
ADDR[05]	77
ADDR[06]	80
ADDR[07]	81
ADDR[08]	82
ADDR[09]	83
ADDR[10]	84
ADDR[11]	85
ADDR[12]	86
ADDR[13]	87
ADDR[14]	89
ADDR[15]	90
ADDR[16]	91
ADDR[17]	92
ADDR[18]	93
ADDR[19]	96
ADDR[20]	97
ADDR[21]	98
ADDR[22]	99
ADDR[23]	100
ADDR[24]	101
ADDR[25]	102
ADDR[26]	103
ADDR[27]	104
ADDR[28]	105
ADDR[29]	106
ADDR[30]	107
ADDR[31]	108
AOS_RR_MMRS	69
ARTRY#	110
C2P_WRL_OPEN	61
CAS[0]#	174
CAS[1]#	173
CAS[2]#	172
CAS[3]#	171

664 Signal Name	Pin
CAS[4]#	170
CAS[5]#	169
CAS[6]#	168
CAS[7]#	165
CPU_BUS_CLAIM#	132
CPU_CLK	121
CPU_DATA_OE#	197
CPU_GNT1#	134
CPU_GNT2#	135
CPU_PAR_ERR#	192
CPU_RDL_OPEN	50
CPU_REQ1#	127
CPU_REQ2#	128
CRS_C2PWXS	65
DBG#	140
DPE#	133
DUAL_CTRL_REF	205
ECC_LE_SEL	2
GBL#	120
GND[01]	9
GND[02]	17
GND[03]	27
GND[04]	45
GND[05]	52
GND[06]	63
GND[07]	79
GND[08]	88
GND[09]	95
GND[10]	113
GND[11]	131
GND[12]	149
GND[13]	167
GND[14]	183
GND[15]	199
IGN_PCI_AD31	57
INT_CPU#	139
INT_REQ	55
MA[00]	190
MA[01]	189
MA[02]	188

664 Signal Name	Pin
MA[03]	187
MA[04]	186
MA[05]	185
MA[06]	184
MA[07]	181
MA[08]	180
MA[09]	179
MA[10]	178
MA[11]	177
MCP#	138
MEM_BE[0]	206
MEM_BE[1]	207
MEM_BE[2]	208
MEM_BE[3]	1
MEM_DATA_OE#	196
MEM_ERR#	194
MEM_RD_SMPL	49
MEM_WRL_OPEN	51
MIO_TEST	154
MWS_P2MRXS	66
NMI_REQ	56
PCI_AD[00]	48
PCI_AD[01]	59
PCI_AD[02]	46
PCI_AD[03]	43
PCI_AD[04]	42
PCI_AD[05]	41
PCI_AD[06]	40
PCI_AD[07]	39
PCI_AD[08]	38
PCI_AD[09]	37
PCI_AD[10]	36
PCI_AD[11]	35
PCI_AD[12]	34
PCI_AD[13]	33
PCI_AD[14]	32
PCI_AD[15]	31
PCI_AD[16]	30
PCI_AD[17]	29
PCI_AD[18]	28

664 Controller Alphabetic Pin List (Continued)

664 Signal Name	Pin
PCI_AD[19]	25
PCI_AD[20]	24
PCI_AD[21]	23
PCI_AD[22]	22
PCI_AD[23]	21
PCI_AD[24]	20
PCI_AD[25]	19
PCI_AD[26]	18
PCI_AD[27]	15
PCI_AD[28]	14
PCI_AD[29]	13
PCI_AD[30]	12
PCI_AD[31]	11
PCI_AD_OE#	195
PCI_C/BE[0]#	6
PCI_C/BE[1]#	5
PCI_C/BE[2]#	4
PCI_C/BE[3]#	3
PCI_CLK	123
PCI_DEVSEL#	204
PCI_EXT_SEL	67
PCI_FRAME#	200
PCI_GNT#	54
PCI_IRDY#	201
PCI_LOCK#	53
PCI_OL_OPEN	64
PCI_OUT_SEL	68
PCI_PAR	7
PCI_PERR#	10
PCI_REQ#	58

664 Signal Name	Pin
PCI_SERR#	71
PCI_STOP#	203
PCI_TRDY#	202
RAS[0]#	164
RAS[1]#	163
RAS[2]#	162
RAS[3]#	161
RAS[4]#	160
RAS[5]#	159
RAS[6]#	158
RAS[7]#	157
RESET#	156
ROM_LOAD	70
ROM_OE#	47
ROM_WE#	60
SBE#	193
SHD#	141
SRAM_ADS#/ADDR0	124
SRAM_ALE	119
SRAM_CNT_EN#/ADDR1	125
SRAM_OE#	117
SRAM_WE#	118
STOP_CLK_EN#	151
TA#	111
TAG_CLR#	116
TAG_MATCH	142
TAG_VALID	115
TAG_WE#	114
TBST#	144

664 Signal Name	Pin
TEA#	137
TEST#	155
TS#	143
TSIZE[0]	145
TSIZE[1]	146
TSIZE[2]	147
TT[0]	150
TT[1]	152
TT[2]	153
TT[3]	126
TT[4]	136
VDD[01]	8
VDD[02]	16
VDD[03]	26
VDD[04]	44
VDD[05]	62
VDD[06]	78
VDD[07]	94
VDD[08]	112
VDD[09]	122
VDD[10]	130
VDD[11]	148
VDD[12]	166
VDD[13]	182
VDD[14]	191
VDD[15]	198
WE[0]#	176
WE[1]#	175
XATS#	129

C.4 664 Controller Numeric Pins

Pin	664 Signal Name	Pin	664 Signal Name	Pin	664 Signal Name
1	MEM_BE[3]	36	PCI_AD[10]	71	PCI_SERR#
2	ECC_LE_SEL	37	PCI_AD[09]	72	ADDR[00]
3	PCI_C/BE[3]#	38	PCI_AD[08]	73	ADDR[01]
4	PCI_C/BE[2]#	39	PCI_AD[07]	74	ADDR[02]
5	PCI_C/BE[1]#	40	PCI_AD[06]	75	ADDR[03]
6	PCI_C/BE[0]#	41	PCI_AD[05]	76	ADDR[04]
7	PCI_PAR	42	PCI_AD[04]	77	ADDR[05]
8	VDD[01]	43	PCI_AD[03]	78	VDD[06]
9	GND[01]	44	VDD[04]	79	GND[07]
10	PCI_PERR#	45	GND[04]	80	ADDR[06]
11	PCI_AD[31]	46	PCI_AD[02]	81	ADDR[07]
12	PCI_AD[30]	47	ROM_OE#	82	ADDR[08]
13	PCI_AD[29]	48	PCI_AD[00]	83	ADDR[09]
14	PCI_AD[28]	49	MEM_RD_SMPL	84	ADDR[10]
15	PCI_AD[27]	50	CPU_RDL_OPEN	85	ADDR[11]
16	VDD[02]	51	MEM_WRL_OPEN	86	ADDR[12]
17	GND[02]	52	GND[05]	87	ADDR[13]
18	PCI_AD[26]	53	PCI_LOCK#	88	GND[08]
19	PCI_AD[25]	54	PCI_GNT#	89	ADDR[14]
20	PCI_AD[24]	55	INT_REQ	90	ADDR[15]
21	PCI_AD[23]	56	NMI_REQ	91	ADDR[16]
22	PCI_AD[22]	57	IGN_PCI_AD31	92	ADDR[17]
23	PCI_AD[21]	58	PCI_REQ#	93	ADDR[18]
24	PCI_AD[20]	59	PCI_AD[01]	94	VDD[07]
25	PCI_AD[19]	60	ROM_WE#	95	GND[09]
26	VDD[03]	61	C2P_WRL_OPEN	96	ADDR[19]
27	GND[03]	62	VDD[05]	97	ADDR[20]
28	PCI_AD[18]	63	GND[06]	98	ADDR[21]
29	PCI_AD[17]	64	PCI_OL_OPEN	99	ADDR[22]
30	PCI_AD[16]	65	CRS_C2PWXS	100	ADDR[23]
31	PCI_AD[15]	66	MWS_P2MRXS	101	ADDR[24]
32	PCI_AD[14]	67	PCI_EXT_SEL	102	ADDR[25]
33	PCI_AD[13]	68	PCI_OUT_SEL	103	ADDR[26]
34	PCI_AD[12]	69	AOS_RR_MMRS	104	ADDR[27]
35	PCI_AD[11]	70	ROM_LOAD	105	ADDR[28]

664 Controller Numeric Pin List (Continued)

Pin	664 Signal Name
106	ADDR[29]
107	ADDR[30]
108	ADDR[31]
109	AACK#
110	ARTRY#
111	TA#
112	VDD[08]
113	GND[10]
114	TAG_WE#
115	TAG_VALID
116	TAG_CLR#
117	SRAM_OE#
118	SRAM_WE#
119	SRAM_ALE
120	GBL#
121	CPU_CLK
122	VDD[09]
123	PCI_CLK
124	SRAM_ADS#/ADDR0
125	SRAM_CNT_EN#/ADDR1
126	TT[3]
127	CPU_REQ1#
128	CPU_REQ2#
129	XATS#
130	VDD[10]
131	GND[11]
132	CPU_BUS_CLAIM#
133	DPE#
134	CPU_GNT1#
135	CPU_GNT2#
136	TT[4]
137	TEA_
138	MCP#
139	INT_CPU#
140	DBG#

Pin	664 Signal Name
141	SHD#
142	TAG_MATCH
143	TS#
144	TBST#
145	TSIZE[0]
146	TSIZE[1]
147	TSIZE[2]
148	VDD[11]
149	GND[12]
150	TT[0]
151	STOP_CLK_EN#
152	TT[1]
153	TT[2]
154	MIO_TEST
155	TEST#
156	RESET#
157	RAS[7]#
158	RAS[6]#
159	RAS[5]#
160	RAS[4]#
161	RAS[3]#
162	RAS[2]#
163	RAS[1]#
164	RAS[0]#
165	CAS[7]#
166	VDD[12]
167	GND[13]
168	CAS[6]#
169	CAS[5]#
170	CAS[4]#
171	CAS[3]#
172	CAS[2]#
173	CAS[1]#
174	CAS[0]#

Pin	664 Signal Name
175	WE[1]#
176	WE[0]#
177	MA[11]
178	MA[10]
179	MA[09]
180	MA[08]
181	MA[07]
182	VDD[13]
183	GND[14]
184	MA[06]
185	MA[05]
186	MA[04]
187	MA[03]
188	MA[02]
189	MA[01]
190	MA[00]
191	VDD[14]
192	CPU_PAR_ERR#
193	SBE#
194	MEM_ERR#
195	PCI_AD_OE#
196	MEM_DATA_OE#
197	CPU_DATA_OE#
198	VDD[15]
199	GND[15]
200	PCI_FRAME#
201	PCI_IRDY#
202	PCI_TRDY#
203	PCI_STOP#
204	PCI_DEVSEL#
205	DUAL_CTRL_REF
206	MEM_BE[0]
207	MEM_BE[1]
208	MEM_BE[2]

Appendix D
FAQs

This FAQ contains 660 bridge information which the PowerPC Application Engineering team believes to be of interest to the PowerPC community, and which are too short to be the subject of individual application notes.

The following information is current as of the date of this document. See your IBM representative for the latest information on the 660.

D.1)	4/9/96	Where can I get 660 information ?
D.2)	4/9/96	Can the 660 address and data buses be left tristated ?
D.3)	4/9/96	What are the access times for 50ns, 60ns, & 70ns EDO DRAM ?
D.4)	4/9/96	Is my DRAM fast enough to do 660 RMW cycles in ECC mode ?
D.5)	4/11/96	How do interrupt acknowledge cycles work ?
D.6)	6/20/96	660 User's Manual Corrections
D.7)	6/20/96	PCI Master Data Latency
D.8)	6/20/96	PCI Target Initial Latency
D.9)	6/20/96	Exception to PCI Revision 2.1 Compliance – Transaction Ordering Rules
D.10)	6/20/96	How are the CAS# lines and check bits used in ECC mode ?
D.11)	9/26/96	Optimizing PCI Performance
D.12)	10/30/97	Memory Access
D.13)	10/30/97	DRAM Timing

D.1 Where can I get 660 information ?

Date 4/9/96

Description . . Currently, 660 data is located in several places. IBM employes (such as your FAE or other IBM technical representative) can access this information using the addresses marked *Internal*. This information can be accessed externally using the other addresses.

D

660 Bridge – User's Manual and other information.

Internal ...... /afs/awd/work/PowerPC/support_chips/660_bridge/
From AIX, rftp
ftp.austin.ibm.com/pub/PPC_support/660_bridge
External ..... ftp://ftp.austin.ibm.com/pub/PPC_support/660_bridge/
The IP address is (192.35.232.38).

Reference Designs – Technical Specifications and design files showing how to use the 660 and other PowerPC products.

Internal ...... /afs/awd/work/PowerPC/reference_designs/
From AIX, rftp
ftp.austin.ibm.com/pub/PPC_support/reference_designs
External ..... ftp://ftp.austin.ibm.com/pub/PPC_support/reference_de
signs/
the IP address is (192.35.232.38).

Application Notes – Technical articles from the PowerPC Application Engineering team about the 660 and other PowerPC products.

Internal ...... /afs/awd/work/PowerPC/app_notes/
From AIX, rftp
ftp.austin.ibm.com/pub/PPC_support/app_notes
External ..... ftp://ftp.austin.ibm.com/pub/PPC_support/app_notes/
The IP address is (192.35.232.38).

D.2 Can the 660 address and data buses be left tristated ?

Date 4/9/96

Description .. We have not seen any functional problems with the 660 which have been caused by allowing CPU_ADDR[0:31] and CPU_DATA[0:63] to float for an extended amount of time (several seconds).

However, the 660 is a CMOS device, so allowing the inputs to float allows the possibility that the input voltage will float to a level that causes both of the input transistors to conduct some current, thus increasing the power dissipation of these inputs. A more remote possibility is that these inputs will oscillate, which will also increase the power dissipation of the inputs without causing functional problems. As with all electrical devices, system level and handling precautions should be taken to ensure that the 660 is not exposed to ESD events in excess of its published protection limits.

D

D.3 What are the access times for 50ns, 60ns, & 70ns EDO DRAM ?

Date 4/9/96

Description .. Additional access times are now available for 50ns, 60ns, and 70ns EDO DRAM (see Table 5-2 and Table D-2). These timings are presented in the same manner as the ones in Section 5 of the 660UM. Notice that the column in Table D-2 for 60ns aggressive timing corrects an error in some copies of the 660 UM.

Table D-1. General Case EDO DRAM Timing Summary

Transfer	**70ns EDO Conservative**	**70ns EDO Aggressive**	**60ns EDO Conserva-tive**	**Note**
Memory Timing Register 1	32	32	31	
Memory Timing Register 2	0A	06	06	
Bridge Chipset Options 3	0C	0C	0C	
Initial Read Burst	12-4-4-4	11-3-3-3	11-3-3-3	(3)
Initial Write Burst	5-4-4-4	5-4-3-3	5-4-3-3	
For a pipelined burst transfer immediately following a read:				
Page Hit Read	-5-4-4-4	-5-3-3-3	-5-3-3-3	(3)
Page Hit Write	-3-3-4-4	-3-3-3-3	-3-3-3-3	
Page Miss & Bank Miss Read	-9-4-4-4	-8-3-3-3	-8-3-3-3	(1,3)
Page Miss & Bank Hit Read	-11-3-3-3	-10-3-3-3	-9-3-3-3	(2,3)
Page Miss & Bank Miss Write	-3-3-4-4	-3-3-3-3	-3-3-3-3	(1)
Page Miss & Bank Hit Write	-3-5-4-4	-3-5-3-3	-3-4-3-3	(2)
For a pipelined burst transfer immediately following a write:				
Page Hit Read	-12-4-4-4	-9-3-3-3	-9-3-3-3	(3)
Page Hit Write	-6-3-4-4	-5-3-3-3	-5-3-3-3	
Page Miss & Bank Miss Read	-15-4-4-4	-12-3-3-3	-12-3-3-3	(1,3)
Page Miss & Bank Hit Read	-17-4-4-4	-14-3-3-3	-13-3-3-3	(2,3)
Page Miss & Bank Miss Write	-6-6-4-4	-5-5-3-3	-5-5-3-3	(1)
Page Miss & Bank Hit Write	-6-8-4-4	-5-7-3-3	-5-6-3-3	(2)

See the notes for Table D-2.

D

Table D-2. General Case EDO DRAM Timing Summary

Transfer	60ns EDO Aggressive	50ns EDO Conservative	50ns EDO Aggressive	Note
Memory Timing Register 1	11	0C	2C	
Memory Timing Register 2	05	05	02	
Bridge Chipset Options 3	0C	0C	0C	
Initial Read Burst	10-3-3-3	10-3-3-3	10-2-2-2	(3)
Initial Write Burst	5-3-3-3	5-3-3-3	5-4-2-2	
For a pipelined burst transfer immediately following a read:				
Page Hit Read	-5-3-3-3	-5-3-3-3	-5-2-2-2	(3)
Page Hit Write	-3-3-3-3	-3-3-3-3	-3-2-2-2	
Page Miss & Bank Miss Read	-7-3-3-3	-7-3-3-3	-7-2-2-2	(1,3)
Page Miss & Bank Hit Read	-8-3-3-3	-7-3-3-3	-7-2-2-2	(2,3)
Page Miss & Bank Miss Write	-3-3-3-3	-3-3-3-3	-3-3-2-2	(1)
Page Miss & Bank Hit Write	-3-3-3-3	-3-3-3-3	-3-3-2-2	(2)
For a pipelined burst transfer immediately following a write:				
Page Hit Read	-9-3-3-3	-9-3-3-3	-7-2-2-2	(3)
Page Hit Write	-5-3-3-3	-5-3-3-3	-4-2-2-2	
Page Miss & Bank Miss Read	-11-3-3-3	-11-3-3-3	-10-2-2-2	(1,3)
Page Miss & Bank Hit Read	-12-3-3-3	-11-3-3-3	-10-2-2-2	(2,3)
Page Miss & Bank Miss Write	-5-4-3-3	-5-4-3-3	-4-5-2-2	(1)
Page Miss & Bank Hit Write	-5-5-3-3	-5-4-3-3	-4-5-2-2	(2)

1) The RAS# of the new bank is high and has been high (precharging) for the minimum RAS# high time. The bridge places the address on the address lines and asserts RAS#.
2) The access is a page miss, but within the same bank, so the RAS# line must be sent high for at least the minimum RAS# high (precharge) time. The bridge also places the new address on the address lines and asserts RAS#.
3) If asynchronous SRAMs are used with the internal L2 controller, an additional clock cycle is added to the fourth beat of any CPU to memory read burst that causes a cache miss. For example, following a read—a pipelined page hit, cache miss, burst read with EDO DRAM, requires –3–3–3–3 CPU clocks when the L2 uses burst SRAMs and –3–3–3–4 CPU clocks when the L2 uses asynchronous SRAMs. This extra beat is caused by delaying the final TA# by one CPU_CLK to allow the asynchronous SRAM sufficient data hold time for the fourth beat.

D

D.4 Is my DRAM fast enough to do 660 RMW cycles in ECC mode ?

Date 4/9/96

Description . . In ECC mode, the 660 can do a read–modify–write cycle to the DRAM if the CPU or a PCI busmaster initiates a memory write of less than 8 bytes.

As shown in Section A.7.5 of the 660UM, when the 660 does a RMW cycle, it does not use the RMW cycle capability that is built into some DRAMs. Instead, it does a read, modifies the data, and then writes back the data (as a pipelined page hit write). This allows RMW cycles with any DRAM that is fast enough to perform read cycles and write cycles. No recalculation of the CAS# pulse width is required.

D.5 How do interrupt acknowledge cycles work ?

Date 4/11/96

Description . . To perform an interrupt acknowledge operation, the CPU initiates a single-byte read to address BFFF FFF0. This causes the 660 to arbitrate for the PCI bus and then to initiate a single-byte PCI Interrupt Acknowledge transaction with PCI_C/BE[0]# active. PCI_C/BE[0]# is active regardless of the endian mode of the 660. The PCI Interrupt Acknowledge transaction that the 660 generates is similar to those generated by x86 to PCI bridges. The interrupt controller (eg SIO) then claims the transaction and supplies the single-byte interrupt vector on PCI byte lane 0. The 660 then returns the vector to the CPU on the correct byte lane.

There is no physical interrupt vector BCR in the bridge. Other PCI bus masters can initiate interrupt acknowledge transactions.

D.6 660 User's Manual Corrections – SC09–3026–00, 660umf2.ps, 660umf3.ps, & 660umf4.ps

Date 6/20/96

5.2.2 General Case DRAM Timing Calculations (6/20/96). In items 4, 5b, and 6, the term (+ 1 CLK if EDO) is incorrectly shown on the right side of the equation. This term should be on the left side of each equation, as shown below:

4. CAS# pulse width (CPW)

(+ 1 CLK if EDO) >	T CAS# fall max .	* CAS# active out if 664
	+ Tcac	* DRAM data access from . CAS#
	+ MD setup max .	* MEM_DATA setup into 663
		* data sampled 1 clk later

The last factor (+ 1 CLK if EDO) is included in the equation only if EDO DRAM is used. Note that CPW must be set to three or fewer clocks.

D

5. b) Col addr setup (ASC)
   + CAS# pulse width (CPW)
   (+ 1 CLK if EDO) > T MA max . . . * MA[11:0] valid out of 664
   + T 244 max * Buffer delay
   + Taa min . . . * DRAM data valid from col addr
   + MD setup max * MEM_DATA setup into 663

6. RAS# to CAS# delay (RCD)
   + CAS# pulse width (CPW)
   (+ 1 CLK if EDO) > T RAS# fall max . . . * max 660 RAS# fall time (note 1)
   + Trac min . . . . * DRAM data access . . . . from RAS#
   + MD setup max/ . . * MEM_DATA setup into 663

5.4.1 **Memory Parity (6/20/96)**. Change the last word of the first paragraph from PCI_PERR# to PCI_SERR#.

6 **L2 Cache (6/20/96)**. Change the second sentence to read: The L2 caches *as much of the* system memory space from 0 to 2G *as is populated by DRAM.*

10.3.34 **Single-Bit Error Trigger Level Register (6/20/96)**. Change the second paragraph to read If the single-bit error level trigger register is set to 00h, no *single-bit ECC* error is ever generated to the CPU. *Setting this BCR to 00h does not inhibit the recognition or reporting of multi-bit ECC errors.*

D.7 PCI Master Data Latency

Date 6/20/96

Description . . The 2.1 PCI specification allows host bridges a Master Data Latency of 8 PCI clocks. While the 660 is in 1:1 CPU:PCI clock mode, it is possible to configure the 660 memory controller to operate the DRAM slowly enough that the master data latency becomes as large as 10 PCI clocks. This does not imply that the 660 is noncompliant, but that it is possible for the designer to use the 660 in a noncompliant manner.

D.8 PCI Target Initial Latency

Date 6/20/96

Description . . The 2.1 PCI specification allows host bridges a Target Initial Latency (TIL) of 32 PCI clocks under certain conditions. When the 660 is in 2:1 CPU:PCI clock mode, and is used with one busmaster on the CPU bus, the TIL is a maximum of 32 PCI clocks.

However, when the 660 is in 1:1 CPU:PCI clock mode with a single busmaster on the CPU bus, it is possible to configure the 660 memory controller to operate the DRAM slowly enough that the TIL becomes as large as 40 PCI clocks. In systems having more than one CPU busmaster, this latency can increase.

Also, when the 660 is used in 2:1 CPU:PCI clock mode with two busmasters on the CPU bus, it is possible to configure the 660 memory controller to operate the DRAM slowly enough that the TIL exceeds 32 PCI clocks. In systems operating at 1:1 CPU:PCI clock mode, this latency can increase

D.9 Exception to 660 PCI Revision 2.1 Compliance – Transaction Ordering Rules

Date 6/20/96

Description . . Transaction ordering requirements were added to the PCI specification between revs 2.0 and 2.1. The PCI 2.1 specification states that writes from a given master must be visible (by any other agent in the system) as completing in the order in which they were initiated. This requirement is intended to implement the Producer–Consumer model contained in Appendix E of the PCI 2.1 specification.

There is a theoretical case in which the 660 does not comply with the PCI Producer–Consumer model (Appendix E, Summary of PCI Ordering Rules, Item 5b). Note that to date (6/20/96) there have been no know actual occurrances of this case with the 660. It was recently discovered during a comprehensive review of the PCI 2.1 specification. IBM is not aware of any existing hardware/software combination that produces a situation where this case can be observed.

For the following discussion, please refer to the PCI 2.1 specification sections 3.2.5 and Appendix E. Assume that the CPU is the Producer, and that it is producing the final 4 bytes of a data set by initiating a PCI memory write (via the 660) to a particular PCI agent (the Consumer). Assume that the 660 posts the write, and that before the posted write completes on the PCI bus (for instance if it is retried), the CPU writes the flag to system memory.

Meanwhile, the PCI agent is polling the flag by periodically reading the system memory location. Assume that the PCI agent reads the flag as set, meaning that the data set transfer is complete. At this point, the possibility exists that the PCI agent has not allowed the CPU to PCI (data set) write to complete. If so, then the two CPU transactions have completed out of order, and the flag will indicate that the data transfer has completed, when in fact it has not.

To prevent this, the PCI 2.1 spec requires the bridge to react to the PCI to memory read request by pulling all posted writes through the bridge before completing the read. In other words, when the PCI agent initiates the read (polls the flag), the bridge is to retry (or delay) the read until the posted CPU to PCI memory write (the data set transfer) has completed. This has the effect of ”pulling” the writes through the bridge before the read is executed. The 660 does not implement this function.

Note that PCI devices (such as SCSI, Ethernet, and Token Ring adaptors) that poll the flag in system memory typically also consume the data set by initiating PCI to system memory reads. These devices are not affected.

Also, PCI devices (such as graphics adaptors and non–PCI busmasters) that receive the data set as a PCI memory target typically receive indication that the data is ready (the flag) as a PCI I/O or PCI memory target; they do not poll system memory for the flag. These devices are also not affected.

For affected devices, the following options apply:

Option 1: Add a dummy CPU to PCI write:

Design the device drivers to cause the CPU BIU to execute an access to the PCI bus (of any type) after the data set is written (posted into the 660 by the CPU write to the PCI target) and before the CPU BIU executes the write that sets the flag in system memory. This solution forces the data set write to complete on the PCI bus before the flag write is initiated on the CPU bus. (Remember to insert eieio instructions before and after the dummy write.) This produces the following sequence:

1) The CPU initiates a CPU to PCI write to produce the final part of the data set. The CPU address tenure completes, and the PCI write is posted.

2) Some time later, the 660 initiates this transaction on the PCI bus. The transaction takes an unknown amount of time to complete. If the transaction is retried on the PCI bus, the 660 will continue to reinnitiate the transaction until it completes.

3) After (1) and before (4), the CPU initiates a dummy CPU to PCI write to flush the posted write buffer. The 660 does not allow this second CPU to PCI write to complete on the CPU bus (by being written into the 660 posted write buffer) until the first CPU to PCI transaction (the data set write) actually completes on the PCI bus. This prevents the following transaction (the flag write) from completing before the data set write completes.

4) The CPU initiates the CPU to memory flag write.

Option 2: Change the flag write procedure:

Design the device drivers such that the Consumer receives the flag in some way that ensures that the data set write completes before the flag write completes. One way to do this is to cause the CPU to write the flag to the PCI agent using a PCI IO or memory write, instead of a CPU to memory write.

a) If the flag write is a PCI memory write, then as in item 3 in option 1, this flag write forces the preceeding data set write to complete before the flag write completes.

b) If the flag write is a PCI IO write, then the 660 will not post the flag write, which forces the data set write(s) to complete on the PCI bus before the flag write is initiated on the PCI bus.

Option 3: Change the data set write procedure:

Design the device drivers such that the Consumer receives data set some way other than as a PCI memory target (such as an as an I/O target or a PCI busmaster).

a) While the Consumer is receiving the data set as a PCI IO target, the 660 does not post the IO writes, and so each write completes in order, even if some are retried. This technique forces the data set writes to complete before the flag write completes.

b) If the Consumer is receiving the data set by initiating PCI to memory reads, and is receiving the flag by initiating a PCI to memory read, then as long as the CPU initiates the data set writes to memory before the CPU initiates the flag write to memory, then the ordering of the transactions will be preserved.

D.10 How are the CAS# lines and check bits used in ECC mode ?

Date 9/26/96

Description . . While the 660 is in ECC mode, the CAS# lines are used in the same fashion as they are while the 660 is in parity mode. They are connected in the same way as they are for parity DRAMs. Each CAS# line goes to a separate byte, and serves as the column address strobe, the byte enable, and a chip enable. In both parity and ECC modes of operation, one bit of the check byte is stored with each byte of data.

Please see the schematics and the DRAM data sheets in the Harley and Zapatos reference designs for connectivity information. Note that CAS_P# is grounded to the DRAMs, and that each DRAM is connected to one byte lane (8 data bits) and one check bit.

Parity Mode

In parity mode, the value of a given check (parity) bit is a function only of the data byte with which it is associated. Therefore in parity mode, the 660 updates the check (parity) bit whenever (and only whenever) the associated data byte is changed.

Parity mode operation allows the memory controller write logic to execute write cycles on data types that are less than 8 bytes. For example, the 660 handles a 4–byte store operation from the CPU bus or PCI bus by writing the 4 bytes to DRAM. The parity bits associated with the affected 4 bytes are automatically upddated as the associated bytes are written.

D

ECC Mode

In ECC mode, the value of the check (ECC) bit is a function not only of the data byte with which it is associated, but also of each of the other data bytes. Thus any change to one or more bytes of date can require a change to any set of the check bits, even check bits that are not associated with the data bytes that are being written. Thus each of the check bits must be updated whenever any data byte is changed. Also please see 660 UM section 5.4.5 and 5.4.6.

For 8–byte writes, the process is automatic, since all of the check bits are written.

However, for writes of less than 8 bytes, The 660 must update all of the check bits. Since the check bits are updated whenever the associated byte is updated, these writes require a read–modify–write (RMW) cycle. The 660 latches the write data from the CPU or PCI bus into a register, and does a DRAM read (always an 8–byte operation). The DRAM bytes that are not being written by the CPU are written into the 660 register. The ECC bits are recalculated and written into the check bits of the 660 register. Then the 660 register is written back to the DRAM. Actually all of this takes place rather asynchronously.

For a CPU (or PCI) write to DRAM byte lanes 0:4, first the CPU write places the write data in the 660 data register. Then the memory read fills bytes 4:7 from DRAM. The check bits are recalculated and are updated in the 660 register. Then the entire 8–byte (plus one check bit per data byte) dword is written to DRAM.

```
CPU Byte 0 ---> 660 Register Byte 0 --> DRAM Byte 0  <--------+
CPU Byte 1 ---> 660 Register Byte 1 --> DRAM Byte 1  <-------+|
CPU Byte 2 ---> 660 Register Byte 2 --> DRAM Byte 2  <------+||
CPU Byte 3 ---> 660 Register Byte 3 --> DRAM Byte 3  <-----+|||
DRAM Byte 4 --> 660 Register Byte 4 --> DRAM Byte 4  <----+||||
DRAM Byte 5 --> 660 Register Byte 5 --> DRAM Byte 5  <---+|||||
DRAM Byte 6 --> 660 Register Byte 6 --> DRAM Byte 6  <--+||||||
DRAM Byte 7 --> 660 Register Byte 7 --> DRAM Byte 7  <-+|||||||
                        ||                             ||||||||
          ECC Check Byte +==> 660 Register Check Byte--76543210
```

D.11 Optimizing PCI Performance.

Date 9/26/96

Description . . First, there is no DMA mode per se on the 660. The 660 can not be programmed as an independent PCI agent to perform PCI to memory transactions or PCI to PCI peer transactions. The 660 acts as a PCI busmaster only to execute CPU to PCI transfers. Otherwise the 660 looks like a PCI target.

D

1) Optimizing PCI to Memory Performance

1) Don't use ECC memory checking/correcting, or always do 4-byte aligned PCI to memory transfers that are a multiple of 8 bytes in length. These strategies will avoid costly RMW cycles. See the PCI Bus section of the 660 User's Manual.

2) The most efficient PCI to memory burst is a 32-byte burst aligned on a 32-byte boundary. A longer burst (e.g., 48 bytes) will incur 2 snoop cycles on the CPU bus instead of just 1, since it crosses a 60x cache block boundary. Depending on PCI arbiter latency and other system issues, it may more efficient to change the burst length to 32 bytes and do 3, 32B bursts instead of 2, 48B bursts.

3) Set the CAS# pulse width to less than 4 clocks.

4) Do not run the CPU:PCI bus clock ratio at 1:1. Use 2:1.

5) Avoid the system configurations shown in 660UM section 1.11.8, to allow the PCI bus to be parked on the 660. This is generally a good idea, but may not affect the target system. To optimize PCI to memory performance, it may even be a good idea to park the PCI bus on the PCI agent that is expected to be doing the bursting.

The overall quantitative amount of improvement due to each of these items is not readily quantifiable.

2) PCI Throughput Considerations

In the following analysis, the application requires a large number of 32B data sets to be moved from the PCI bus into the CPU, crunched by the CPU, and then moved from the CPU to the PCI bus. We analyzed two different cases. Each case assumes the use of 60ns EDO DRAM with aggressive timing. Neither case includes CPU crunch time, which is assumed to be the same for both cases.

Case 1 – CPU Initiates PCI Transactions

In this case, the CPU reads a 32B block of data from a PCI agent, crunches it, and writes it back to the agent. Then the cycle repeats. This case assumes that the PCI bus is parked on the CPU (on the 660), that the CPU bus is parked on the CPU, and that there are no snoop hits.

Action	Sequence		CPU Clocks
CPU reads 32B from PCI	8, 4B reads		80
CPU writes 32B to PCI	4, 8B writes		36
		Total	**116**

Case 2 – PCI Busmaster Initiates PCI Transactions

In the second case, the PCI busmaster bursts the 32B block into DRAM, the CPU reads the block out of DRAM, crunches it, and then bursts it back into DRAM. Then the PCI busmaster reads it back out again. Then the cycle repeats. This case assumes that the PCI bus is parked on the PCI busmaster.

Action	Sequence	CPU Clocks
PCI writes 32B to DRAM	5,1,1,1,3,1,1,1 PCI clocks	28
CPU reads 32B from DRAM	10,3,3,3 CPU clocks	19
CPU writes 32B to DRAM	5,3,3,3 CPU clocks	14
PCI reads 32B from DRAM	8,1,1,1,1,1,1,1 PCI clocks	30
	Total	**91**

The second case seems to be preferable for several reasons. First, it requires fewer CPU clocks to actually move the data. Second, the semaphore required for the operation may be simpler to implement for case 2 than for case 1. Finally, with case 2, the CPU bus is generally free to other tasks while the CPU is reading DRAM, crunching the data, and writing the DRAM. Likewise, except for the snoop cycle which is usually pipelined and therefore transparent, the CPU bus is generally free to other tasks while the PCI busmaster is accessing DRAM.

Please note that these are expected typical performance numbers which are not guaranteed to occur in every system.

D.12 Memory Access

Date 10/30/97

Description . . If the 660 bridge gets a PCI device retry on the PCI bus, will another master be able to access the memory through the 660 bridge, or will the 660 generate a retry to the master until its own requested data is complete?

If the 660 bridge gets a PCI device retry on the PCI bus, will another master be able to access the memory through the 660 bridge, or will the 660 generate a retry to the master until its own requested data is complete?

Case 1 = 1B to 4B IO Read/Write or Memory Read

The 603 initiates a transfer to an address mapped to the PCI bus, so the 660 arbitrates for the PCI bus and then initiates a PCI (memory read or IO read/write) transaction. The target claims the PCI cycle, and then asserts STOP# without TRDY# to signal a target retry. In this case, the 660 will terminate the PCI cycle in compliance to the PCI specification and signal ARTRY# to the 603. The transaction is finished as far as the 660 is concerned.

In this case, the PCI busmaster (once granted the PCI bus) has full access to system memory. The 660 gets off of the PCI bus for 2 PCI clocks, which allows the bus protocols to operate. The 660 will allow the PCI busmaster to access DRAM. No problem.

Case 2 = 1B to 4B Memory Write

In the write case, the CPU transfer is posted to the 660, and the CPU address tenure ends. The 660 arbitrates for the PCI bus and then initiates

D

a PCI memory write transaction. The target claims the PCI cycle, and then asserts STOP# without TRDY# to signal a target retry. In this case, the 660 will terminate the PCI cycle in compliance with the PCI specification, and then retry the PCI transaction.

In this case, the 660 retries the PCI transaction on behalf of the CPU, but it also gets off of the PCI bus for two PCI clocks between successive retries. In this case also, if another PCI busmaster initiates a transaction to system memory, the 660 will not honor it.

Case 1 and case 2

In both Case 1 and Case 2, the address of the PCI busmaster transaction is broadcast onto the CPU bus. If the CPU signals a snoop hit, the the 660 retries the PCI busmaster, and the CPU pushes out the block. Then the PCI busmaster can try again.

D.13 DRAM Timing

Date 10/30/97

Some DRAM data sheets specify MAximum RCD $\leq$ 40 ns. In 1:1 mode, the 660 can appear to violate this specification (see Section 5.2.1.2); however, the RCD specification is only for reference. The 660 does not violate the RCD specification's underlying RCD.

Contacts

USA and Canada:
IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6531
Tel: (919) 543–5701
Fax: (919) 543–7575
ppcsupp@raleigh.ibm.com
http://www.chips.ibm.com