PowerPC

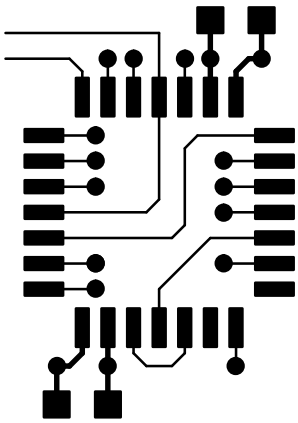
Preliminary, IBM Internal Use Only

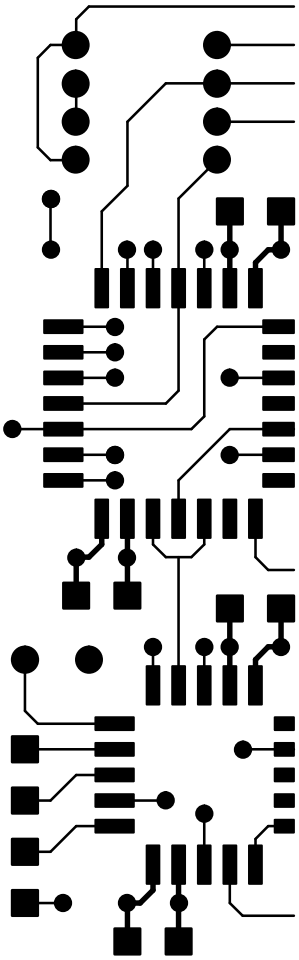
100 MHz PPC 603e Multi-Chip Module User's Manual

Release 0.1

This document provides a detailed technical description of the PPC 603e MCM. It is intended as a first source of information for both hardware and software designers. Where appropriate, other documents are referenced.

Document Number:
G5220297-00
February, 1997





IBM®

Contacts

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6531
Tel: (800) PowerPC

http://www.chips.ibm.com
http://www.ibm.com
ftp://ftp.austin.ibm.com/pub/PPC_support

Dale Elson, Applications Engineer
IBM PowerPC Embedded Processor Solutions
elson@austin.ibm.com

ESD Warning

The planar and MCM contain CMOS devices which are very susceptible to ElectroStatic Discharge (ESD). DO NOT remove them from the antistatic bags until you have connected yourself to an acceptable ESD grounding strap. Work in a static free environment and be sure any person or equipment coming into contact with the cards do not have a static charge. The cards are particularly susceptible until they are placed in a properly designed enclosure. Bench work should be done by persons connected to ESD grounding straps.

IBM 100 MHz PPC 603e MCM AGREEMENT

BEFORE READING THE REST OF THE DOCUMENT, YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS. OPENING THE PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED TO YOUR IBM SALES OFFICE.

International Business Machines Corporation ("IBM") agrees to provide you an 100 MHz PPC 603e MCM in return for your promise to use reasonable efforts to develop a system based on the technology in the MCM. The MCM contains documentation and software listed below:

Documentation
IBM27-82660 PowerPC to PCI Bridge and Memory Controller User's Manual
PowerPC 603e RISC Microprocessor Hardware Specification
PowerPC 603e RISC Microprocessor Technical Summary
IBM 64K x 18 Burst SRAM (IBM041814PQK) Data Sheet
Texas Instruments 3.3V ABT 16-Bit Bus Transceivers with 3-State Output (SN54LVTH16245A, SN54LVTH16245A) Data Sheet
Integrated Device Technology BiCMOS StaticRAM 240K (16K x 15-Bit) Cache-Tag RAM for PowerPC and RISC Processors (IDT71216) Data Sheet
Motorola Low Voltage PLL Clock Driver (MPC970/D) Data Sheet

LICENSE TO SOFTWARE

The software is licensed not sold. IBM, or the applicable IBM country organization, grants you a license for the software only in the country where you received the software. Title to the physical software and documentation (not the information contained in such documentation) transfers to you upon your acceptance of these terms and conditions. The term "software" means the original and all whole or partial copies of it, including modified copies or portions merged into other programs. IBM retains title to the software. IBM owns, or has licensed from the owner, copyrights to the software provided under this agreement. The terms of this Agreement apply to all of the hardware, software and documentation provided to you as part of the 100 MHz PPC 603e MCM.

With regard to the software provided hereunder, it is understood and agreed that you intend to use the software solely for the purpose of designing PowerPC compatible products, testing your designs, and making your own independent determination of whether you wish to eventually manufacture PowerPC compatible products commercially. In accordance with this understanding, IBM hereby grants you the rights to: a) use, run, and copy the software, but only make such number of copies and run on such number of machines as are reasonably necessary for the purpose of designing PowerPC compatible products and testing such designs; and b) copy the software for the purpose of making one archival or backup copy.

With regard to any copy made in accordance with the foregoing license, you must reproduce any copyright notice appearing thereon. With regard to the software provided hereunder, you may not: a) use, copy, modify or merge the software, except as provided in this license; b) reverse assemble or reverse compile it; or c) sell, sublicense, rent. lease, assign or otherwise transfer it. In the event that you no longer wish to use the software, you will return it to IBM.

LICENSE TO DESIGN DOCUMENTATION

With regard to the design documentation provided hereunder, it is understood that you intend to use such documentation solely for the purpose of designing your own PowerPC compatible products, testing your designs, and making your own independent determination of whether you wish to eventually manufacture PowerPC compatible products commercially. In accordance with this understanding, IBM hereby grants you the right to: a) use the design documentation for the purpose of designing PowerPC compatible products and testing such designs; b) make derivative works of the design documentation for the purpose of designing PowerPC compatible products, and testing such designs; and c) make copies of the design documentation and any such derivative works, but only such numbers as are reasonably necessary for designing PowerPC compatible products and testing such designs.

With regard to any copy made in accordance with the forgoing license, you must reproduce any copyright notice appearing thereon. With regard to the design documentation provided hereunder, you may not: a) use, copy, modify, or merge the design documentation as provided in this license; or b) sell, sublicense, rent, lease, assign, or otherwise transfer it.

In the event you no longer wish to use the design documentation or any derivative versions thereof, you must return them to IBM.

DISCLAIMER OF WARRANTY

IBM does not represent or warrant that the 100 MHz PPC 603e MCM (which may contain prototype items): a) meets any particular requirements; b) operates uninterrupted; c) is error free; or d) is non–infringing of any patent, copyright, or other intellectual property right of any third party. IBM makes no representation or warranty regarding the performance or compatibility that may be obtained from the use of the MCM or that the MCM is adequate for any use. The MCM may contain errors and may not provide the level of completeness, functionality, support, performance, reliability, or ease of use available with other products, whether or not similar to the MCM. IBM does not represent or warrant that errors or other defects will be identified or corrected.

THE 100 MHz PPC 603e MCM IS PROVIDED "AS IS" WITH ALL FAULTS, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE MCM IS WITH YOU.

Some jurisdictions do not allow exclusion of implied warranties, so the above exclusions may not apply to you.

LIMITATION OF REMEDIES

IBM's entire cumulative liability and your exclusive remedy for damages for all causes, claims or actions wherever and whenever asserted relating in any way to the subject matter of this agreement including the contents of the 100 MHz PPC 603e MCM and any components thereof, is limited to twenty five thousand dollars ($25,000.00) or its equivalent in your local currency and is without regard to the number of items in the MCM that caused the damage. This limitation will apply, except as otherwise stated in this Section, regardless of the form of the action, including negligence. This limitation will not apply to claims by you for bodily injury or damages to real property or tangible personal property. In no event will IBM be liable for any lost profits, lost savings, or any incidental damages or economic consequential damages, even if IBM has been advised of the possibility of such damages, or for any damages caused by your failure to perform your responsibilities. In addition, IBM will not be liable for any damages claimed by you based on any third party claim. Some jurisdictions do not allow these limitations or exclusions, so they may not apply to you.

RISK OF LOSS

You are responsible for all risk of loss or damage to the 100 MHz PPC 603e MCM upon its delivery to you.

IBM TRADEMARKS AND TRADE NAMES

This Agreement does not give you any rights to use any of IBM's trade names or trademarks. You agree that should IBM determine that any of your advertising, promotional, or other materials are inaccurate or misleading with respect to IBM trademarks or trade names, that you will, upon written notice from IBM, change or correct such materials at your expense.

NO IMPLIED LICENSE TO IBM INTELLECTUAL PROPERTY

Notwithstanding the fact that IBM is hereby providing design information for your convenience, you expressly understand and agree that, except for the rights granted under sections 1 and 2 above, no right or license of any type is granted, expressly or impliedly, under any patents, copyrights, trade secrets, trademarks, or other intellectual property rights of IBM. Moreover, you understand and agree that in the event you wish to be granted any license beyond the scope of the expressly stated herein, you will contact IBM's Intellectual Property Licensing and Services Office (currently located at 500 Columbus Avenue, Thornwood, N.Y.), or such other IBM offices responsible for the licensing of IBM intellectual property, when you seek the license.

YOUR ASSUMPTION OF RISK

You shall be solely responsible for your success in designing, developing, manufacturing, distributing, and marketing any product(s), or portion(s), where use of all or any part of the 100 MHz PPC 603e MCM is involved. You are solely responsible for any claims, warranties, representations, indemnities and liabilities you undertake with your customers, distributors, resellers or others, concerning any product(s) or portion(s) of product(s) where use of all or any part of the MCM is involved. You assume the risk that IBM may introduce other Reference Design that are somehow better than the MCM which is the subject of this Agreement. Furthermore, you accept sole responsibility for your decision to select and use the MCM; for attainment or non-attainment of any schedule, performance, cost, reliability, maintainability, quality, manufacturability or the like, requirements, or goals, self–imposed by you or accepted by you from others, concerning any product(s) or portion(s) of product(s), or for any delays, costs, penalties, charges, damages, expenses, claims or the like, resulting from such non-attainment, where use of all or any part of the MCM is involved.

GENERAL

In the event there is a conflict between the terms of this Agreement and the terms printed or stamped on any item or any ambiguities with respect thereto, including documentation, contained in the 100 MHz PPC 603e MCM, the terms of this Agreement control to the extent IBM is afforded greater protection thereby. IBM may terminate this Agreement if you fail to comply with the terms and conditions of this Agreement. Upon termination of this Agreement, you must destroy all copies of the software and documentation. You are responsible for payment of any taxes, including personal property taxes, resulting from this Agreement. Neither party may bring an action hereunder, regardless of form, more than one (1) year after the cause of the action arose. If you acquired the MCM in the United States, this Agreement is governed by the laws of the State of New York. In the event of litigations, trial shall be in New York without a jury. If you acquired the MCM in Canada, this Agreement is governed by the laws of the Province of Ontario; otherwise, this Agreement is governed by the laws of the country in which you acquired the MCM. All obligations and duties which, by their nature, survive termination or expiration of this Agreement, shall remain in effect beyond termination or expiration of this Agreement, and shall bind IBM, you and your successors and assigns. If any section or paragraph of this Agreement is found by competent authority to be invalid, illegal or unenforceable in any respect for any reason, the validity, legality, and enforceability of any such section or paragraph in every other respect, and the remainder of this Agreement, shall continue in effect so long as it still expresses the intent of the parties. If the intent of the parties cannot be preserved, the parties will attempt to renegotiate this Agreement and failing renegotiation, this Agreement will then be terminated. The headings in this Agreement shall not affect the meaning or interpretation of this Agreement in any way. No failure by IBM in exercising any right, power or remedy under this Agreement shall serve as a waiver of any such right, power or remedy. Neither this Agreement nor any activities hereunder will impair any right of IBM to develop, manufacture, use or market, directly or indirectly, alone or with others, any products or services competitive with those offered or to be offered by you; nor will this Agreement or any activities hereunder require IBM to disclose any business planning information to you. You agree to comply with all applicable government laws and regulations. Any changes to this Agreement must be in writing and signed by the parties.

Table of Contents

Figures

Tables

About This Book

This document is designed for engineers and system designers who are interested in implementing PowerPC systems that use the 100 MHz PPC 603e MCM. The material requires a detailed understanding of computer systems at the hardware and software level.

Power management is beyond the scope of this document. This document was written by Dale Elson.

Reference Material:

Understanding of the relevant areas of the following documents is required for a good understanding of the 100 MHz PPC 603e MCM:

- *PowerPC 603e RISC Microprocessor User's Manual*, IBM document MPR603EUM-01
- *PowerPC 603e Hardware Specification*, IBM document MPR603EHS-01
- *PowerPC 603e Technical Summary*, IBM document MPR603TSU-04
- *IBM27-82660 PowerPC to PCI Bridge User's Manual*, IBM document number MPR660UMU–01
- *PCI Local Bus Specification*, Revision 2.1, available from the PCI SIG
- *PowerPC Reference Platform Specification,* Version 1.1, IBM document MPRPRPPKG
- *The Power PC Architecture,* second edition, Morgan Kaufmann Publishers (800) 745–7323, IBM document MPRPPCARC–02
- *Intel 82378ZB System I/O (SIO) Data Book*, Intel order number 290473-004.

The following documents are useful as sources of tutorial and supplementary information about the MCM.

- *PowerPC System Architecture*, Tom Shanley, Mindshare Press (800) 420-2677.
- *IBM27-82650 PowerPC to PCI Bridge User's Manual, IBM document number MPR650UMU–01*

Document Conventions:

The terms 660 and 660 bridge refer to the IBM27-82660.

The terms 660 UM and 660 User's Manual refer to the current version of the *IBM27-82660 PowerPC to PCI Bridge User's Manual.*

The terms 603e UM and 603e User's Manual refer to the *IBM PowerPC 603e RISC Microprocessor User's Manual.*

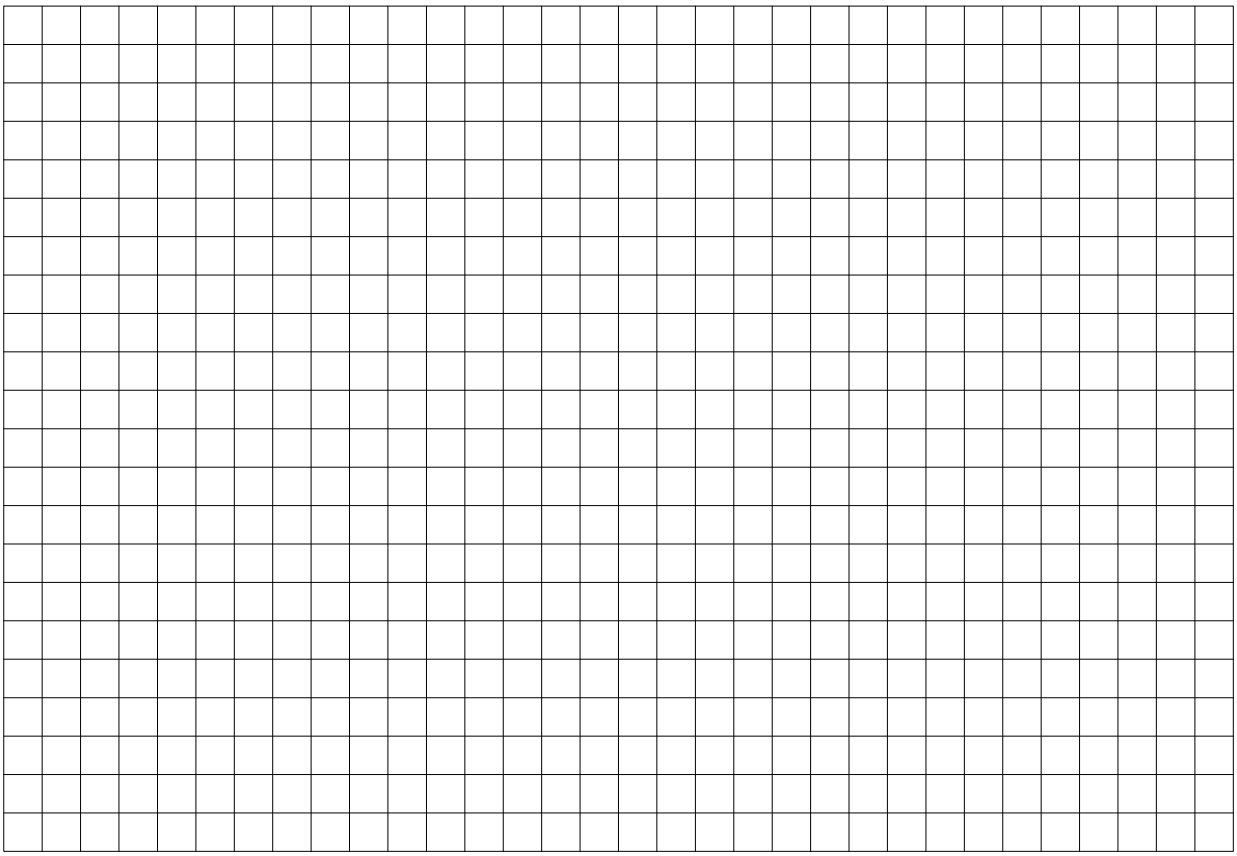
Kilobytes, megabytes, and gigabytes are indicated by a single capital letter after the numeric value. For example, 4K means 4 kilobytes, 8M means 8 megabytes, and 4G means 4 gigabytes.

The terms DIMM and SIMM are 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. Binary values are identified (where not clear from context) with a lower-case letter b at the end of the value.

In identifying ranges of values *from* and *to* are used whenever possible. 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.



Section 1
Introduction

This document provides a detailed technical description of the 100 MHz PPC 603e Multi-Chip Module (MCM), and is intended to be used by hardware, software, test, and simulation engineers as a first source of information. Software developers should read through the entire document because information relevant to their tasks may be located in hardware sections.

1.1 The MCM

The MCM consists of the CPU, L2, PCI Bridge, Memory Controller and master clock elements of a PowerPC system. The chips that provide this function are all packaged on a single ceramic substrate.

1.2 100 MHz PPC 603e MCM

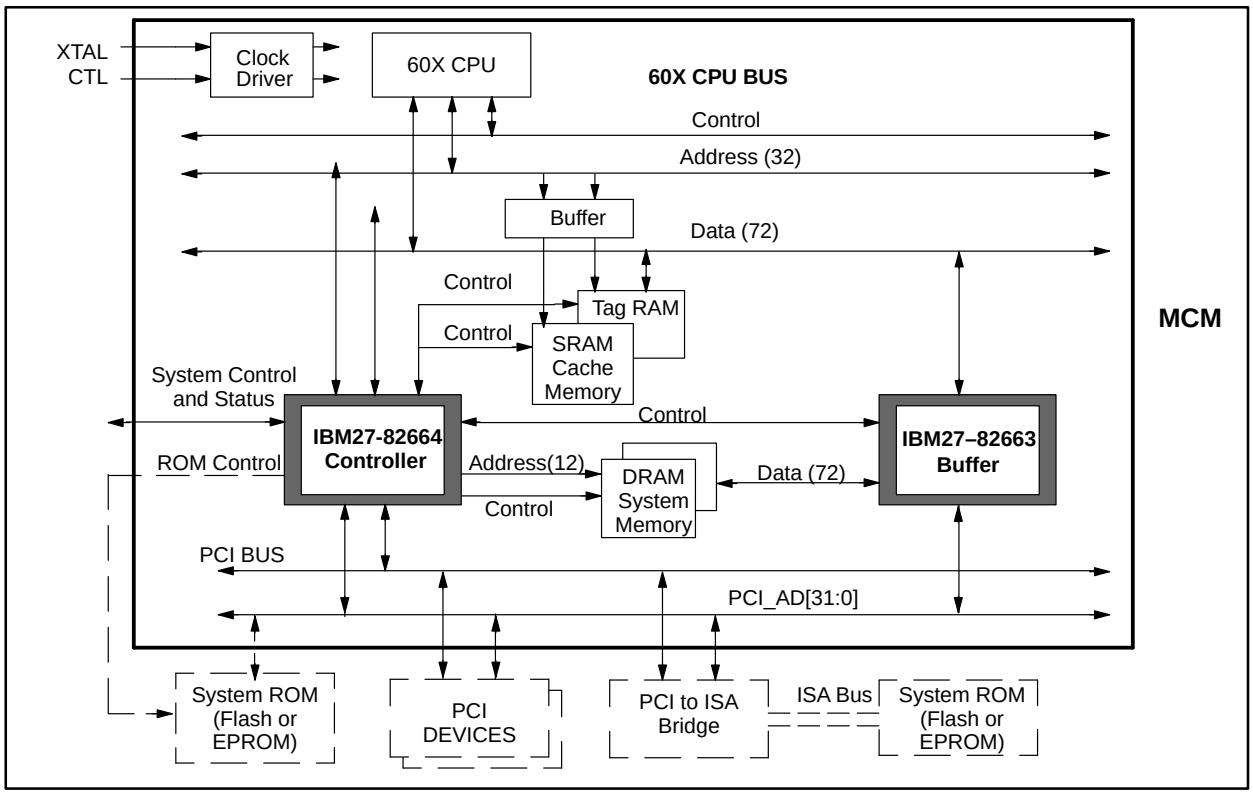


Figure 1-1. MCM Block Diagram

The MCM (see Figure 1-1) is compliant with the PowerPC Reference Platform Specification Version 1.1, and the PCI Local Bus Specification, revision 2.0/2.1 (see Section 5.5) for host bridges.

The MCM uses the PowerPC 603e™ RISC microprocessor. The IBM27-82660 Bridge chipset (660 Bridge or 660) interfaces the CPU to the PCI bus and DRAM memory, and provides L2 cache control. The tagRAM and 512K SRAM components are also included on the MCM. The MCM also includes a master clock generator. For a list of the specific dice packaged within the MCM, see Table 1-1.

Table 1-1. MCM Dice

Die	Part Number	Vendor
PowerPC 603e Processor		IBM
663 Buffer		IBM
664 Controller		IBM
Clock Driver	MPC970	Motorola
Tag RAM	IDT1216	Integrated Device Technology
Buffer	SCBS143F	Texas Instruments
Synchronous Burst SRAM		IBM

1.2.1 The CPU

The MCM uses a PowerPC 603e (PID6–603e) at CPU bus speeds up to 66MHz. See the PowerPC 603e Technical Summary, Users Manual, and Hardware Specifications for more information. The MCM:

- Runs at 3:2 CPU_core:CPU_bus speed ratio up to 99:66 MHz. Consult your IBM representative for available choices of CPU and operating frequency.
- Supports one level of address bus pipelining. Most data writes are posted.
- Reports precise exceptions via TEA#, and reports imprecise exceptions via MCP#.
- Operates the 603e in 64-bit data bus mode.
- Supports bi-endian operation.
- Supplies an ESP connector for RISCWatch debugging and monitor systems.
- Runs in DRTRY# mode.

1.2.2 L2 Cache

The MCM contains a complete L2. The controller is located inside the 660 Bridge. The L2:

- Uses 512K SRAM and a 16K x 15 synchronous tagRAM.
- Is unified, write-thru, direct-mapped, look-aside.
- Caches system memory from 0 to 1G.
- Maintains coherence for 32-byte blocks, the PowerPC 60X coherence unit.
- Supplies data to the CPU on burst read hits and snarfs the data on CPU burst read/ write misses. The L2 is not updated during a PCI transaction.
- Ignores CPU single-beat reads, and invalidates on CPU bus single-beat write hits.
- Snoops PCI to memory transactions, invalidates on PCI write hits. The L2 does not supply data to the PCI on read hits.

1.2.3 System Memory Interface

The MCM memory controller, located in the 660, provides the system DRAM memory interface as required by the PowerPC Architecture™. System memory can be accessed from both the CPU bus and the PCI bus. The system memory interface:

- Has a 72-bit wide data path – 64 data bits and 8 bits of optional ECC or parity data
- Supports page mode and EDO (hyper-page mode) DRAM
- Supplies 8 RAS# outputs, 8 CAS# outputs, and 2 WE# outputs, for up to 8 SIMM banks
- Supports 8-byte, 168-pin and 4-byte, 72-pin SIMMs for up to 1G of DRAM
- Allows mixed use of different size SIMMs, including mixed 4-byte and 8-byte SIMMs
- Includes full refresh support, including refresh counter and programmable timer
- Implements burst-mode memory address generation for both CPU and PCI bursts
- Implements both little-endian and big-endian addressing and byte swapping modes
- Provides row and column address multiplexing for 10x10 thru 12x12 RxC DRAMs

1.2.4 The PCI Interface

The MCM allows CPU to PCI access and PCI busmaster to system memory access (with snooping), and handles all PCI related system memory and cache coherency issues.

The MCM supplies a 32-bit PCI host bridge interface that is:

- Compliant with the PCI Specification, revisions 2.0 and 2.1 (see Section 5.5)
- 3.3v and 5v signalling environment compliant.

- Operates up to 33MHz, at 1/2 of the CPU bus frequency.
- Supports a tertiary IO bus bridge, including support for ISA masters.
- Allows system memory block locking by PCI busmasters
- Supports type 0 and type 1 PCI configuration cycles

1.2.5 System Clocks

The MCM uses a Motorola™ MPC970 PLL clock generator to provide the clocks required by the MCM components, and six clocks for system use. A 16.5 MHz quartz crystal (seed clock) provided by the user is used as an input to the clock generator. It then produces the system and PCI clocks.

1.2.6 MCM Performance

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

1.3 MCM Technology Overview

The IBM Multi-Chip Module (MCM) is shown in Figure 1-2. The MCM consists of a 45mm x 45mm cofired dark ceramic substrate upon which are mounted eleven unpackaged semiconductor dice (chips), capacitors, and resistors. The top of the substrate and the chips are completely covered by an aluminum cap, which is non-hermetically sealed to the substrate. A thermally conductive grease fills the area between the cap and the chips. The MCM is electrically connected to the circuit board (planar) by solder columns.

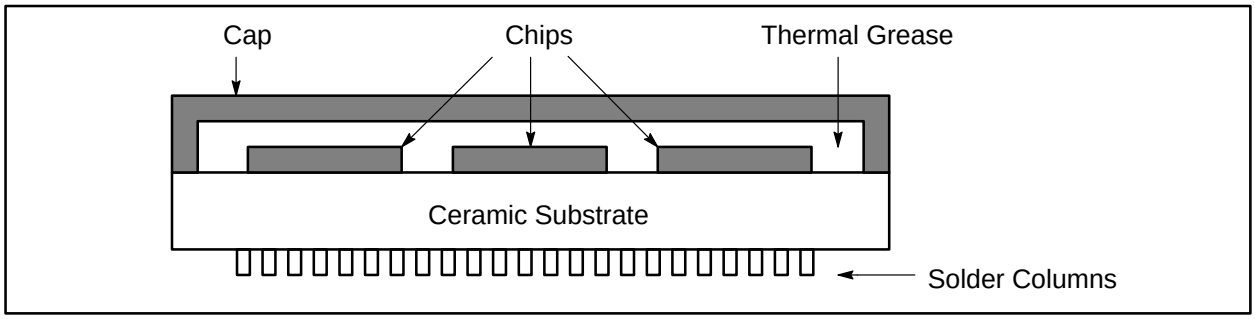


Figure 1-2. MCM C

1.3.1 Chip Attachment to MCM

All die on the MCM are attached to the ceramic substrate using IBM's C4-based flip chip attachment technology. In this technology, C4 solder balls provide an interconnection between the die and the substrate. For those vendor die originally designed for wire bond base interconnection, additional processing steps were applied to allow the C4 interconnection. Figure 1-3 shows a typical wire–bond die. In wire bond technology, the wafer is sliced into individual dies, each die is attached to a wire frame, and (for each chip pad) one end of a wire is bonded to the chip pad, and the other end is bonded to the package pin.

The MCM uses the "flip–chip" process. An additional metalization layer is added to the chip, the chip is "flipped" over, and C4 balls are used to connect the chip pads to the MCM substrate pads.

Unpackaged die have a much smaller footprint than packaged die. This reduces the overall size of the MCM circuitry and allows the MCM a much smaller footprint than would be required by the same circuitry implemented on a planar in discrete packages.

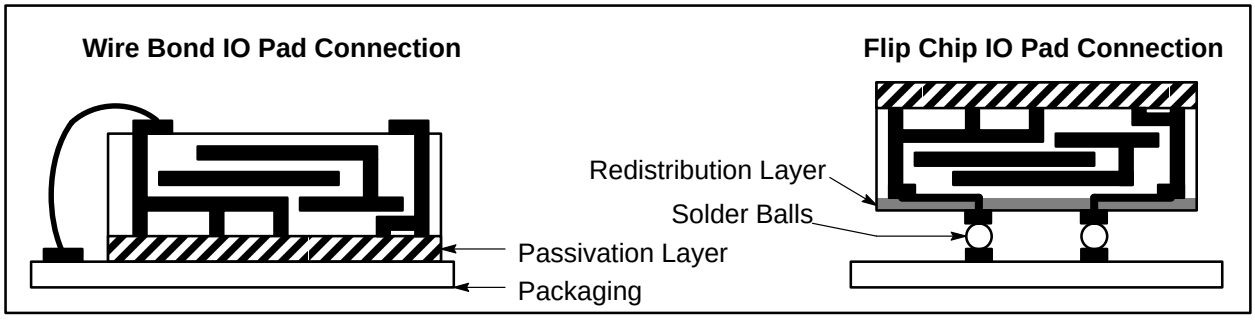


Figure 1-3. Chip Wire Bond and Flip Chip Connection

Figure 1-4 shows chips mounted to the MCM substrate. Figure 1-4 also illustrates the internal structure of the MCM substrate. The MCM acts as the circuit board between the various chips. Structures analogous to printed circuit board traces and vias form the electrical interconnections between the various MCM chips. The trace lengths on the MCM are much shorter than are achievable on a planar. This means that signal propagation (flight times) between the devices are minimized, which can allow faster operating frequencies than can be achieved on a planar. The impedance of the various nets can be more precisely controlled, which contributes to high signal quality and low noise.

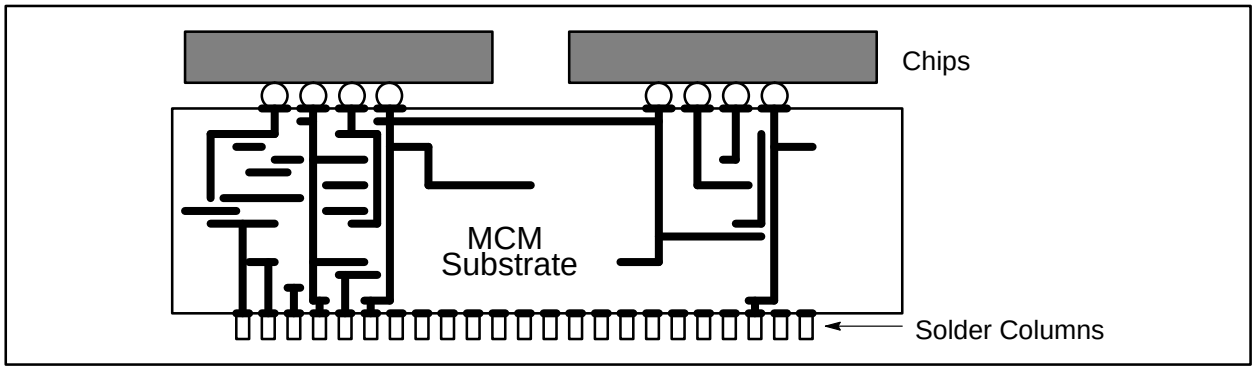


Figure 1-4. MCM Flip Chips and Internal Wiring

1.3.2 Substrate Attachment to the Planar

Figure 1-5 shows how solder columns are used to connect the MCM to the planar. Each column is attached to the MCM by an eutectic solder joint. The solder at this point melts at a higher temperature than the solder used for the column itself. The MCM is then soldered to the planar, again using an eutectic solder. The solder column process is robust, and offers a lower impedance. It is also one of the higher density interconnection methods.

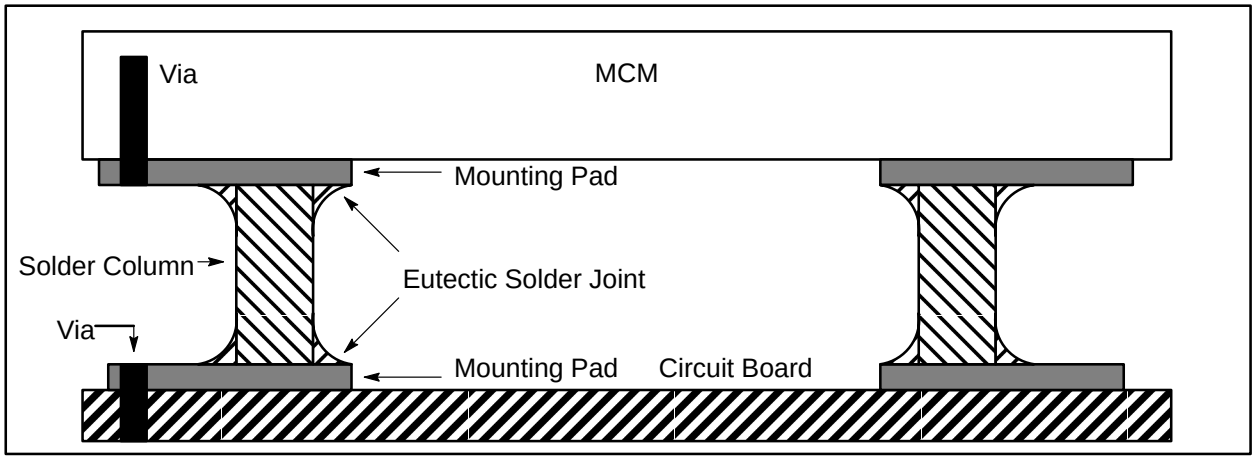


Figure 1-5. MCM Solder Columns

For more information on solder columns, assembly, and rework, see "*Ceramic Ball and Column Grid Array Surface Mount Assembly and Rework*", IBM document number APD–SBSC–101.0.

1.4 Incorporating the MCM into a System Design

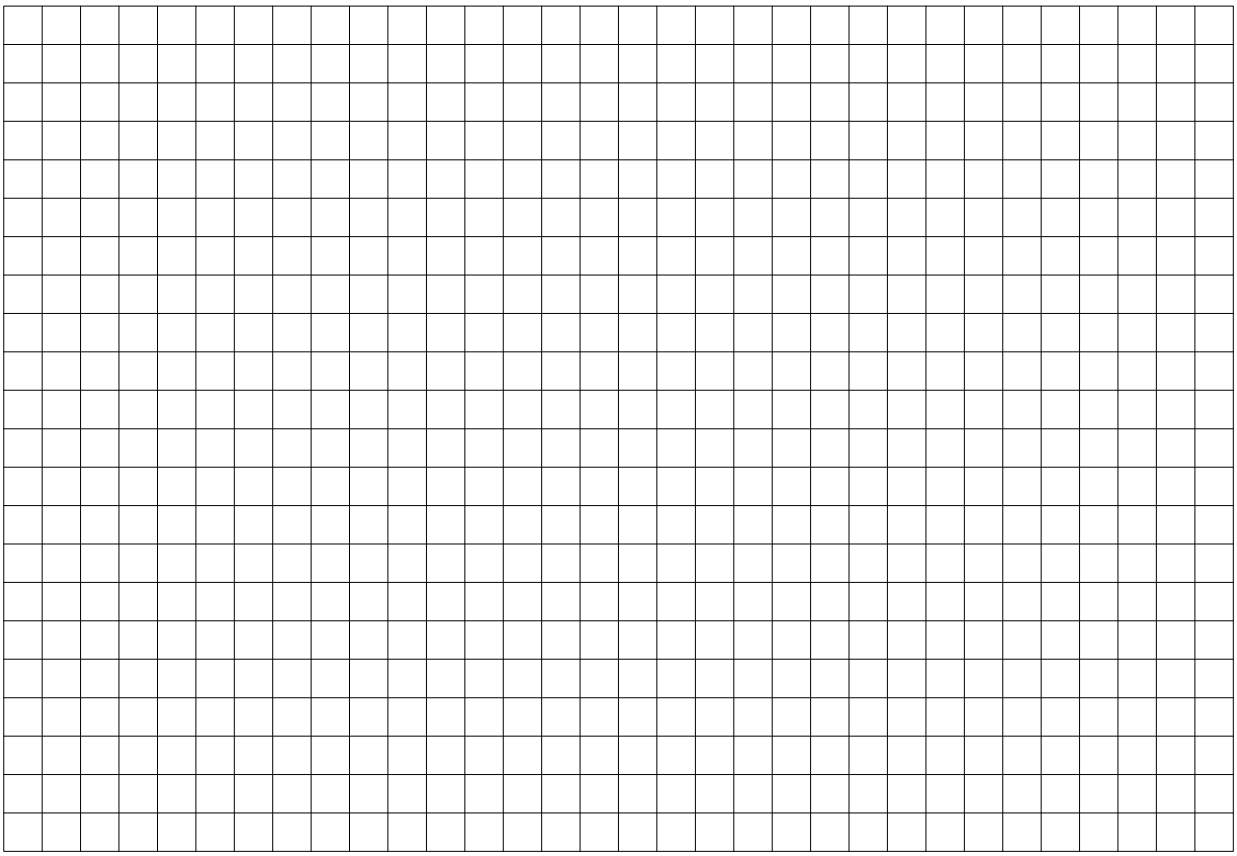
The MCM captures, in one ceramic package, all key components of a PowerPC system design that resides on a CPU's local bus. The MCM's primary ports of interface to the rest of a system are the PCI bus and the system memory (DRAM) port. Other signals that connect the MCM to a system fall into the following groups:

- Processor PLL configuration
- MPC 970 configuration
- Signal nets broken at the MCM level for test purposes and need to be reconnected on the planar
- RISCWatch interface
- 60X bus signals requiring pullup/pulldown connections at the planar
- L2 cache configuration and presence detect
- Clock nets broken at the MCM level to allow net length matching with planar resident clock nets
- ROM control signals
- Local bus signals used for control functions by the planar.

Understanding all of these groups is vital to the use of the MCM. Pay particular attention to those signals broken at the MCM level and which are reconnected at the planar. Those nets are associated with MCM I/O pairs, the names of which are prefaced by the letter "X."

When applying the MCM to a system design, it is recommended that the designer review the schematics for both the MCM (see Section 13) and the example planar (see Appendix F). Although the example planar is a specific design implementation of the MCM, most MCM interface signals are used in a manner that would be common to any PowerPC system design. A review of the planar schematics and MCM and planar component data sheets (see Appendix G) will quickly answer most questions concerning how to interface with the MCM I/O.

Note that it is not intended for the user to interface with all of the MCM I/O. Section 2 contains I/O listings that distinguish between signals intended for application use and those broken out only for manufacturing purposes.



Section 2
Signal Descriptions

This section describes the connectivity of the 100 MHz PPC 603e MCM. Tables are used extensively for this purpose (see Table 2-1 and the following paragraphs for an explanation).

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 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.

Table 2-1. Example Signal Table

Signal Name	MCM	Nodes		Description
AACK#	R08	R1.A02 U1.28 U2.109	pu I/O I/O	CPU bus address acknowledge. The CPU bus target (660 or external CPU bus target) asserts AACK# to signal the end of the current address tenure. 660 also asserts AACK# for one clock to signal the end of a broadcast snoop cycle. AACK# is an input to the 664 when a CPU bus target claims the current transaction by means of L2_CLAIM#. AACK# is always an input to the CPU.

The **Signal Name** column contains the name of the signal. In the Example Signal Table (Table 2-1), the AACK# signal is shown.

The **MCM** column shows which pin(s) (solder columns) of the MCM connect to the signal. The AACK# net is connected to column R08 of the MCM.

The **Nodes** columns show the internal MCM devices and pins to which the signal connects. The left column shows the device and pin number, in device.pin format. The right column shows whether the signal is an input or output of the device. An **I** indicates that the signal is an input to the device. An **O** indicates that the signal is an output from that device. A **pu** indicates that the resistor to which the signal is connected is a 10k pullup resistor.

AACK# connects to R1 pin A02, which is a 10k pullup resistor. AACK# also connects to U1 pin 28 and U2 pin 109, both of which are I/Os.

2.1 Pin Descriptions

The following sections contain information about the connectivity and function of the MCM pins. See the 660 User's Manual (660 UM), the 603e User's Manual (603e UM), and the SRAM, Tag, and buffer data sheets for more information.

Note that the MCM I/O pins make a connection to every net on the MCM. This was only done to aid in the manufacturing test of the MCM. Of the 1089 I/O pins that the MCM provides, only 296 signals are needed by the user to interface with the MCM. These signals are listed in Table 2-2.

Table 2-2. User Supplied Signals

Type	Signal
PCI	PCI_AD0:9
PCI	PCI_PAR
PCI	PCI_AD10:31
PCI	PCI_IRDY#
PCI	PCI_LOCK#
PCI	PCI_PERR#
PCI	PCI_SERR#
PCI	PCI_STOP#
PCI	PCI_TRDY#
PCI	PCI_AD_OE#
PCI	PCI_C/BE0:3#
PCI	PCI_CLK_IN
PCI	PCI_DEVSEL#
PCI	PCI_EXT_SEL
PCI	PCI_OL_OPEN
PCI	PCI_OUT_SEL
PCI	–664_PCI_REQ#
PCI	PCI_FRAME_664#
MEMORY	MA0:11
MEMORY	MDP0:10
MEMORY	MD10:63
MEMORY	MCE0:7
MEMORY	MRE0:7
MEMORY	MWE0:1
MISC. FUNCTION	TEA#
MISC. FUNCTION	DRTRY#
MISC. FUNCTION	ROM_OE#
MISC. FUNCTION	ROM_WE#
MISC. FUNCTION	INT_60X#
MISC. FUNCTION	MCP_60X#
MISC. FUNCTION	SRAM_OE#
MISC. FUNCTION	PLL_CFG0:3
MISC. FUNCTION	ROM_LOAD
MISC. FUNCTION	60X_AVDD
MISC. FUNCTION	QACK_60X
MISC. FUNCTION	QREQ_60X
MISC. FUNCTION	X_INT_60X#
MISC. FUNCTION	X_MCP_60X#
MISC. FUNCTION	X_SRAM_OE#
MISC. FUNCTION	INT_TO_664
MISC. FUNCTION	X_PCLK_60X
MISC. FUNCTION	X_TAG_BCLK
MISC. FUNCTION	SRESET_60X#
MISC. FUNCTION	TAG_ADDR_13
MISC. FUNCTION	TAG_DATA_11
MISC. FUNCTION	IGN_PCI_AD31
MISC. FUNCTION	X_SRAM_BCLK0:3
MISC. FUNCTION	X_663_CPU_CLK
tMISC. FUNCTION	X_664_CPU_CLK

Type	Signal
.MISC. FUNCTION	X_CPU_RDL_OPEN
MISC. FUNCTION	NMI_FROM_ISABRDG
MISC. FUNCTION	POWER_GOOD/RESET
MISC. FUNCTION	HRESET#
RW	TCK
RW	TDI
RW	TDO
RW	TMS#
RW	CKSTP_OUT#
RW_P_UP	TRST#
P_UP	TT0:3
P_UP	SHD#
P_UP	SMI#
P_UP	TAG_CS2
P_UP	TT2_664
P_UP	DBG_60X#
P_UP	L2_CLAIM#
P_UP	663_TEST#
P_UP	664_TEST#
P_UP	TAG_MATCH
P_UP	BG_MASTER#
P_UP	SRAM_ADSP#
P_UP	TAG_PWRDN#
P_UP	MWS_P2MRXS
P_UP	664_STOP_CLK_EN#
P_DOWN	TT4
P_DOWN	GBL#
P_DOWN	SRAM_CS#
P_DOWN	TAG_CS1#
P_DOWN	OE_245_B
P_DOWN	DIR_245_A
P_DOWN	DIR_245_B
P_DOWN	TAG_SFUNC
P_DOWN	CRS_C2PWXS
P_DOWN	663_MIO_TEST
P_DOWN	664_MIO_TEST
CLK	XTAL1:2
CLK	FRZ_CLK
CLK	FRZ_DATA
CLK	PCLK_60X
CLK	TAG_BCLK
CLK	CLK_EXT_FB
CLK	CLK_FB_SEL
CLK	CLK_PLL_EN
CLK	SRAM_BCLK0:3
CLK	CLK_COM_FRZ
CLK	CLK_REF_SEL
CLK	CLK_TTL_CLK
CLK	CLK_VCO_SEL

Type	Signal
CLK	663_CPU_CLK
CLK	664_CPU_CLK
CLK	CLK_BCLK_DIV0:1
CLK	CLK_FRZ_STROBE
CLK	CLK_MPC601_CLKS
CLK	CLK_MR/TRISTATE

Type	Signal
CLK	664_PCI_CLK
CLK	CLK_PCI_DIV0:1
CLK	USER_PCICLK0:6
DAISEY	DAISY01:20
PWR	VSS
PWR	VDD1:3

2.1.1 Clock Subsystem

Table 2-3. Clock Signals

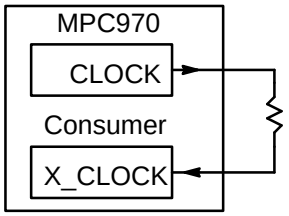
Signal Name	MCM	Nodes[1]		Description (See MPC970 data sheet for more information.)	
663_CPU_CLK	G04	U4.38	O	MPC970 2x_PCLK output.	These IOs are the (nominally 66MHz) CPU bus clocks. They are intended to be used in pairs, where the CLOCK output is connected on the planar to the X_CLOCK input. The MPC970 drives the CLOCK outputs, and the normal consumer of that clock is connected to the X_CLOCK input. A series termination resistor and/or EMC components can be used as determined by the application. 
X_663_CPU_CLK	F05	U3.157	I	The CPU bus clock input of the 663.	
664_CPU_CLK	G30	U4.42	O	MPC970 BCLK0 output.	
X_664_CPU_CLK	H29	U2.121	I	The CPU bus clock input of the 664.	
PCLK_60X	E18	U4.34	O	MPC970 PCLK_EN output.	
X_PCLK_60X	D19	U1.212	I	The SYSCLK input of the CPU.	
SRAM_BCLK0	W30	U4.44	O	MPC970 BCLK1 output.	
X_SRAM_BCLK0	Y29	U7.51	I	The CPU bus clock input of SRAM bank 0.	
SRAM_BCLK1	AE30	U4.46	O	MPC970 BCLK2 output.	
X_SRAM_BCLK1	AF29	U8.51	I	The CPU bus clock input of SRAM bank 1.	
SRAM_BCLK2	AG04	U4.48	O	MPC970 BCLK3 output.	
X_SRAM_BCLK2	AF05	U9.51	I	The CPU bus clock input of SRAM bank 2.	
SRAM_BCLK3	AA04	U4.50	O	MPC970 BCLK4 output.	
X_SRAM_BCLK3	Y05	U10.51	I	The CPU bus clock input of SRAM bank 3.	
TAG_BCLK	N30	U4.36	O	MPC970 BCLK_EN output.	
X_TAG_BCLK	P29	U5.69	I	The CLK input of the TAG.	
664_PCI_CLK	J30	U4.18	O	MPC970 PCI_CLK1 output. Normally connected to PCI_CLK_IN. This clock is intended to be run at one half of the CPU bus frequency. See the PCI Bus section.	
PCI_CLK_IN	K29	U2.123	I	660 PCI_CLK input. Normally connected to 664_PCI_CLK.	
USER_PCICLK1	AE01	U4.16	O	MPC970 PCI_CLK0 output.	These User PCI Clock outputs are intended for use as PCI clocks on the planar. They are nominally one half the frequency of the CPU bus clock. See the PCI Bus section.
USER_PCICLK2	N01	U4.21	O	MPC970 PCI_CLK2 output.	
USER_PCICLK3	C01	U4.23	O	MPC970 PCI_CLK3 output.	
USER_PCICLK4	A11	U4.25	O	MPC970 PCI_CLK4 output.	
USER_PCICLK5	A14	U4.29	O	MPC970 PCI_CLK5 output.	
USER_PCICLK6	E33	U4.32	O	MPC970 PCI_CLK6 output.	
Clock Control (See MPC970 data sheet for more information.)					
CLK_601_CLKS	K27	U4.40	I	MPC970 input. Tie to GND for normal operation.	
CLK_BCLK_DIV0	C16	U4.31	I	MPC970 input. Tie to GND for normal operation.	
CLK_BCLK_DIV1	B17	U4.27	I	MPC970 input. Tie to GND for normal operation.	
CLK_COM_FRZ	A18	U4.6	I	MPC970 input. Pull high or allow to float for normal operation.	
CLK_EXT_FB	A17	U4.14	I	MPC970 input. Pull high or allow to float for normal operation.	
CLK_FB_SEL	A15	U4.9	I	MPC970 input. Pull high or allow to float for normal operation.	
CLK_FRZ_STB	F33	U4.4	I	MPC970 input. Pull high or allow to float for normal operation.	
CLK_MR/3S	A16	U4.2	I	MPC970 input. Pull high or allow to float for normal operation.	

Table 2-3. Clock Signals (Continued)

Signal Name	MCM	Nodes[1]		Description (See MPC970 data sheet for more information.)
CLK_PCI_DIV0	B21	U4.20	I	MPC970 input. Pull high or allow to float for normal operation.
CLK_PCI_DIV1	C20	U4.26	I	MPC970 input. Tie to GND for normal operation.
CLK_PLL_EN	D33	U4.7	I	MPC970 input. Pull high or allow to float for normal operation.
CLK_REF_SEL	B13	U4.8	I	MPC970 input. Pull high or allow to float for normal operation.
CLK_TTL_CLK	A26	U4.11	I	MPC970 input. Pull high or allow to float for normal operation.
CLK_VCO_SEL	A13	U4.52	I	MPC970 input. Tie to GND for normal operation.
FRZ_CLK	B11	U4.3	I	MPC970 input. Can be connected to the ISA clock for use, or pulled high.
FRZ_DATA	A22	U4.5	I	MPC970 input. Pull down to GND for normal operation.
XTAL1	U30	U4.12	I/O	MPC970 crystal connection.
XTAL2	T29	U4.13	I/O	MPC970 crystal connection.

1. The following lists the names for the various nodes:
   U1 = PPC 603e
   U2 = 664
   U3 = 663
   U4 = MPC970 Clock Driver
   U5 = IDT Tag RAM
   U6 = ”245 Buffer Chip
   U7 – U10 = SRAMs

2.1.2 CPU Bus

Table 2-4. CPU Bus Signals

Signal Name	MCM	Nodes		Description
A[0:31]	(I/O)	U1, U2, U5, U6	I/O I/O I I	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. The CPU drives A[0:31] during transactions mastered by the CPU. The 660 drives A[0:31] while broadcasting CPU address bus snoop cycles in response to a PCI to memory transaction. The CPU and 660 snoop A[0:31] during transactions initiated by other CPU bus agents.
AACK#	R08	R1.A02 U1.28 U2.109	pu I/O I/O	CPU bus address acknowledge. The CPU bus target (660 or external CPU bus target) asserts AACK# to signal the end of the current address tenure. 660 also asserts AACK# for one clock to signal the end of a broadcast snoop cycle. AACK# is an input to the 664 when a CPU bus target claims the current transaction by means of L2_CLAIM#. AACK# is always an input to the CPU.
ABB#	R30	R1.A01 U1.36	pu I/O	CPU bus Address Bus Busy. While asserted, the address bus is busy. The 660 does not touch this signal.
AP0 AP1 AP2 AP3	AJ30 AH29 AG30 AE28	R1,U1 same same same	pu I/O	CPU bus Address Parity. One bit of odd (see DP[0:7]) parity per byte. AP0 maps to A[0:7], AP1 maps to A[8:15], AP2 maps to A[16:23], and AP3 maps to A[24:31]. The 660 does not check address parity.
APE_60X#	AA18	R1.F03 U1.218	pu O	CPU Address Parity Error output. The CPU signals address parity errors using this open drain output. The 660 does not touch this signal.
ARTRY#	P07	R1.A03 U1.32 U2.110	pu I/O I/O	CPU bus Address Retry. ARTRY# is asserted by a CPU bus device (either the target or a snooping master) to signal that the current address tenure needs to be rerun at a later time. The 660 samples ARTRY# (during broadcast snoops and transactions by CPU busmasters) on the second clock after TS# is sampled active. 660 will only assert ARTRY# on the clock after it asserts AACK# (during a PCI retry). See the 603e User’s Manual for more information.
BG_664#	F11	U2.134	O	660 CPU_GNT1# – 660 asserts to grant the address bus to CPU1, the MCM CPU.

Table 2-4. CPU Bus Signals (Continued)

Signal Name	MCM	Nodes		Description
BG_60X#	F09	R1.B01 U1.27	pu I	CPU BG# – CPU address bus grant input. Normally connected to BG_664#.
BG_MASTER#	N20	U2.135	O	660 CPU_GNT2# – 660 asserts to grant the bus to a second CPU busmaster.
BR_60X#	F03	R1.A05 U1.219	pu O	CPU BR# – The CPU asserts this output to request the CPU address bus. Normally connected to BR_664#.
BR_664#	D03	U2.127	I	660 CPU_REQ1# – 660 CPU bus request input for CPU1, the MCM CPU.
BR_MASTER#	AG01	R1.H05 U2.128	pu I	660 CPU_REQ2# – 660 CPU bus request input for CPU2, a second CPU busmaster.
CI_60X#	L10	R1.J02 U1.237	pu O	CPU Cache Inhibit output. Asserted to indicate that a single-beat transaction should not be cached. See 603e UM.
CKSTP_IN#	W16	R1.D02 U1.215	pu I	CPU ChecKSToP INput. When this signal is asserted, the CPU will enter checkstop.
CKSTP_OUT#	AJ02	R1.J01 U1.216	pu O	CPU ChecKSToP OUTput. The CPU asserts this open drain output to indicate that it has detected a checkstop condition and has entered the checkstop state.
CSE0	M09	U1.225	O	CPU Cache Set Element 0 output. See 603e UM.
CSE1	C12	U1.150	O	CPU Cache Set Element 1 output. See 603e UM.
D[0:63]		U1, U3, SRAM	I/O I/O I/O	The 64-bit CPU data bus. D0 is the MSbit. D[0:31] connect to CPU DH[0:31]. D[32:63] connect to CPU DL[0:31]. D[0:63] also connect to the 660 CPU_DATA[0:63], and to the SRAM.
DBB#	C08	R1.C04 U1.145	pu I/O	CPU Data Bus Busy signal. The CPU that validly asserts DBB# is the current CPU busmaster. The 660 does not touch this signal.
DBDIS#	K21	R1.A04 U1.153	pu I	CPU Data Bus DISable input. See 603e UM.
DBG_60X#	G08 (I/O)	U1.26 U2.140	I O	CPU bus Data Bus Grant. 660 asserts DBG# (while ARTRY# is inactive) to signal that the requesting CPU may take ownership of the CPU data bus. DBG# is input-only on the CPU. External agents must not drive this signal.
DBWO_60X#	N16	R1.D05 U1.25	pu I	CPU Data Bus Write Only input. Asserted to allow the CPU to run the data tenure of a write out of order. See 603e UM.
DP[0:7]		U1, U3, SRAM	I/O I/O I/O	CPU Data Parity bus. DP0 maps to D[0:7], DP1 maps to D[8:15], ... DP7 maps to D[56:63]. Odd parity is used; an odd number of bits (including the parity bit) are driven high.
DPE_60X#	AC18 (I/O)	R1.F02 U1.217 U2.133	pu O I	CPU Data Parity Error. The CPU checks data parity and reports any data beat with bad parity two CPU bus clocks after the TA# for that data beat. DPE# is output-only on the CPU, and input-only on 660. 660 samples DPE# to detect L2 parity errors.
DRTRY#	AJ01	R1.H01 U1.156	pu I	CPU Data Retry input. When asserted, means that the CPU must invalidate the data from the previous read data beat. DRTRY# is also sampled at the negation of HRESET# to set the DRTRY# mode. See the application section for details.
GBL#	A12	U1.1 U2.120	I/O O	Global. This signal is asserted during snoop operations to indicate that CPU busmasters must snoop the transaction. 660 does not monitor GBL#.
HRESET#	E32	R1.D04 U1.214	pu I	CPU Hard RESET input. Initiates hardware reset operations. See 603e UM for more information.

Table 2-4. CPU Bus Signals (Continued)

Signal Name	MCM	Nodes		Description
L2_CLAIM#	AL01	U2.132	I	660 CPU_BUS_CLAIM# input. This signal is asserted by an external CPU bus target to claim a CPU bus memory transaction. It inhibits the 660 from driving AACK, TA#, TEA#, and the CPU data bus lines. L2_CLAIM# connects only to 660. L2_CLAIM# is sampled by the 660 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 660 L2. The 660 L2 caches as much of the space from 0 to 2G as is populated by DRAM. So, if 8M is installed starting at 0, L2_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 L2_CLAIM#.
MCP_60X#	R16	R1.H02 U1.186	pu I	CPU MCP# Machine Check Pin input. The CPU enters either a machine check interrupt operation or a check stop state, depending on the state of the machine.
X_MCP_60X#	H27	U2.138	O	660 MCP# Machine Check Pin output. The 660 drives this pin to notify the CPU of a system error from a source which may not be affiliated with the current transaction. The 660 asserts MCP# in the event of a catastrophic or unrecoverable system error. The 660 asserts MCP# for two CPU bus cycles.
PLL_CFG0 PLL_CFG1 PLL_CFG2 PLL_CFG3	D25 B27 A27 C28	U1.213 U1.211 U1.210 U1.208	I I I I	CPU Phase Lock Loop Configuration Inputs. See the 603e UM for more information. Normally set to PLL_CFG[0:3] = 1100 for 99MHz CPU core and 66MHz CPU bus operation.
QACK_60X#	D17	R1.E03 U1.235	pu I	CPU Quiesce Acknowledge input. Indicates that the other CPU bus agents have ceased any bus activity that the CPU has to snoop. QACK# is sampled at the negation of HRESET# to select 32-bit mode. Drive QACK# low during HRESET# with an open collector gate to select 64-bit mode. See the applications section for details.
QREQ_60X#	C18	U1.31	O	CPU QREQ# Quiescence Request output. The CPU is requesting a low power mode. See the 603e UM.
RSRV_60X#	G16	U1.232	O	Represents the state of the Reservation bit. See 603e UM.
SHD#	D27	U2.141	I/O	660 Shared output. The function of this pin is to restore the SHD# net to a high state after it has been asserted. This pin is not used, and should be weakly pulled up.
SMI#	AL02	R1.F05 U1.187	pu I	CPU System Management Interrupt. The CPU initiates an interrupt if MSR[EE] is set (603e), else it ignores the input. Active low, level sensitive, unlatched.
SRESET#	D31	U1.189	I	CPU Soft RESET input. Async falling edge sensitive. Initiates reset exception. See 603e UM for more information.
TA#	AA16 (I/O)	R1.C05 U1.155 U2.111	pu I I/O	CPU bus Transfer Acknowledge. TA# signals that the data is valid. For every CPU clock that TA# is asserted, a data beat completes. For a single-beat cycle, TA# is only one clock.
TBEN_60X#	L16	R1.E01 U1.234	pu I	CPU Time Base Enable input. See 603e UM. Leave this input pulled high for normal operation.
TBST#	U10	R1.C03 U1.192 U2.144	pu I/O I/O	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.
TC0 TC1	G28 L30	U1.224 U1.223	O O	CPU Transfer Code outputs. See the 603e UM.
TEA#	N10 (I/O)	R1.D01 U1.154 U2.137	pu I O	CPU bus Transfer Error Acknowledge. Assertion of TEA# terminates the current data tenure, and causes a machine check exception (or checkstop) in the CPU. 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, the 660 asserts TEA# regardless of the state of MASK_TEA#.

Table 2-4. CPU Bus Signals (Continued)

Signal Name	MCM	Nodes		Description
TLBISYNC#	J16	R1.E02 U1.233	pu I	CPU TLBI sync input. See 603e UM.
TS#	L08	R1.B02 U1.149 U2.143	pu I/O I/O	CPU bus transfer start. TS# is asserted low for one CPU bus clock to signal a valid address on the address bus to start a transaction. TS# is an input to the 664 when a CPU busmaster initiates a CPU bus transaction. TS# is an output of the 664 when it initiates a snoop cycle on behalf of a PCI busmaster accessing system memory.
TSIZ0 TSIZ1 TSIZ2	L06 N06 R06	R1, U1, U2	pu O I/O	CPU bus Transfer Size in number of bytes. The TSIZ lines are valid with the CPU address lines.
TT0 TT1 TT2 TT3 TT4	A19 B19 A20 A21 A23	U1,U2 U1,U2 U1 U1,U2 U1,U2	I/O	CPU bus TT[0:4] Transfer Type. Indicates the type of transaction currently in progress. The TT lines are valid with the CPU address lines. TT2 connects only to the CPU. Normally connect TT2 to TT2_664.
TT2_664	C14	U2.153	I/O	660 TT2. Connect to TT2.
WT_60X#	D15	R1.H03 U1.236	pu O	CPU Write Through output. Asserted to indicate that a single–beat transaction is write-through.
XATS#	K07	R1.B03 U2.129	pu I	CPU bus eXtended Address Transfer Start for PIO operations, which are not supported by the 660. If XATS# is asserted, the 660 generates a TEA# error to the CPU (regardless of the setting of MASK_TEA#). The 603e CPU does not have an XATS# output. This pin is not normally connected.

2.1.3 PCI Bus

Table 2-5. PCI Bus Signals

Signal Name	MCM	Nodes		Description
664_PCI_REQ#	AF01	U2.58	O	660 PCI_REQ# PCI bus request. The 660 asserts PCI_REQ# to the PCI bus arbiter to request the PCI bus.
CPU_GNT_664#	AG02	U2.54	I	660 PCI_GNT# PCI bus grant input. PCI_GNT# is driven by the PCI bus arbiter in response to the 660 asserting PCI_REQ# to request the PCI bus.
PCI_AD[31:0]		U2, U3	I/O	660 PCI multiplexed Address–Data bus. A PCI bus transaction consists of an address phase followed by one or more data phases. 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 initiator during the address phase, and by the originator of the data (initiator or target) during the data phases.
PCI_C/BE[3:0]#	V01 W02 W01 Y01	U2.6 U2.5 U2.4 U2.3	I/O I/O I/O I/O	660 C (bus command) and BE (byte enable) multiplexed lines. During a PCI address phase C[3:0] carry the bus command. During a PCI data phase, PCI_C/BE[3:0]# are active low byte enables; BE0# enables AD[7:0], BE1# enables AD[8:15], BE2# enables AD[16:23], and BE3# enables AD24:31]. The initiator drives C/BE[3:0]#. If no bus transaction is in progress, then the current PCI busmaster must drive the PCI_C/BE[3:0]# pins.
PCI_DEVSEL#	AA02	U2.204	I/O	660 PCI device select. PCI_DEVSEL# is driven by a PCI target that is claiming the current transaction. The 660 claims PCI memory transactions from 0 to 2G within the installed DRAM space.
PCI_FRAME#	AC01	U2.200	I/O	660 PCI Frame. The current PCI busmaster 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.

Table 2-5. PCI Bus Signals (Continued)

Signal Name	MCM	Nodes		Description
PCI_IRDY#	AB01	U2.201 U3.167	I/O	660 PCI initiator ready. PCI_IRDY# is driven by the current PCI busmaster. Assertion of PCI_IRDY# indicates that the PCI initiator is ready to complete this data phase.
PCI_LOCK#	AE04	U2.53	I	660 PCI lock. This signal is used to allow PCI masters to establish a resource lock of one cache line of system memory. The 660 never asserts LOCK# as a master, but it honors PCI busmaster locks of system memory.
PCI_PAR	AH01	U2.7	I/O	660 PCI parity. Even parity across AD[31:0] and the 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 PCI_PAR. The 660 checks and drives PCI parity.
PCI_PERR#	U01	U2.10	I/O	660 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_SERR#	U02	U2.71	O	660 PCI system error. PCI_SERR# is asserted for one PCI clock when a catastrophic failure is detected while the 660 is a PCI target. PCI_SERR# is not monitored by the 660. The 660 asserts PCI_SERR# for certain PCI bus errors.
PCI_STOP#	AA01	U2.203	I/O	660 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#	AC02	U2.02 U3.168	I/O	660 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.

2.1.4 DRAM

Table 2-6. DRAM Signals

Signal Name	MCM	Nodes		Description
MA[11:0]		U2	O	660 multiplexed Memory address. MA[11] is the most significant bit.
MCE[7:0]#		U2	O	660 CAS[7:0]# Column address selects. CAS[0]# selects memory byte lane 0.
MD[63:0]			I/O	660 MEM_DATA[63:0] Memory data bus. MD[63:56] is always called memory byte lane 7 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]. **MEM DATA** / **Most Signif. Bit** / **Mem Byte Lane** / **CAS#** / **Corresponding BE mode** / **CPU Data [ ] in: LE mode** 63:56 / 63 / 7 / 7 / 56:63 / 0:7 55:48 / 55 / 6 / 6 / 48:55 / 8:15 47:40 / 47 / 5 / 5 / 40:47 / 16:23 39:32 / 39 / 4 / 4 / 32:39 / 24:31 31:24 / 31 / 3 / 3 / 24:31 / 32:39 23:16 / 23 / 2 / 2 / 16:23 / 40:47 15:8 / 15 / 1 / 1 / 8:15 / 48:55 7:0 / 7 / 0 / 0 / 0:7 / 56:63
MDP[7:0]		U3	I/O	660 MEM_CHECK[7:0] Memory ECC/parity bus. MDP[0] is always the memory check bit for memory byte lane 0 (MD[7:0]). ECC or even parity is generated and written on memory write cycles when enabled. ECC or even parity across eight bytes is checked on memory read cycles when enabled.

Table 2-6. DRAM Signals (Contnued)

Signal Name	MCM	Nodes		Description
MRE[7:0]#		U2	O	660 RAS[7:0]# Row Address Selects.
MWE0# MWE1#	M33 N33	U2.176 U2.175	O O	660 WE[1:0]# DRAM Write Enables. WE[1]# and WE[0]# are the same and are used to avoid the need for external buffers.

2.1.5 L2

Table 2-7. L2 Signals

Signal Name	MCM	Nodes		Description
ABUF[13:28]		U6 SR	O I	Buffered CPU A[13:28], for the SRAM.
DIR_245_A	G22	U6A.1 U6A.24	I I	SRAM address buffer A Direction input. Pull low for normal operation.
DIR_245_B	L22	U6B.1 U6B.24	I I	SRAM address buffer B Direction input. Pull low for normal operation.
OE_245_B#	D29	U6A.25 U6A.48 U6B.25 U6B.48	I I I I	SRAM address buffer (A & B) Output Enable input. Pull low for normal operation.
SRAM_ADS#/ ADDR0	N22	U2.124 SR.2	O I	660 SRAM ADdress Strobe. Enables latching of new address for the burst SRAM.
SRAM_ADSP#	AK03	SR.1	I	SRAM ADSP# input. Pull high for normal operation.
SRAM_ALE	U22	U2.119	O	660 SRAM Address Latch Enable. This signal is always high when burst SRAMs are used. This signal is not used on the MCM.
SRAM_CNT_EN #/ ADDR1	R22	U2.125 SR.52	O I	660 SRAM CouNT ENable. Enables incrementing of burst address for the burst SRAM.
SRAM_CS#	AD29	SR.5	I	SRAM Chip Select. Pull to GND for normal operation.
SRAM_OE#	AA22	SR.50	I	SRAM Output Enable input.
X_SRAM_OE#	G26	U2.117	O	660 SRAM Output Enable output. Connect to SRAM_OE# for normal operation.
SRAM_WE#	W22	U2.114 SR.3,4	O I	660 SRAM Write Enable.
TA_GATES	N24	U5.23,2 7,28	I	TAG TT1, TAH, & TAIN# tied together and pulled high with 2 resistors. Leave pulled high for normal operation.
TAG_ADDR_13	L26	U5.33	I	TAG A13. Pull high for 256K L2. Connect to A13 for 512K L2.
TAG_A_IN	AC04	U6A	I/O	SRAM buffer spare inputs. Pull to GND or Vdd for normal operation.
TAG_CLR#	AC22	U2.116 U5.70	O I	660 Tag RAM clear. When asserted, all tags are forced to the invalid state. See the L2 Invalidate BCR.
TAG_CS1#	C33	U5.75	I	TAG CS1# Pull this pin to GND for normal operation.
TAG_CS2	H31	U5.76	I	TAG CS2. Pull this pin high for normal operation.
TAG_DATA_11	M27	U5.65	I	TAG D13. Connect to A13 for 256K L2. Pull high for 512K L2.
TAG_DTY	P25	U5.34	O	TAG DTY/S3 output. Not used by MCM.
TAG_MATCH	K31	U5.50 U2.142	O I	TAG match indication (cache hit). This signal is asserted for a cache hit. It is an active high, open drain output of the tag RAM. Pull this signal to 3.3v with a 200 ohm (nominal) resistor.
TAG_PWRDN#	M31	U5.77	I	TAG Power down mode control. Pull this pin high for normal operation.
TAG_SFUNC	C32	U5.22	I	TAG Status Bit Function Control input. Pull to GND for normal operation.
TAG_TA#	N28	U5.51	O	TAG TA# output. Not used by MCM.
TAG_VALID	AC20	U2.115 U5.80	O I	660 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.
TAG_VLD	L28	U5.54	O	TAG VLD/S1 output. Not used by MCM.

Table 2-7. L2 Signals (Continued)

Signal Name	MCM	Nodes		Description
TAG_WE#	AA20	U2.114 U5.73,74	O I	660 Tag RAM write enable. Asserted to write to the Tag RAM.
TAG_WT	J28	U5.47	O	TAG WT/S3 output. Not used by MCM.
TAG_WT_DTY_IN	L24	U5.5,6 R1.G04	I pu	TAG DTY/S2 input tied to WT/S3 input and pulled high. Leave pulled high for normal operation.
TAOE#	P31	U5.23,27,28 R1, R1	I pu pu	TAG TAOE#, OE_STAT#, and OE_TAG# signals wired together & pulled up. Leave pulled up for normal operation.

2.1.6 System Interface

Table 2-8. System Interface Signals

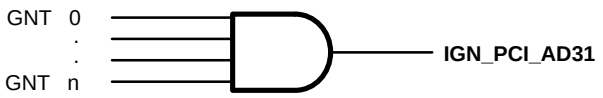
Signal Name	MCM	Nodes		Description
IGN_PCI_AD31	AD01	U2.57	I	660 Ignore PCI_AD[31] input. This signal is required when the Intel™ SIO is the PCI master, because it does not latch ISA_MASTER on posted ISA writes to 0 to 16M. IGN_PCI_AD31 is used to allow ISA busmasters to access system memory when the 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. 
664_STOP_CLK_EN#	B29	R1.F04 U2.151	pu I	660 input. Prepares the 660 for stopping the CPU_CLK during power management. This feature is not supported. Leave this pin pulled high for normal operation.
Exceptions				
INT_TO_664	B31	U2.55	I	660 INT_REQ Interrupt request input. The 660 synchronizes this signal from the interrupt controller to the CPU bus clock and passes it through to the CPU as an interrupt.
INT_60X#	U16	R1.D03 U1.188	pu I	CPU INT# Interrupt input. The CPU initiates an interrupt if MSR[EE] is set (603e), else it ignores the input. Active low, level sensitive, unlatched.
X_INT_60X#	F31	U2.139	O	660 INT_CPU# CPU interrupt output. The 660 asserts INT_CPU# to signal the CPU to run an interrupt cycle. The software is expected to eventually run a PCI interrupt acknowledge transaction to get the interrupt vector. 660 can assert INT_CPU# in response to an INT_REQ input. Normally connected to INT_60X#
NMI_FROM_ISABRDG	A31	U2.56	I	660 NMI_REQ input. Non–maskable interrupt request. When detected active (normally from the ISA bridge), an error is reported to the CPU.
POWER_GOOD/RESET#	A28	U2.156	I	660 Power–On–RESET#. While 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				
ROM_OE#	V31	U2.47	O	660 ROM output enable. ROM_OE# enables direct-attached ROM. This signal is always high during remote ROM operation.
ROM_WE#	T31	U2.60	O	660 ROM write enable. Write enable for flash ROM for direct-attach ROM. This signal is always high during remote ROM operation.

Table 2-8. System Interface Signals (Continued)

Signal Name	MCM	Nodes		Description
Daisy Chain				
Daisy01	E04, D05			The DAISYxx columns are arranged in pairs which are connected to each other. For example, Daisy01 consists of columns E04 and D05, which are connected to each other.
Daisy02	E06, D07			
Daisy03	E08, D09			
Daisy04	E10, D11			
Daisy05	E12, D13			
Daisy06	E30, F29			
Daisy07	E28, F27			
Daisy08	E26, F25			
Daisy09	E24, F23			
Daisy10	E22, F21			
Daisy11	AK05, AK07			
Daisy12	AKO9, AK11			
Daisy13	AK13, AK15			
Daisy14	AK17, AK19			
Daisy15	AK21, AK23			
Daisy16	AD03, AF03			
Daisy17	Y03, AB03			
Daisy18	T03, V03			
Daisy19	M03, P03			
Daisy20	H03, K03			
Power & Ground				
$60X\_AV_{DD}$	B02	U1.209		603e Analog VDD. Bypass as recommended in the 603e documentation.
V_{DD1}				
V_{DD2}				
V_{DD3}				
V_{SS} (GND)				

2.1.7 660 InterChip Communication

These signals provide communication between the 663 and the 664. They are generally not intended for use by the system. Note that CPU_RDL_OPEN requires a resistor or other delay component between the 663 and the 664. For more information, see Section 2 of the 660 User's Manual.

Table 2-9. InterChip Communication Signals

Signal Name	MCM	Nodes		Description
663_CPU_PAR_ERR#	G18	U2.192 U3.174	I O	660 CPU_PAR_ERR# signal. CPU Data Bus Parity Error. When asserted, this signal indicates a parity error on the CPU data bus during a write cycle.
AOS_RR_MMRS	W20	U2.69 U3.166	O I	660 All Ones Select/ROM Remote/Mask MEM_RD_SMPL signal. This signal is used to force the data bus to 64 one-bits. While ROM_LOAD is asserted, this signal is used to determine the location of the ROM. When the PCI is burst reading memory, MASK_MEM_RD_SMPL is asserted after the first MEM_RD_SMPL, and stays asserted until the PCI–to–MEM read latch is empty.

Table 2-9. InterChip Communication Signals (Continued)

Signal Name	MCM	Nodes		Description
C2P_WRL_OPEN	AA14	U2.61 U3.154	O I	660 CPU–to–PCI Write Latch Open signal. 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.
CPU_DATA_OE#	J14	U2.197 U3.146	O I	660 CPU Data Output Enable signal. When asserted, the 663 drives the CPU_DATA bus on the next CPU_CLK.
CPU_RDL_OPEN	G12	U2.50	O	664 CPU Read Latch Open output. 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.
X_CPU_RDL_ OPEN	G10	U3.148	I	663 CPU Read Latch Open input. See the 660 User's Manual for more information. Add a 200Ω (nominal) 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. Case — 663 Requires — 664 Supplies 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.
CRS_C2PWXS	A29	U2.65 U3.151	O I	660 CPU Read Select/CPU-to-PCI Write Crossover Select signal. When the CPU read latch is sampling data, this signal controls the CPU read multiplexer. When the CPU-to-PCI write latch is sampling data, this signal controls the CPU-to-PCI write crossover. Pull down with a 1K (nominal) resistor to select (during reset) direct-attach ROM.
DUAL_CTRL_REF	N12	U2.205 U3.170	O I	660 Control Signal Mux Select signal. For 663 inputs that have two functions this signal selects the function. This signal is generated by dividing CPU_CLK by two. This is a useful first point to check when debugging a "dead" system.
ECC_LE_SEL	L12	U2.2 U3.149	O I	660 ECC Select/Little-Endian Select signal. 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.
MEM_BE[3:0]		U2. U3	O, I	660 Memory Byte Enables signal. The eight BEs for read-modify-write memory cycles are multiplexed on MEM_BE[3:0]. 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#	L14	U2.196 U3.145	O I	660 Memory Data Output Enable signal. When asserted, the 663 drives the MEM_DATA bus on the next CPU_CLK.

Table 2-9. InterChip Communication Signals (Continued)

Signal Name	MCM	Nodes		Description
MEM_ERR# (663_MEM_ERR#)	J20	U2.194 U3.171	I O	660 Memory Error signal. 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	J12	U2.49 U3.161	O I	660 Memory Read Sample signal. This signal 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.
MEM_WRL_OPEN	AC14	U2.51 U3.150	O I	660 Memory Write Latch Open signal. When asserted, the MEM write latch accepts new data on each CPU_CLK. When deasserted, the MEM write latch holds its current contents.
MWS_P2MRXS	C30	U2.66 U3.152	O I	660 Memory write select/PCI-to-memory read crossover select signal. When the memory write latch is sampling data, this signal controls the memory write multiplexer. Pull up with 10K Ω (nominal) resistor to select (during reset) 603e in 3:2 core:bus clock mode.
PCI_AD_OE#	N14	U2.195 U3.144	O I	660 PCI Data Output Enable signal. While asserted, the 663 drives the PCI_AD bus. Note: This is an asynchronous input to the 663.
PCI_EXT_SEL	U14	U2.67 U3.153	O I	660 PCI Read Extension Select/PCI Write Extension Select signal. When the PCI is reading from memory, this signal controls the PCI read extension multiplexer.
PCI_OL_OPEN	W14	U2.64 U3.165	O I	660 PCI Other Latches Open signal. This signal controls the latch enables for the PCI-to-MEM read latch, the PCI read extension latch, and the PCI write extension latch.
PCI_OUT_SEL	R14	U2.68 U3.169	O I	660 PCI Output Select signal. When asserted, memory data is routed to the PCI output bus, else CPU data is routed to the PCI output bus. This signal is asynchronous.
ROM_LOAD	U20	U2.70 U3.160	O I	660 ROM Load signal. 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.
663_SBE#	G20	U2.193 U3.175	I O	660 SBE# Single-Bit Error signal. 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.

2.1.8 Test Signals

Table 2-10. Test Signal Descriptions

Signal Name	MCM	Nodes		Description
CPU Test Signals				
LSSD_MODE#	R18	R1.E04 U1.205	pu I	CPU LSSD test input. Leave this input pulled high during normal operation.
L1_TST_CLK#	N18	R1.E05 U1.204	pu I	CPU LSSD test input. Leave this input pulled high during normal operation.
L2_TST_CLK#	L18	R1.F01 U1.203	pu I	CPU LSSD test input. Leave this input pulled high during normal operation.
60X_CLK_OUT	J18	U1.221	O	CPU system CLocK OUTput. CLK_OUT is a clock test output.
TDO	C26	U1.198	O	CPU JTAG serial scan output.
TMS	A24	R1.J03 U1.200	pu I	CPU JTAG TAP controller mode input. Leave this input pulled high during normal operation.
TDI	A25	R1.J05 U1.199	pu I	CPU JTAG serial scan input. Leave this input pulled high during normal operation.
TCK	B25	R1.J04 U1.201	pu I	CPU JTAG scan clock input. Leave this input pulled high during normal operation.

Table 2-10. Test Signal Descriptions (Continued)

Signal Name	MCM	Nodes		Description
TRST#	C24	R1.H04 U1.202	pu I	CPU JTAG TAP controller reset. Most systems will assert this signal while either HRESET# or TRST# (from the RISCWatch connector) are asserted. See the applications section.
660 Test Signals				
663_MIO_TEST	A30	U3.156	I	Chip level test. Deassert low for normal operation. Do not casually assert this signal.
664_MIO_TEST	K05	U2.154	I	
663_TEST#	G14	U3.155	I	Test mode. Pull to logic high during normal operation. Do not casually assert this signal.
664_TEST#	J04	U2.155	I	

2.2 Net Names and MCM Nodes

2.2.1 Complete Pin List for MCM

Table 2-11 matches version 5.0 of the schematic. Note that, while all nets on the MCM are brought to the MCM I/O, only a subset is intended for customer use (see Table 2-2). The rest of the I/O signals can be left floating.

Table 2-11. Pin List With Net Names and MCM Nodes

Pin[1]	Net Name[2]	MCM Nodes[3]
A03	PCI_AD20	U2.24,U3.44
A04	PCI_AD21	U2.23,U3.45
A05	PCI_AD23	U2.21,U3.47
A06	PCI_AD24	U2.20,U3.48
A07	PCI_AD26	U2.18,U3.50
A08	PCI_AD27	U2.15,U3.51
A09	PCI_AD29	U2.13,U3.63
A10	PCI_AD30	U2.12,U3.64
A11	USER_PCICLK4	U4.25
A12	GBL#	U1.1,U2.120
A13	CLK_VCO_SEL	U4.52
A14	USER_PCICLK5	U4.29
A15	CLK_FB_SEL	U4.9
A16	CLK_MR/ TRISTATE	U4.2
A17	CLK_EXT_FB	U4.14
A18	CLK_COM_FRZ	U4.6
A19	TT0	U1.191,U2.150
A20	TT2	U1.185
A21	TT3	U1.184,U2.126
A22	FRZ_DATA	U4.5
A23	TT4	U1.180,U2.136
A24	TMS#	R1.J03,U1.200
A25	TDI	R1.J05,U1.199
A26	CLK_TTL_CLK	U4.11
A27	PLL_CFG2	U1.210
A28	POWER_GOOD/RE- SET#	U2.156
A29	CRS_C2PWXS	U2.65,U3.151

Pin[1]	Net Name[2]	MCM Nodes[3]
A30	663_MIO_TEST	U3.156
A31	NMI_FROM_ ISABRDG	U2.56
B02	60X_AVDD	U1.209
B03	PCI_AD18	U2.28,U3.42
B04	VSS	
B05	PCI_AD22	U2.22,U3.46
B06	VDD1	
B07	PCI_AD25	U2.19,U3.49
B08	VSS	
B09	PCI_AD28	U2.14,U3.62
B10	VDD1	
B11	FRZ_CLK	U4.3
B12	VSS	
B13	CLK_REF_SEL	U4.8
B14	VDD1	
B15	VSS	
B16	VSS	
B17	CLK_BCLK_DIV1	U4.27
B18	VSS	
B19	TT1	U1.190,U2.152
B20	VDD1	
B21	CLK_PCI_DIV0	U4.20
B22	VSS	
B23	NC	
B24	VDD1	
B25	TCK	R1.J04,U1.201
B26	VSS	

Pin[1]	Net Name[2]	MCM Nodes[3]
B27	PLL_CFG1	U1.211
B28	VDD1	
B29	664_STOP_CLK_EN#	R1.F04 U2.151
B30	VSS	
B31	INT_TO_664	U2.55
B32	VDD1	
C01	USER_PCICLK3	U4.23
C02	PCI_AD17	U2.29,U3.41
C03	NC, DEPOP	
C04	PCI_AD19	U2.25,U3.43
C05	NC, DEPOP	
C06	NC	
C07	NC, DEPOP	
C08	DBB#	R1.C04,U1.145
C09	NC, DEPOP	
C10	PCI_AD31	U2.11,U3.65
C11	NC, DEPOP	
C12	603E_CSE0#	U1.150
C13	NC, DEPOP	
C14	TT2_664	U2.153
C15	NC, DEPOP	
C16	CLK_BCLK_DIV0	U4.31
C17	NC, DEPOP	
C18	QREQ_60X#	U1.31
C19	NC, DEPOP	
C20	CLK_PCI_DIV1	U4.26
C21	NC, DEPOP	
C22	VSS	
C23	NC, DEPOP	
C24	TRST#	R1.H04,U1.202
C25	NC, DEPOP	
C26	TDO	U1.198
C27	NC, DEPOP	
C28	PLL_CFG3	U1.208
C29	NC, DEPOP	
C30	MWS_P2MRXS	U2.66,U3.152
C31	NC, DEPOP	
C32	TAG_SFUNC	U5.22
C33	TAG_CS1#	U5.75
D01	PCI_AD16	U2.30,U3.40
D02	VSS	
D03	BR_664#	U2.127
D04	VDD1	
D05	DAISY01	J1.E04
D06	VSS	
D07	DAISY02	J1.E06
D08	VDD1	

Pin[1]	Net Name[2]	MCM Nodes[3]
D09	DAISY03	J1.E08
D10	VSS	
D11	DAISY04	J1.E10
D12	VDD1	
D13	DAISY05	J1.E12
D14	VSS	
D15	WT_60X#	R1.H03,U1.236
D16	VDD1	
D17	QACK_60X#	R1.E03,U1.235
D18	VDD1	
D19	X_PCLK_60X	U1.212
D20	VSS	
D21	NC	
D22	VDD1	
D23	NC	
D24	VSS	
D25	PLL_CFG0	U1.213
D26	VDD1	
D27	SHD#	U2.141
D28	VSS	
D29	OE_245_B	U6A.25A,U6A.48A, U6B.25B,U6B.48B
D30	VDD1	
D31	SRESET_60X#	U1.189
D32	VSS	
D33	CLK_PLL_EN	U4.7
E01	PCI_AD14	U2.32,U3.20
E02	PCI_AD15	U2.31,U3.21
E03	NC, DEPOP	
E04	DAISY01	J1.D05
E05	NC, DEPOP	
E06	DAISY02	J1.D07
E07	NC, DEPOP	
E08	DAISY03	J1.D09
E09	NC, DEPOP	
E10	DAISY04	J1.D11
E11	NC, DEPOP	
E12	DAISY05	J1.D13
E13	NC, DEPOP	
E14	NC	
E15	NC, DEPOP	
E16	NC	
E17	NC, DEPOP	
E18	PCLK_60X	U4.34
E19	NC, DEPOP	
E20	NC	
E21	NC, DEPOP	

Pin[1]	Net Name[2]	MCM Nodes[3]
E22	DAISY10	J1.F21
E23	NC, DEPOP	
E24	DAISY09	J1.F23
E25	NC, DEPOP	
E26	DAISY08	J1.F25
E27	NC, DEPOP	
E28	DAISY07	J1.F27
E29	NC, DEPOP	
E30	DAISY06	J1.F29
E31	NC, DEPOP	
E32	HRESET#	R1.D04,U1.214
E33	USER_PCICLK6	U4.32
F01	PCI_AD13	U2.33,U3.19
F02	VDD1	
F03	BR_60X#	R1.A05,U1.219
F04	VSS	
F05	X_663_CPU_CLK	U3.157
F06	VDD1	
F07	MEM_BE1	U2.207,U3.162
F08	VSS	
F09	BG_60X#	R1.B01,U1.27
F10	VDD1	
F11	BG_664#	U2.134
F12	VSS	
F13	NC	
F14	VDD1	
F15	NC	
F16	VSS	
F17	NC	
F18	VSS	
F19	NC	
F20	VDD1	
F21	DAISY10	J1.E22
F22	VSS	
F23	DAISY09	J1.E24
F24	VDD1	
F25	DAISY08	J1.E26
F26	VSS	
F27	DAISY07	J1.E28
F28	VDD1	
F29	DAISY06	J1.E30
F30	VSS	
F31	X_INT_60X#	U2.138
F32	VDD1	
F33	CLK_FRZ_STROBE	U4.4
G01	PCI_AD11	U2.35,U3.17
G02	PCI_AD12	U2.34,U3.18

Pin[1]	Net Name[2]	MCM Nodes[3]
G03	NC, DEPOP	
G04	663_CPU_CLK	U4.38
G05	NC, DEPOP	
G06	MEM_BE3	U2.1,U3.164
G07	NC, DEPOP	
G08	DBG_60X#	U1.26,U2.140
G09	NC, DEPOP	
G10	X_CPU_RDL_OPEN	U3.148
G11	NC, DEPOP	
G12	CPU_RDL_OPEN	U2.50
G13	NC, DEPOP	
G14	663_TEST#	U3.155
G15	NC, DEPOP	
G16	#RSRV_60X	U1.232
G17	NC, DEPOP	
G18	663_CPU_PAR_ERR#	U2.192,U3.174
G19	NC, DEPOP	
G20	663_SBE#	U2.193,U3.175
G21	NC, DEPOP	
G22	DIR_245_A	U6A.1A,U6A.24A
G23	NC, DEPOP	
G24	NC	
G25	NC, DEPOP	
G26	X_SRAM_OE#	U2.117
G27	NC, DEPOP	
G28	TC0	U1.224
G29	NC, DEPOP	
G30	664_CPU_CLK	U4.42
G31	NC, DEPOP	
G32	MCE6#	U2.168
G33	MCE7#	U2.165
H01	PCI_AD10	U2.3, U3.16
H02	VSS	
H03	DAISY20	J1.K03
H04	VDD1	
H05	MEM_BE0	U2.206,U3.161
H06	VSS	
H07	NC	
H08	VDD1	
H09	NC	
H10	VSS	
H11	NC	
H12	VDD1	
H13	NC	
H14	VSS	
H15	NC	
H16	VDD1	

Pin[1]	Net Name[2]	MCM Nodes[3]
H17	NC	
H18	VDD1	
H19	NC	
H20	VSS	
H21	NC	
H22	VDD1	
H23	NC	
H24	VSS	
H25	NC	
H26	VDD1	
H27	X_MCP_60X#	U2.138
H28	VSS	
H29	X_664_CPU_CLK	U2.121
H30	VDD1	
H31	TAG_CS2	U5.76
H32	VSS	
H33	MCE5#	U2.169
J01	PCI_AD9	U2.37,U3.15
J02	NC	
J03	NC, DEPOP	
J04	664_TEST#	U2.155
J05	NC, DEPOP	
J06	MEM_BE2	U2.208,U3.163
J07	NC, DEPOP	
J08	NC	
J09	NC, DEPOP	
J10	NC	
J11	NC, DEPOP	
J12	MEM_RD_SMPL	U2.49,U3.147
J13	NC, DEPOP	
J14	CPU_DATA_OE#	U2.197,U3.146
J15	NC, DEPOP	
J16	TLBISYNC#	R1.E02,U1.233
J17	NC, DEPOP	
J18	60X_CLK_OUT	U1.221
J19	NC, DEPOP	
J20	663_MEM_ERR#	U2.194,U3.171
J21	NC, DEPOP	
J22	NC	
J23	NC, DEPOP	
J24	NC	
J25	NC, DEPOP	
J26	VSS	
J27	NC, DEPOP	
J28	TAG_WT	U5.47
J29	NC, DEPOP	
J30	664_PCI_CLK	U4.18

Pin[1]	Net Name[2]	MCM Nodes[3]
J31	NC, DEPOP	
J32	MCE3#	U2.171
J33	MCE4#	U2.170
K01	PCI_AD8	U2.38,U3.14
K02	VDD1	
K03	DAISY20	J1.H03
K04	VSS	
K05	664_MIO_TEST	U2.154
K06	VDD1	
K07	–XATS	R1.B03,U2.129
K08	VSS	
K09	NC	
K10	VDD1	
K11	NC	
K12	VSS	
K13	NC	
K14	VDD1	
K15	NC	
K16	VSS	
K17	NC	
K18	VSS	
K19	NC	
K20	VDD1	
K21	DBDIS#	R1.A04,U1.153
K22	VSS	
K23	NC	
K24	VDD1	
K25	NC	
K26	VSS	
K27	CLK_MPC601_CLKS	U4.40
K28	VDD1	
K29	PCI_CLK_IN	U2.123
K30	VSS	
K31	TAG_MATCH	U2.142mU5.50
K32	VDD1	
K33	MCE2#	U2.172
L01	PCI_AD6	U2.40,U3.12
L02	PCI_AD7	U2.39,U3.13
L03	NC, DEPOP	
L04	ABUF24	U7.22,U8.22,U9.22, U10.22,U6B.41B
L05	NC, DEPOP	
L06	TSIZ0	R1.K05,U1.197, U2.145
L07	NC ,DEPOP	
L08	TS#	R1.B02,U1.149, U2.143
L09	NC, DEPOP	

Pin[1]	Net Name[2]	MCM Nodes[3]
L10	CI_60X#	R1.J02,U1.237
L11	NC, DEPOP	
L12	ECC_LE_SEL	U2.2,U3.149
L13	NC, DEPOP	
L14	MEM_DATA_OE#	U2.196,U3.145
L15	NC, DEPOP	
L16	TBEN_60X#	R1.E01,U1.234
L17	NC, DEPOP	
L18	L2_TST_CLK#	R1.F01,U1.203
L19	NC, DEPOP	
L20	NC	
L21	NC, DEPOP	
L22	DIR_245_B	U6B.1B,U6B.24B
L23	NC, DEPOP	
L24	TAG_WT_DTY_IN	R1.G04,U5.5,U5.6
L25	NC, DEPOP	
L26	TAG_ADDR_13	U5.33
L27	NC, DEPOP	
L28	TAG_VLD	U5.54
L29	NC, DEPOP	
L30	TC1	U1.223
L31	NC, DEPOP	
L32	MCE0#	U2.174
L33	MCE1#	U2.173
M01	PCI_AD5	U2.41,U3.11
M02	VSS	
M03	DAISY19	J1.P03
M04	VDD1	
M05	ABUF23	U7.21,U8.21,U9.21, U10.21,U6B.40B
M06	VSS	
M07	NC	
M08	VDD1	
M09	CSE	U1.225
M10	VSS	
M11	NC	
M12	VDD1	
M13	NC, DEPOP	
M14	VSS	
M15	NC, DEPOP	
M16	VDD1	
M17	NC, DEPOP	
M18	VDD1	
M19	NC, DEPOP	
M20	VSS	
M21	NC, DEPOP	
M22	VDD3	

Pin[1]	Net Name[2]	MCM Nodes[3]
M23	NC	
M24	VSS	
M25	NC	
M26	VDD3	
M27	TAG_DATA_11	U5.65
M28	VSS	
M29	NC	
M30	VDD3	
M31	TAG_PWRDN#	U5.77
M32	VSS	
M33	MWE0#	U2.176
N01	USER_PCICLK2	U4.21
N02	PCI_AD4	U2.42,U3.10
N03	NC, DEPOP	
N04	ABUF22	U7.7,U8.7,U9.7, U10.7,U6B.38B
N05	NC, DEPOP	
N06	TSIZ1	R1.K01,U1.196, U2.146
N07	NC, DEPOP	
N08	NC	
N09	NC, DEPOP	
N10	TEA#	R1.D01,U1.154, U2.137
N11	NC, DEPOP	
N12	DUAL_CTRL_REF	U2.205,U3.170
N13	NC, DEPOP	
N14	PCI_AD_OE#	U2.195,U3.144
N15	NC, DEPOP	
N16	DBWO_60X#	R1.D05,U1.25
N17	NC, DEPOP	
N18	L1_TST_CLK#	R1.E05,U1.204
N19	NC, DEPOP	
N20	BG_MASTER#	U2.135
N21	NC, DEPOP	
N22	SRAM_ADS/ADDR0	U2.124,U7.2,U8.2, U9.2,U10.2
N23	NC, DEPOP	
N24	TA_GATES	R1.G03,R1.G05, U5.23,U5.27,U5.28
N25	NC, DEPOP	
N26	A22	U1.160,U2.99,U5.13, U6B.11B
N27	NC, DEPOP	
N28	TAG_TA#	U5.51
N29	NC, DEPOP	
N30	TAG_BCLK	U4.36
N31	NC, DEPOP	
N32	MDP7	U3.195

Pin[1]	Net Name[2]	MCM Nodes[3]
N33	MWE1#	U2.175
P01	PCI_AD3	U2.43,U3.239
P02	VDD1	
P03	DAISY19	J1.M03
P04	VSS	
P05	ABUF21	U7.6,U8.6,U9.6, U10.6,U6B.37B
P06	VDD1	
P07	ARTRY#	R1.A03.U1.32, U2.110
P08	VSS	
P09	NC	
P10	VDD1	
P11	NC	
P12	VSS	
P13	NC, DEPOP	
P14	VDD1	
P15	NC, DEPOP	
P16	VSS	
P17	NC, DEPOP	
P18	VSS	
P19	NC, DEPOP	
P20	VDD1	
P21	NC, DEPOP	
P22	VSS	
P23	NC	
P24	VDD1	
P25	TAG_DTY	U5.34
P26	VSS	
P27	A30	U1.144,U2.107
P28	VDD1	
P29	X_TAG_BCLK	U5.69
P30	VSS	
P31	TAOE#	R1.G01,R1.G02, U5.66,U5.67,U5.68
P32	VDD1	
P33	MD63	U3.140
R01	PCI_AD1	U2.59,U3.237
R02	PCI_AD2	U2.46,U3.238
R03	NC, DEPOP	
R04	ABUF20	U7.49,U8.49,U9.49, U10.49,U6B.36B
R05	NC, DEPOP	
R06	TSIZ2	R1.K02,U1.195, U2.147
R07	NC, DEPOP	
R08	AACK#	R1.A02,U1.28, U2.109
R09	NC, DEPOP	

Pin[1]	Net Name[2]	MCM Nodes[3]
R10	NC	
R11	NC, DEPOP	
R12	NC	
R13	NC, DEPOP	
R14	PCI_OUT_SEL	U2.68,U3.169
R15	VDD1	
R16	MCP_60X#	R1.H02,U1.186
R17	VDD1	
R18	LSSD_MODE#	R1.E04,U1.205
R19	VDD1	
R20	NC	
R21	NC, DEPOP	
R22	SRAM_CNT_EN/ ADDR1	U2.125,U7.52,U8.52, U9.52,U10.52
R23	NC, DEPOP	
R24	A10	U1.170,U2.84,U5.57
R25	NC, DEPOP	
R26	A13	U1.12,U2.87, U6B.23B
R27	NC, DEPOP	
R28	A27	U1.23,U2.104, U6A.9A
R29	NC, DEPOP	
R30	ABB#	R1.A01,U1.36
R31	NC, DEPOP	
R32	MD61	U3.138
R33	MD62	U3.139
T01	PCI_AD0	U2.48,U3.236
T02	VSS	
T03	DAISY18	J1.V03
T04	VDD1	
T05	ABUF19	U7.48,U8.48,U9.48, U10.48,U6B.35B
T06	VSS	
T07	NC	
T08	VDD1	
T09	NC	
T10	VSS	
T11	NC	
T12	VDD1	
T13	NC, DEPOP	
T14	VSS	
T15	NC, DEPOP	
T16	VDD1	
T17	NC, DEPOP	
T18	VDD1	
T19	NC, DEPOP	
T20	VSS	

Pin[1]	Net Name[2]	MCM Nodes[3]
T21	NC, DEPOP	
T22	VDD3	
T23	A26	U1.158,U2.103,U5.7, U6B.5B
T24	VSS	
T25	A9	U1.7,U2.83,U5.56
T26	VDD3	
T27	A18	U1.165,U2.93,U5.21, U6B.16B
T28	VSS	
T29	XTAL2	U4.13
T30	VDD3	
T31	ROM_WE#	U2.60
T32	VSS	
T33	MD60	U3.133
U01	PCI_PERR#	U2.10
U02	PCI_SERR#	U2.71
U03	NC, DEPOP	
U04	ABUF18	U7.47,U8.47,U9.47, U10.47,U6B.33B
U05	NC, DEPOP	
U06	ABUF28	U7.26,U8.26,U9.26, U10.26,U6B.47B
U07	NC, DEPOP	
U08	NC	
U09	NC, DEPOP	
U10	TBST#	R1.C03,U1.192, U2.144
U11	NC, DEPOP	
U12	NC	
U13	NC, DEPOP	
U14	PCI_EXT_SEL	U2.67,U3.153
U15	VDD1	
U16	INT_60X#	R1.D03,U1.188
U17	VDD1	
U18	NC	
U19	VDD1	
U20	ROM_LOAD	U2.70,U3.160
U21	NC, DEPOP	
U22	SRAM_ALE	U2.119
U23	NC, DEPOP	
U24	A8	U1.174,U2.82,U5.55
U25	NC, DEPOP	
U26	A17	U1.15,U2.92,U5.29, U6B.17B
U27	NC, DEPOP	
U28	A25	U1.22,U2.102,U5.8, U6B.6B
U29	NC, DEPOP	
U30	XTAL1	U4.12

Pin[1]	Net Name[2]	MCM Nodes[3]
U31	NC, DEPOP	
U32	MD58	U3.131
U33	MD59	U3.132
V01	PCI_C/BE0#	U2.6
V02	VSS	
V03	DAISY18	J1.T03
V04	VDD2	
V05	ABUF17	U7.33,U8.33,U9.33, U10.33,U6B.32B
V06	VSS	
V07	ABUF27	U7.25,U8.25,U9.25, U10.25,U6A.40A
V08	VDD2	
V09	NC	
V10	VSS	
V11	DP7	U1.50,U3.137, U10.20
V12	VDD2	
V13	NC, DEPOP	
V14	VSS	
V15	NC, DEPOP	
V16	VDD2	
V17	NC, DEPOP	
V18	VDD2	
V19	NC, DEPOP	
V20	VSS	
V21	NC, DEPOP	
V22	VDD3	
V23	A31	U1.37,U2.108
V24	VSS	
V25	A7	U1.6,U2.81,U5.46
V26	VDD3	
V27	A16	U1.166,U2.91,U5.30, U6A.11A
V28	VSS	
V29	A24	U1.159,U2.101,U5.9, U6B.8B
V30	VDD3	
V31	ROM_OE#	U2.47
V32	VSS	
V33	MD57	U3.130
W01	PCI_C/BE2#	U2.4
W02	PCI_C/BE1#	U2.5
W03	NC, DEPOP	
W04	ABUF16	U7.32,U8.32,U9.32, U10.32,U6A.38A
W05	NC, DEPOP	
W06	ABUF15	U7.31,U8.31,U9.31, U10.31,U6A.37A
W07	NC, DEPOP	

Pin[1]	Net Name[2]	MCM Nodes[3]
W08	ABUF14	U7.30,U8.30,U9.30, U10.30,U6B.27B
W09	NC, DEPOP	
W10	DP4	U1.46,U3.77,U9.46
W11	NC, DEPOP	
W12	NC	
W13	NC, DEPOP	
W14	PCI_OL_OPEN	U2.64,U3.165
W15	VDD2	
W16	CKSTP_IN#	R1.D02,U1.215
W17	VDD2	
W18	NC	
W19	VDD2	
W20	AOS_RR_MMRS	U2.69,U3.166
W21	NC, DEPOP	
W22	SRAM_WE#	U2.118,U7.3,U7.4, U8.3,U8.4,U9.3, U9.4,U10.3,U10.4
W23	NC, DEPOP	
W24	A6	U1.175,U2.80,U5.45
W25	NC, DEPOP	
W26	A15	U1.13,U2.90,U5.31, U6A.12A
W27	NC, DEPOP	
W28	A23	U1.21,U2.100,U5.12, U6B.9B
W29	NC, DEPOP	
W30	SRAM_BCLK0	U4.44
W31	NC, DEPOP	
W32	MDP6	U3.214
W33	MD56	U3.123
Y01	PCI_C/BE3#	U2.3
Y02	VDD2	
Y03	DAISY17	J1.AB03
Y04	VSS	
Y05	X_SRAM_BCLK3	U10.51
Y06	VDD2	
Y07	NC	
Y08	VSS	
Y09	DP1	U1.40,U3.219,U7.20
Y10	VDD2	
Y11	DP6	U1.48,U3.116, U10.46
Y12	VSS	
Y13	NC, DEPOP	
Y14	VDD2	
Y15	NC, DEPOP	
Y16	VSS	
Y17	NC, DEPOP	

Pin[1]	Net Name[2]	MCM Nodes[3]
Y18	VSS	
Y19	NC, DEPOP	
Y20	VDD2	
Y21	NC, DEPOP	
Y22	VSS	
Y23	NC	
Y24	VDD2	
Y25	A5	U1.5,U2.77,U5.40
Y26	VSS	
Y27	A14	U1.168,U2.89,U5.32, U6B.22B
Y28	VDD2	
Y29	X_SRAM_BCLK0	U7.51
Y30	VSS	
Y31	MA11	U2.177
Y32	VDD2	
Y33	MD55	U3.121
AA01	PCI_STOP#	U2.203
AA02	PCI_DEVSEL#	U2.204
AA03	NC, DEPOP	
AA04	SRAM_BCLK3	U4.50
AA05	NC, DEPOP	
AA06	ABUF26	U7.24,U8.24,U9.24, U10.24,U6B.44B
AA07	NC, DEPOP	
AA08	NC	
AA09	NC, DEPOP	
AA10	DP3	U1.42,U3.34,U8.20
AA11	NC, DEPOP	
AA12	NC	
AA13	NC, DEPOP	
AA14	C2P_WRL_OPEN	U2.61,U3.154
AA15	NC, DEPOP	
AA16	TA#	R1.C05,U1.155, U2.111
AA17	NC, DEPOP	
AA18	APE_60X#	R1.F03,U1.218
AA19	NC, DEPOP	
AA20	TAG_WE#	U2.114,U5.73U5.74
AA21	NC, DEPOP	
AA22	SRAM_OE#	U7.50,U8.50,U9.50, U10.50
AA23	NC, DEPOP	
AA24	A4	U1.176,U2.76,U5.39
AA25	NC, DEPOP	
AA26	NC	
AA27	NC, DEPOP	

Pin[1]	Net Name[2]	MCM Nodes[3]
AA28	A21	U1.17,U2.98,U5.14, U6B.12B
AA29	NC, DEPOP	
AA30	A29	U1.30,U2.106
AA31	NC, DEPOP	
AA32	MD53	U3.118
AA33	MD54	U3.119
AB01	PCI_IRDY#	U2.201,U3.167
AB02	VSS	
AB03	DAISY17	J1.Y03
AB04	VDD2	
AB05	ABUF13	U7.29,U8.29,U9.29, U10.29,U6B.26B
AB06	VSS	
AB07	D2	U1.113,U3.178, U7.38
AB08	VDD2	
AB09	DP0	U1.38,U3.196,U7.46
AB10	VSS	
AB11	DP5	U1.47,U3.95,U9.20
AB12	VDD2	
AB13	NC, DEPOP	
AB14	VSS	
AB15	NC, DEPOP	
AB16	VDD2	
AB17	NC, DEPOP	
AB18	VDD2	
AB19	NC, DEPOP	
AB20	VSS	
AB21	NC, DEPOP	
AB22	VDD2	
AB23	NC	
AB24	VSS	
AB25	A3	U1.3,U2.75,U5.37
AB26	VDD2	
AB27	A12	U1.169,U2.86,U5.63
AB28	VSS	
AB29	A20	U1.164,U2.97,U5.15, U6B.13B
AB30	VDD2	
AB31	MA10	U2.178
AB32	VSS	
AB33	MD52	U3.113
AC01	PCI_FRAME_664#	U2.200
AC02	PCI_TRDY#	U2.202,U3.168
AC03	NC, DEPOP	

Pin[1]	Net Name[2]	MCM Nodes[3]
AC04	TAG_A_IN	U6A.2A,U6A.3A, U6A.5A,U6A.6A, U6A.8A,U6A.13A, U6A.14A,U6A.16A, U6A.17A,U6A.19A, U6A.20A,U6A.22A, U6A.23A
AC05	NC, DEPOP	
AC06	ABUF25	U7.23,U8.23,U9.23, U10.23,U6B.43B
AC07	NC, DEPOP	
AC08	D3	U1.110,U3.179, U7.39
AC09	NC, DEPOP	
AC10	DP2	U1.41,U3.3,U8.46
AC11	NC, DEPOP	
AC12	NC	
AC13	NC, DEPOP	
AC14	MEM_WRL_OPEN	U2.51,U3.150
AC15	NC, DEPOP	
AC16	NC	
AC17	NC, DEPOP	
AC18	DPE_60X#	R1.F02,U1.217, U2.133
AC19	NC, DEPOP	
AC20	TAG_VALID	U2.115,U5.80
AC21	NC, DEPOP	
AC22	TAG_CLEAR#	U2.116,U5.70
AC23	NC, DEPOP	
AC24	A2	U1.178,U2.74,U5.35
AC25	NC, DEPOP	
AC26	A11	U1.11,U2.85,U5.61
AC27	NC, DEPOP	
AC28	A19	U1.16,U2.96,U5.16, U6B.14B
AC29	NC, DEPOP	
AC30	A28	U1.151U2.105, U6B.2B
AC31	NC, DEPOP	
AC32	MD50	U3.111
AC33	MD51	U3.112
AD01	IGN_PCI_AD31	U2.57
AD02	VDD2	
AD03	DAISY16	J1.AF03
AD04	VSS	
AD05	D5	U1.108,U3.186, U7.41
AD06	VDD2	
AD07	D11	U1.92,U3.207,U7.13
AD08	VSS	
AD09	D17	U1.84,U3.221,U8.35

Pin[1]	Net Name[2]	MCM Nodes[3]
AD10	VDD2	
AD11	D23	U1.76,U3.2,U8.45
AD12	VSS	
AD13	D29	U1.68,U3.31,U8.15
AD14	VDD2	
AD15	D35	U1.139,U3.57,U9.39
AD16	VSS	
AD17	D41	U1.129,U3.79,U9.9
AD18	VSS	
AD19	D47	U1.118,U3.94,U9.19
AD20	VDD2	
AD21	D53	U1.101,U3.107, U10.41
AD22	VSS	
AD23	D59	U1.57,U3.126, U10.13
AD24	VDD2	
AD25	A1	U1.2 ,U2.73
AD26	VSS	
AD27	NC	
AD28	VDD2	
AD29	SRAM_CS#	U7.5,U8.5,U9.5, U10.5
AD30	VSS	
AD31	MA9	U2.179
AD32	VDD2	
AD33	MD49	U3.109
AE01	USER_PCICLK1	U4.16
AE02	PCI_LOCK#	U2.53
AE03	NC, DEPOP	
AE04	D4	U1.109,U3.185, U7.40
AE05	NC, DEPOP	
AE06	D10	U1.93,U3.199,U7.12
AE07	NC, DEPOP	
AE08	D16	U1.85,U3.220,U8.34
AE09	NC, DEPOP	
AE10	D22	U1.78,U3.1,U8.44
AE11	NC, DEPOP	
AE12	D28	U1.71,U3.27,U8.14
AE13	NC, DEPOP	
AE14	D34	U1.140,U3.56,U9.38
AE15	NC, DEPOP	
AE16	D40	U1.130,U3.78,U9.8
AE17	NC, DEPOP	
AE18	D46	U1.119,U3.89,U9.18
AE19	NC, DEPOP	
AE20	D52	U1.102,U3.106, U10.40

Pin[1]	Net Name[2]	MCM Nodes[3]
AE21	NC, DEPOP	
AE22	D58	U1.56,U3.125, U10.12
AE23	NC, DEPOP	
AE24	A0	U1.179,U2.72
AE25	NC, DEPOP	
AE26	NC	
AE27	NC, DEPOP	
AE28	AP3	R1.B04,U1.226
AE29	NC, DEPOP	
AE30	SRAM_BCLK1	U4.46
AE31	NC, DEPOP	
AE32	MDP5	U3.234
AE33	MD48	U3.108
AF01	664_PCI_REQ#	U2.58
AF02	VSS	
AF03	DAISY16	J1.AD03
AF04	VDD2	
AF05	X_SRAM_BCLK2	U9.51
AF06	VSS	
AF07	D9	U1.94,U3.198,U7.9
AF08	VDD2	
AF09	D15	U1.87,U3.218,U7.19
AF10	VSS	
AF11	D21	U1.80,U3.229,U8.41
AF12	VDD2	
AF13	D27	U1.72,U3.26,U8.13
AF14	VSS	
AF15	D33	U1.141,U3.55,U9.35
AF16	VDD2	
AF17	D39	U1.131,U3.70 ,U9.45
AF18	VDD2	
AF19	D45	U1.123,U3.88 ,U9.15
AF20	VSS	
AF21	D51	U1.105,U3.105, U10.39
AF22	VDD2	
AF23	D57	U1.55,U3.124,U10.9
AF24	VSS	
AF25	D63	U1.64,U3.136, U10.19
AF26	VDD2	
AF27	NC	
AF28	VSS	
AF29	X_SRAM_BCLK1	U8.51
AF30	VDD2	
AF31	MA8	U2.180
AF32	VSS	

Pin[1]	Net Name[2]	MCM Nodes[3]
AF33	MD47	U3.102
AG01	BR_MASTER#	R1.H05,U2.128
AG02	CPU_GNT_664#	U2.54
AG03	NC, DEPOP	
AG04	SRAM_BCLK2	U4.48
AG05	NC, DEPOP	
AG06	D8	U1.97,U3.197,U7.8
AG07	NC, DEPOP	
AG08	D14	U1.89,U3.210,U7.18
AG09	NC, DEPOP	
AG10	D20	U1.81,U3.228,U8.40
AG11	NC, DEPOP	
AG12	D26	U1.73,U3.25,U8.12
AG13	NC, DEPOP	
AG14	D32	U1.143,U3.54,U9.34
AG15	NC, DEPOP	
AG16	D38	U1.133,U3.69,U9.44
AG17	NC, DEPOP	
AG18	D44	U1.124,U3.87,U9.14
AG19	NC, DEPOP	
AG20	D50	U1.106,U3.104, U10.38
AG21	NC, DEPOP	
AG22	D56	U1.52,U3.117,U10.8
AG23	NC, DEPOP	
AG24	D62	U1.63,U3.135, U10.18
AG25	NC, DEPOP	
AG26	NC	
AG27	NC, DEPOP	
AG28	NC	
AG29	NC, DEPOP	
AG30	AP2	R1.B05,U1.227
AG31	NC, DEPOP	
AG32	MD45	U3.100
AG33	MD46	U3.101
AH01	PCI_PAR	U2.7
AH02	VDD2	
AH03	NC	
AH04	VSS	
AH05	D1	U1.114,U3.177, U7.35
AH06	VDD2	
AH07	D7	U1.98,U3.188,U7.45
AH08	VSS	
AH09	D13	U1.90,U3.209,U7.15
AH10	VDD2	
AH11	D19	U1.82,U3.227, U8.39

Pin[1]	Net Name[2]	MCM Nodes[3]
AH12	VSS	
AH13	D25	U1.74,U3.24,U8.9
AH14	VDD2	
AH15	D31	U1.66,U3.33,U8.19
AH16	VSS	
AH17	D37	U1.134,U3.68, U9.41
AH18	VSS	
AH19	D43	U1.125,U3.86,U9.13
AH20	VDD2	
AH21	D49	U1.107,U3.99,U10.35
AH22	VSS	
AH23	D55	U1.51,U3.115,U10.45
AH24	VDD2	
AH25	D61	U1.62,U3.134,U10.15
AH26	VSS	
AH27	NC	
AH28	VDD2	
AH29	AP1	R1.C01, U1.230
AH30	VSS	
AH31	MA7	U2.181
AH32	VDD2	
AH33	MD44	U3.93
AJ01	DRTRY#	R1.H01, U1.156
AJ02	CKSTP_OUT#	R1.J01, U1.216
AJ03	NC, DEPOP	
AJ04	D0	U1.115,U3.176,U7.34
AJ05	NC, DEPOP	
AJ06	D6	U1.99,U3.187,U7.44
AJ07	NC, DEPOP	
AJ08	D12	U1.91,U3.208,U7.14
AJ09	NC, DEPOP	
AJ10	D18	U1.83,U3.226,U8.38
AJ11	NC, DEPOP	
AJ12	D24	U1.75,U3.4,U8.8
AJ13	NC, DEPOP	
AJ14	D30	U1.67,U3.32,U8.18
AJ15	NC, DEPOP	
AJ16	D36	U1.135,U3.67,U9.40
AJ17	NC, DEPOP	
AJ18	D42	U1.126,U3.80,U9.12
AJ19	NC, DEPOP	
AJ20	D48	U1.117,U3.98,U10.34
AJ21	NC, DEPOP	
AJ22	D54	U1.100,U3.114,U10.44
AJ23	NC, DEPOP	
AJ24	D60	U1.58,U3.127,U10.14

Pin[1]	Net Name[2]	MCM Nodes[3]
AJ25	NC, DEPOP	
AJ26	NC	
AJ27	NC, DEPOP	
AJ28	NC	
AJ29	NC, DEPOP	
AJ30	AP0	R1.C02,U1.231
AJ31	NC, DEPOP	
AJ32	MD42	U3.91
AJ33	MD43	U3.92
AK01	NC	
AK02	VSS	
AK03	SRAM_ADSP#	U7.1,U8.1,U9.1,U10.1
AK04	VDD2	
AK05	DAISY11	J1.AK07
AK06	VSS	
AK07	DAISY11	J1.AK05
AK08	VDD2	
AK09	DAISY12	J1.AK11
AK10	VSS	
AK11	DAISY12	J1.AK09
AK12	VDD2	
AK13	DAISY13	J1.AK15
AK14	VSS	
AK15	DAISY13	J1.AK13
AK16	VDD2	
AK17	DAISY14	J1.AK19
AK18	VDD2	
AK19	DAISY14	J1.AK17
AK20	VSS	
AK21	DAISY15	J1.AK23
AK22	VDD2	
AK23	DAISY15	J1.AK21
AK24	VSS	
AK25	NC	
AK26	VDD2	
AK27	NC	
AK28	VSS	
AK29	NC	
AK30	VDD2	
AK31	MA6	U2.184
AK32	VSS	
AK33	MD41	U3.90
AL01	L2_CLAIM#	U2.132
AL02	SMI#	R1.F05,U1.187
AL03	NC, DEPOP	
AL04	MRE0#	U2.164
AL05	NC, DEPOP	

Pin[1]	Net Name[2]	MCM Nodes[3]
AL06	MRE1#	U2.163
AL07	NC, DEPOP	
AL08	MRE2#	U2.162
AL09	NC, DEPOP	
AL10	MRE3#	U2.161
AL11	NC, DEPOP	
AL12	MRE4#	U2.160
AL13	NC, DEPOP	
AL14	MRE5#	U2.159
AL15	NC, DEPOP	
AL16	MRE6#	U2.158
AL17	NC, DEPOP	
AL18	MRE7#	U2.157
AL19	NC, DEPOP	
AL20	MA0	U2.190
AL21	NC, DEPOP	
AL22	MA1	U2.189
AL23	NC, DEPOP	
AL24	MA2	U2.188
AL25	NC, DEPOP	
AL26	MA3	U2.187
AL27	NC, DEPOP	
AL28	MA4	U2.186
AL29	NC, DEPOP	
AL30	MA5	U2.185
AL31	NC, DEPOP	
AL32	MDP4	U3.37
AL33	MD40	U3.83
AM02	VDD2	
AM03	MD1	U3.182
AM04	VSS	
AM05	MD4	U3.189
AM06	VDD2	
AM07	MD7	U3.194
AM08	VSS	
AM09	MD9	U3.201
AM10	VDD2	
AM11	MD12	U3.206
AM12	VSS	
AM13	MD15	U3.213
AM14	VDD2	
AM15	MD17	U3.222
AM16	VSS	
AM17	MD20	U3.225
AM18	VSS	
AM19	MD23	U3.233
AM20	VDD2	

Pin[1]	Net Name[2]	MCM Nodes[3]
AM21	MD25	U3.6
AM22	VSS	
AM23	MD28	U3.29
AM24	VDD2	
AM25	MD31	U3.36
AM26	VSS	
AM27	MD33	U3.59
AM28	VDD2	
AM29	MD36	U3.74
AM30	VSS	
AM31	MD39	U3.81
AM32	VDD2	
AN03	MD0	U3.180
AN04	MD2	U3.183
AN05	MD3	U3.184
AN06	MD5	U3.190
AN07	MD6	U3.193
AN08	MDP0	U3.141
AN09	MD8	U3.200
AN10	MD10	U3.202
AN11	MD11	U3.203

Pin[1]	Net Name[2]	MCM Nodes[3]
AN12	MD13	U3.211
AN13	MD14	U3.212
AN14	MDP1	U3.122
AN15	MD16	U3.215
AN16	MD18	U3.223
AN17	MD19	U3.224
AN18	MD21	U3.231
AN19	MD22	U3.232
AN20	MDP2	U3.103
AN21	MD24	U3.5
AN22	MD26	U3.7
AN23	MD27	U3.28
AN24	MD29	U3.30
AN25	MD30	U3.35
AN26	MDP3	U3.82
AN27	MD32	U3.58
AN28	MD34	U3.60
AN29	MD35	U3.73
AN30	MD37	U3.75
AN31	MD38	U3.76

Notes:

1.1077 pins are listed.

2.The net name is the net name from the MCM schematics (see Section 13). Use it when referring to a net. Many pin sites are not populated with a solder column. These sites are indicated with DEPOP in the Net Name column. NC in the Net Name column means that the MCM makes no connection to that site.

3.The MCM Nodes indicate the pins of the devices that the MCM pin is connected to. For example, U2.24 represents pin 24 of device U2.

Section 3
CPU and Level Two Cache

This section discusses topics that are directly related to the CPU, CPU bus, and Level Two (L2) cache, including how the 660 bridge decodes CPU initiated transfers as a function of the transfer type and address range. For more information, refer to the *660 User's Manual*.

The 100 MHz PPC 603e MCM supports CPU bus speeds up to 66MHz, and PCI bus speeds up to 33MHz. The MCM is initially configured with a CPU_core:CPU_bus:PCI bus frequency ratio of 99:66:33 MHz. This accomplished by setting the configuration bits for the MPC970 clock driver via off MCM pullups.

3.1 CPU Busmasters

The MCM uses a single PowerPC 603e CPU, and an internal L2 cache; thus, there are only two busmasters on the CPU bus: the CPU and the 660. CPU bus arbitration is greatly simplified, and the multi-processor capabilities of the 660 are not used. The remaining arbitration on the CPU bus is between the CPU and the snoop broadcasting logic in the 660. Since the 660 parks the CPU bus on CPU1 (the 603e) whenever the bus is idle, CPU latency is minimized.

One level of address bus pipelining is supported, and most data writes are posted. Precise exceptions are reported via TEA#, and imprecise exceptions are reported via MCP#. PIO, or programmed I/O transactions (XATS# type) are not supported.

The MCM is not currently specified for use with external CPU busmasters. For more information on the CPU bus protocol, see the 603e User's Manual. For more information on the multiprocessor capabilities of the 660, see the *660 User's Manual*. For MCM applications involving external CPU bus agents, please contact IBM PowerPC Embedded Processor Systems Application Engineering.

3.1.1 603e CPU

The MCM operates the 603e in 64-bit data bus mode. The 660 will not operate in 32-bit mode.

The MCM is configured for DRTRY# mode, and should not be configured for no-DRTRY# mode. In DRTRY# mode, data is assumed to have been speculatively presented to the CPU, and so is held for one clock in the 603e BIU before being presented to the CPU core data consumers.

The MCM runs at 3:2 603e internal clock to bus clock ratio at 99MHz:66MHz. CPU PLL_CFG[0:3] is set to 1100. The 603e may be run at 1:1 core:bus ratio, and the frequency of the CPU bus clock can be varied. See Section 7, Clocks, for more information.

3.2 System Response by CPU Bus Transfer Type

All access to the rest of the system is provided to the CPU by the 660. 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 one of the following operations:

- 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.

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

TT[0:3]	60X Operation	60X Bus Transaction	660 Operation For CPU to Memory Transfers	660 Operation For CPU to PCI Transactions
0000	Clean block or lwarx	Address only	Asserts AACK#. No other response. No PCI transaction.	
0001	Write with flush	SBW(1) 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(1) 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.

Note:

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.

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.

References in the remainder of this document to a CPU read, assume one of the transfer types in Table 3-1 that produce the read response from the 660. Likewise, references to a CPU write refer to those transfer type that produce the write response.

3.3 System Response by CPU Bus Address Range

The 660 determines the target of a CPU busmaster transaction based on the CPU bus address range as shown in Table 3-2. The acronym BCR means bridge control register.

Table 3-2. 660 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 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 0000 0000h to 007F FFFFh	(3)
	Non-Contiguous Mode		0 to 64K 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 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 FFDF FFFFh		PCI Memory Transaction	0 to 1G – 2M 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 0000 0000h to 001F FFFFh (ROM Address Space)	(2)
	Remote ROM	PCI Memory Transaction to I/O Bus Bridge	1G – 2M to 1G 3FE0 0000h to 3FFF FFFFh	(2)

Notes:
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 the 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.

3.3.1 Address Mapping for Non-Contiguous I/O

Figure 3-1 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 60X bus is compressed into 64K of PCI address space, and the 60X CPU cannot create PCI I/O addresses from 64K to 8M.

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 60X 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.

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.

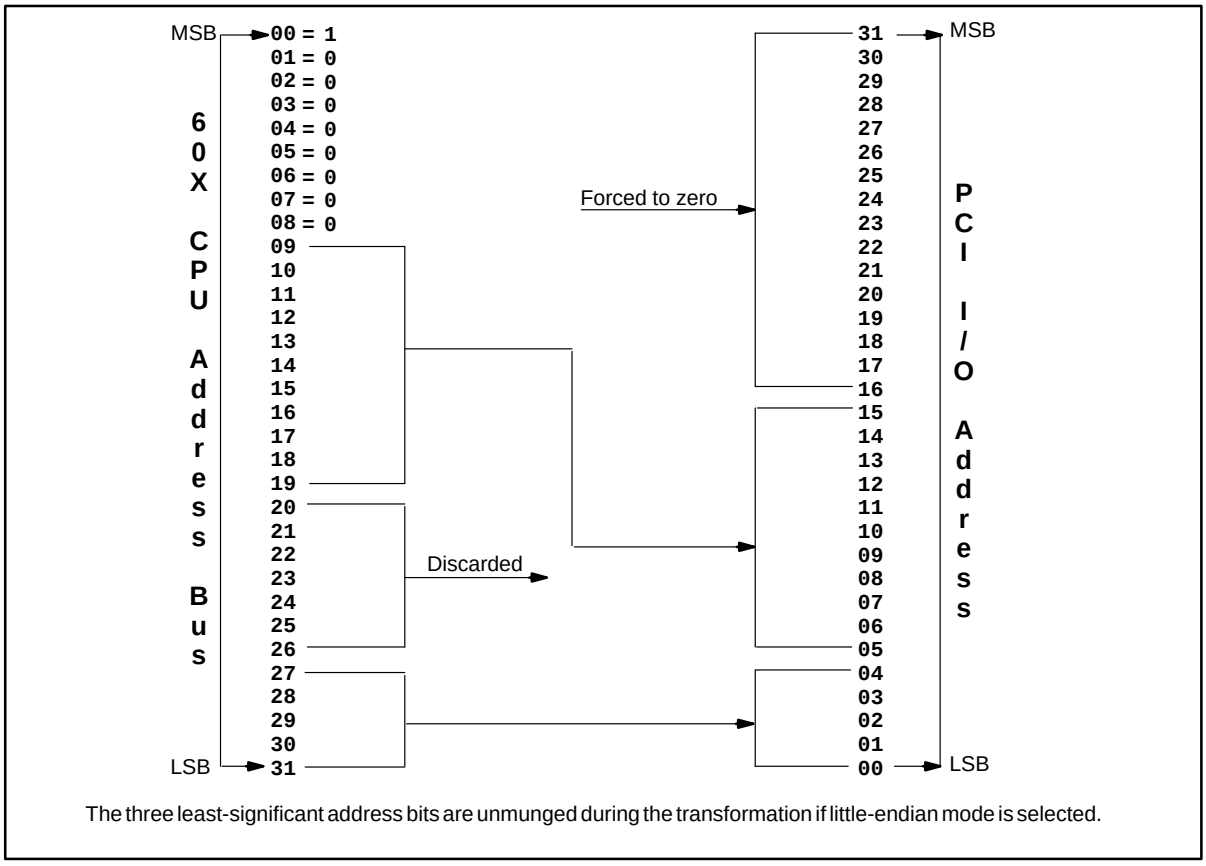


Figure 3-1. Non-Contiguous PCI I/O Address Transformation

For example, in Figure 3-2, 60X 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 60X CPU address 8000 1000h.

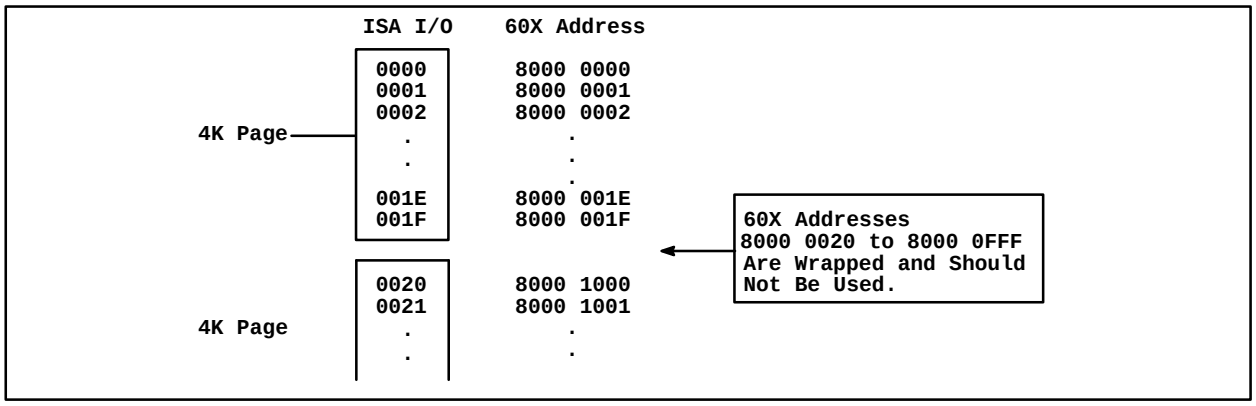


Figure 3-2. Non-Contiguous PCI I/O Address Translation

3.3.2 Address Mapping for Contiguous I/O

In contiguous I/O mode, 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 may 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 3-3 gives an example of contiguous I/O partitioning.

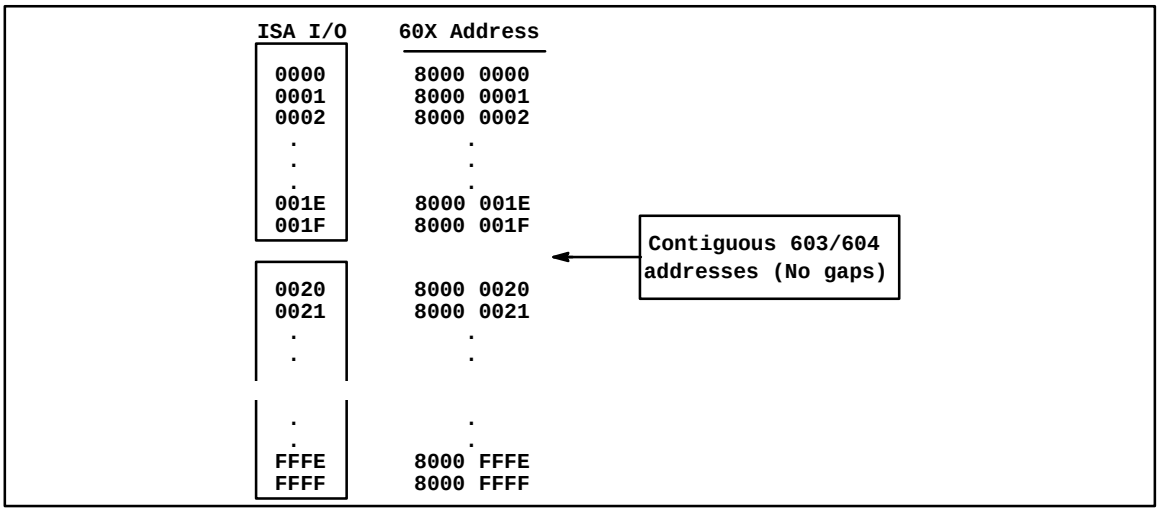


Figure 3-3. Contiguous PCI I/O Address Translation

3.3.3 PCI Final Address Formation

The 660 maps 60X CPU bus addresses from 2G to 4G as PCI transactions, error address register reads, or ROM reads and writes. The 660 manipulates 60X bus addresses from 2G to 4G to generate PCI addresses as follows:

- PCI_AD[31:30] are set to zero.
- PCI_AD[2:0] are unmunged if little-endian mode is selected.
- After unmunging, PCI_AD[1:0] are set to 00b except for PCI I/O cycles.

3.4 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 660 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 603/604 CPU.

3.4.1 LE Mode

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 CPU will take an alignment exception). Also, load/store multiple word and load/store string word instructions are not supported in the CPU in LE mode.

3.5 CPU to PCI Transactions

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 be sensitive to order of access also. See the 660 User's Manual.

All addresses from 2G to 4G (including ROM space) must be marked non-cacheable. See the PowerPC Reference Platform Specification. The MCM supports all PCI bus protocols during CPU to PCI transactions.

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

When the 660 decodes a CPU access as targeted for the PCI, the 660 requests the PCI bus. Once the SIO grants the PCI bus to the 660, the bridge initiates the PCI cycle and releases the bus.

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).

3.5.1 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).

3.5.2 CPU to PCI Write

If the CPU to PCI cycle is a write, a PCI write cycle is run. CPU to PCI 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.

CPU to PCI memory writes are posted, so the CPU write cycle is ended as soon as the data is latched. If the PCI cycle is retried, the Bridge retries the cycle until it completes.

3.5.2.1 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. 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 4-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.

3.5.3 CPU to PCI Memory Transactions

CPU transfers from 3G to 4G–2M are mapped to the PCI bus as memory transactions.

3.5.4 CPU to PCI I/O Transactions

CPU transfers from 2G+16M to 3G–8M are mapped to the PCI bus as I/O transactions. In compliance with the PCI specification, the 660 master aborts all I/O transactions that are not claimed by a PCI agent.

3.5.5 CPU to PCI Configuration Transactions

The MCM allows the CPU to generate type 0 and type 1 PCI configuration cycles. The CPU initiates a transfer to the appropriate address. The 660 decodes the cycle and generates a request to the PCI arbiter in the SIO. When the PCI bus is acquired, the 660 enables its PCI_AD drivers and drives the address onto the PCI_AD lines for one PCI clock before it asserts PCI_FRAME#. Predriving the PCI_AD lines for one clock before asserting PCI_FRAME# allows the IDSELs to be resistively connected to the PCI_AD[31:0] bus at the system level.

The transfer size must match the capabilities of the target PCI device for configuration cycles. The MCM supports 1-, 2-, 3-, and 4-byte transfers that do not cross a 4-byte boundary.

Address unmunging and data byte swapping follow the same rules as those for system memory with respect to BE and LE modes of operation. Address unmunging has no effect on the CPU address lines which correspond to the IDSEL inputs of the PCI devices.

The generation of PCI configuration cycles is via the 660 indexed Bridge Control Registers (BCR). This configuration method is described in section 4 of the 660 User's Manual. The IDSEL assignments and their respective PCI_AD lines are shown in Table 3-3. The addresses used for configuration are assigned as shown in Table 3-3.

Table 3-3. CPU to PCI Configuration Mapping

Device	IDSEL Line	60X Address (1)	PCI Address
PCI to ISA Bridge	A/D 11	8080 08XXh	080 08XX
PCI slot 1	A/D 12	8080 10XXh	080 10XX
PCI slot 2	A/D 13	8080 20XXh	080 20XX
PCI slot 3	A/D 14	8080 40XXh	080 40XX
PCI slot 4	A/D 15	8080 80XXh	080 80XX
PCI slot 5	A/D 16	8081 00XXh	081 00XX
PCI slot 6	A/D 17	8082 00XXh	082 00XX
PCI slot 7	A/D 18	8084 00XXh	084 00XX
PCI slot 8	A/D 19	8088 00XXh	088 00XX
PCI slot 9	A/D 20	8090 00XXh	090 00XX
PCI slot 10	A/D 21	80A0 00XXh	0A0 00XX
PCI slot 11	A/D 22	80C0 00XXh	0C0 00XX

Notes:
1. This address is independent of contiguous I/O mode.

3.5.6 CPU to PCI Interrupt Acknowledge Transaction

Reading the interrupt acknowledge address (BFFF FFF0h) causes the bridge to arbitrate for the PCI bus and then to execute a standard PCI interrupt acknowledge transaction. The system interrupt controller in the ISA bridge claims the transaction and supplies the 1-byte interrupt vector. There is no physical interrupt vector BCR in the bridge. Other PCI busmasters can initiate interrupt acknowledge transactions, but this may have unpredictable effects.

3.5.7 PCI Lock

The 660 does not set PCI locks when acting as the PCI master. The PCI_LOCK# signal in the 660 supports resource locking of one 32-byte cache sector (block) of system memory. Once a PCI lock is established, the block address is saved. Subsequent accesses to that block from other PCI busmasters or from the CPU bus are retried until the lock is released.

The bridge generates a flush-sector snoop cycle on the CPU bus when a PCI busmaster sets the PCI lock. The flush-sector 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, the PCI cycle is retried and the modified data is pushed out to memory.

Note: 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. To work with the lwarx and stwcx instructions, the 660 generates a flush-sector operation to the CPU in response to the PCI read that begins a PCI lock.

3.6 CPU to BCR Transfers

The 660 can be extensively programmed by means of the Bridge Control Registers (BCR). See the 660 User's Manual for a description of the operation and programming of the 660 BCRs.

3.7 L2

The MCM contains an L2 cache controller (located inside the 660), 512K of synchronous SRAM, and a 16K x 15 synchronous tagRAM. This forms a unified, write-thru, direct-mapped, look-aside level 2 cache (L2) that caches CPU memory space from 0 to 1G.

The L2 directory contains 8K entries with one tag per entry. The cache line (block) size is 32 bytes, the PowerPC 60X coherence unit. The L2 maintains coherence for the 32-byte block and neither operates on nor maintains the status of smaller units of memory.

Typical L2 read performance is 3-1-1-1, followed by -2-1-1-1 on pipelined reads. For more information on the operation and capabilities of the L2, see the 660 User's Manual.

3.7.1 Cache Response to CPU Bus

The L2 supplies data to the CPU bus on burst read hits and snarfs the data (updates the SRAM data while the memory controller is accessing the DRAM) on CPU burst read/write misses. It snoops PCI to memory transactions, and it invalidates on PCI write hits. The L2 does not supply data to the PCI on read hits.

Table 3-4. 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	Snarf	Snarf
0010		Flush sector	Invalidate	Ignore
0011	Single	Write with kill	Invalidate	Ignore
	Burst	Write with kill	Snarf	Snarf
0101	Single	Read	Ignore	Ignore
	Burst	Read	Claim cycle and supply data	Snarf
0110		Kill sector	Invalidate	Ignore
0111	Always Burst	RWITM	Claim cycle and supply data	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	Snarf
1110		External control in	Ignore	Ignore
1111	Always Burst	RWITM Atomic	Claim cycle and supply data	Ignore

The L2 ignores CPU bus single-beat reads, and invalidates on CPU bus single-beat write hits. Table 3-4 shows the actions taken by the L2 cache based on transfer type and single-beat or burst mode.

The CPU cache inhibit (CI#) signal is not used because cache-inhibited bus operations are always single-beat. The 660 does not use TT[4]. Accesses to populated memory are snooped by L2 regardless of the state of GBL#. The 660 only uses GBL# as an output.

3.7.2 Cache Response to PCI Bus

The L2 maintains coherency during PCI to memory transactions as shown in Table 3-5. The L2 does not supply data to the PCI bus. The L2 is not updated during a PCI transaction.

Table 3-5. L2 Operations for PCI to Memory Transactions, 603 Mode

PCI Bus Transaction	CPU Bus Broadcast Snoop		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

3.7.3 L2 Configuration

The L2 controller does not require any software configuration to set the size or organization of the tag and data SRAM. The connection to the CPU address bus determines the size of the tag RAM and data SRAM. Bridge Control Register D4 bit 3 should be set to 1 to select ”burst” as the SRAM type.

3.7.4 L2 Organization

The L2 directory contains 8K entries with one tag per entry. The cache line (block) size is 32 bytes, the PowerPC 60X coherence unit. The L2 maintains coherence for the 32-byte block and neither operates on nor maintains the status of smaller units of memory. All of the tags together are sometimes called the cache directory.

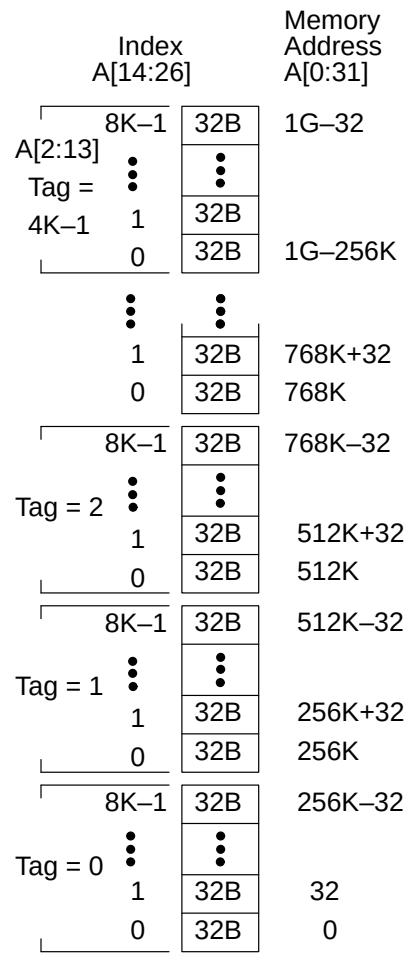
The L2 uses only A[2:31] to store and access L2 data. A[0:1] are not used or saved in the cache directory because the MCM only caches the lowest 1G of the address space. A[0:1] must equal 00 for the MCM to access the directory.

3.7.5 Other L2 Related BCRs

See Table 3-6.

Table 3-6. Other L2 Related BCRs

Bridge Control Register	Index	R/W	Bytes
Error Enable 2	Index C4	R/W	1
Error Status 2	Index C5	R/W	1
Bridge Chip Set Options 3	Index D4	R/W	1
System Control 81C	8000 081C	R/W	1
L2 Invalidate	8000 0814	R	
L2 Error Status	8000 0842	R	
L2 Parity Error Read and Clear	8000 0843	R	
Cache Status	Index B1h	R/W	



The L2 considers the 1G of cacheable system memory to be logically organized into 4K pages. Every member of a given page has the same address tag, which in this case is defined as bits A[2:13] of the address.

Each page consists of 8K blocks of memory, where each block consists of one 32-byte doubleword of memory. A block is referred to within the page by the index, which is defined as bits A[14:26] of the address.

Thus each block of memory in the 0 to 1G range has a tag, A[2:13], which ranges from 0 to 4K–1, and an index, A[14:26], which ranges from 0 to 8K–1. All of the blocks that have the same index are said to be in the same congruence class, or set. Each block in a given set has a unique tag. Some mapping examples:

Line Memory Address	Index	Tag
0	0	0
256K	0	1
256K+32	1	1
768K–32	8K–1	2

Figure 3-4. L2 Mapping of System Memory – 512K Configuration

Figure 3-5 shows how the MCM maps the SRAM to main memory using the index and tag fields of the address. Notice that there are 8k tags and 8k 32-byte blocks in the SRAM, and that the main memory is divided up into 4k pages, each one of which is composed of 8k blocks.

When an address is presented to the cache directory for snooping, the MCM uses the index to select which directory location to access, by presenting A[14:26] to the tagRAM address inputs. The tagRAM then compares the tag in that location (the A[2:13] of the previous cacheable access to that location) to the tag (A[2:13]) of the current transaction. If there is a match, and the tag is marked ”valid,” then there is a cache hit (signalled by TAG_MATCH).

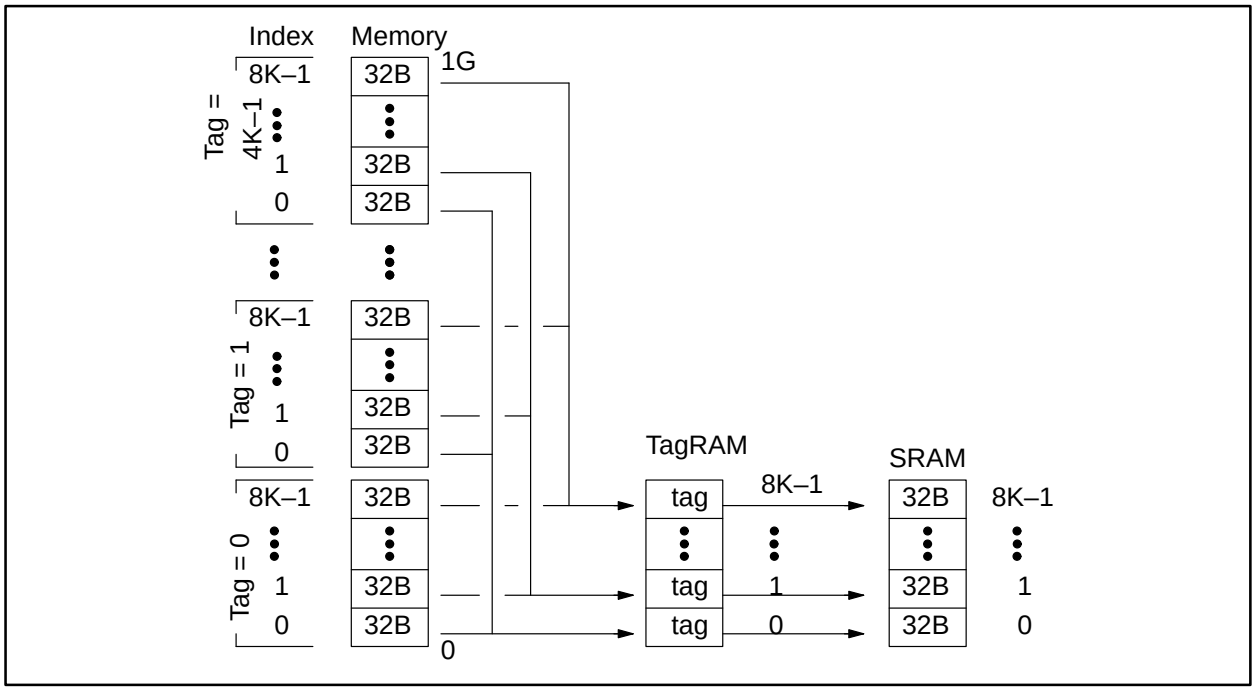


Figure 3-5. SLC L2 Cache Directory – 512K Configuration

Suppose the CPU requests a burst store to location 512K (8 0000h), which is initially invalid (either stale or never accessed). The index is A[14:26], which is 0h, so the accessed tagRAM location is 0h. The tag currently on the address bus is A[2:13], which equals 2. So the MCM stores 2h into tagRAM location 0h. The valid bit is set for that location.

While the CPU is accessing other blocks of memory with different low order addresses, other locations in the tagRAM are being accessed; however, if the CPU again accesses a block of memory with this same low order address, then this tagRAM location will again be accessed, and the tag stored therein will be compared against A[2:13] of the current access. If they are the same, then there is a cache hit.

3.7.6 SRAM

The MCM uses four IBM041814 synchronous 64K x 18 SRAMs to implement the SRAM portion of the L2. Figure 3-6 shows the basic connectivity of the MCM SRAM. These are synchronous devices, and each SRAM consumes one of the MPC970 clocks. In burst operation, the address of the data for the initial beat of the burst is latched into the SRAM; the address of the data for the next beat of the burst is incremented internally under the control of the ADV# input (SRAM_CNT_EN#).

- Each SRAM is connected to two CPU bus data bytes and the associated two CPU bus parity lines.
- ABUF[13:28] are a buffered copy of CPU A[13:28]. The SRAMs are arranged in parallel, so that an 8-byte doubleword is addressed during each access; thus, ABUF[28] is connected to A0 of the SRAM.
- The 660 asserts SRAM_WE# to write into the SRAM and asserts SRAM_OE# to read data out of the SRAM.
- The 660 asserts SRAM_ADS# to signal the initial beat of the burst.

- Pull SRAM_CS# low for normal operation. Pull SRAM_ADSP# high for normal operation.

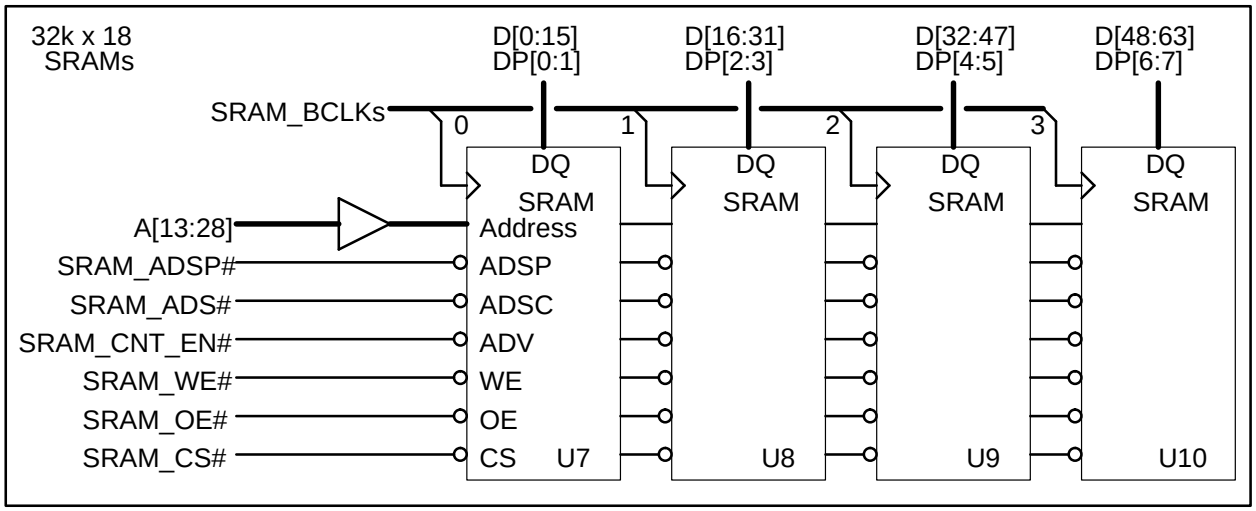


Figure 3-6. Synchronous SRAM, 512K L2

3.7.7 TagRAM

The MCM uses an IDT71216 synchronous 16K x 15 cache tagRAM to implement the tags of the L2. Figure 3-7 shows the basic connectivity of the MCM SRAM. The tagRAM is a synchronous device, and so consumes one of the MPC970 clocks.

- CPU address lines A[14:26] form the index of the directory entry. Pull TAG_ADDR_13 high for normal operation.
- CPU address lines A[2:13] form the tag of the directory entry. Tie TAG_DATA_11 to A13 for normal operation.
- During tagRAM writes, the valid bit associated with the index is set to match the TAG_VALID input.
- During tagRAM reads, the TAG_MATCH output is released to the active high (open–drain) state only when A[2:13] matches the contents of tagRAM location A[14:26], and the Valid bit for that location is set to 1. If the the Valid bit is 0 (invalid) or the address stored in the tag does not match the current value of A[2:13], then the TAG_MATCH output is driven low.

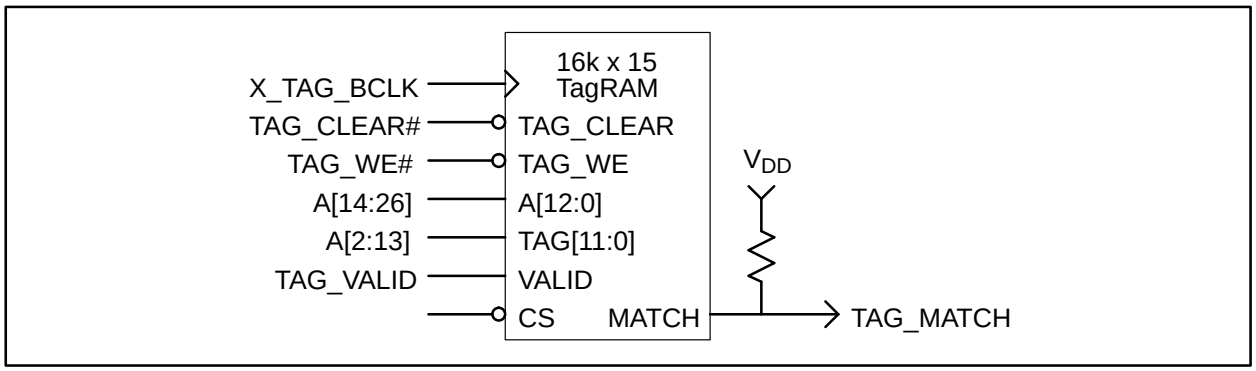


Figure 3-7. Synchronous TagRAM, 512K L2

Section 4
DRAM

The memory controller in the 660 bridge controls the system memory DRAM. The system memory can be accessed from both the CPU bus and the PCI bus. Much of the information in this section has been drawn from the 660 User's Manual.

4.1 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
- 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
- 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

4.1.1 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.

4.1.1.1 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	–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	–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

4.1.1.2 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, even 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

4.2 DRAM Performance

4.2.1 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 vari-

ous memory controller BCRs. Figure 4-1 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.

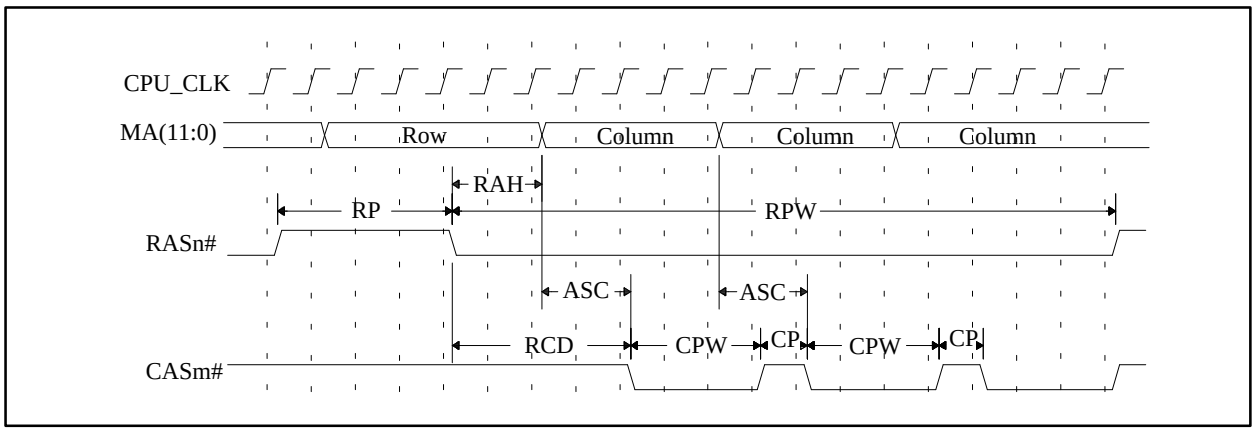


Figure 4-1. CPU to Memory Transfer Timing Parameters

Table 4-1 shows the function, location, and section references for the variables shown in Figure 4-1.

Table 4-1. Memory Timing Parameters

Variable	Function	BCR	Section
ASC	Column Address Setup (min)	Memory Timing Register 2	4.2.1.2
CP	CAS# Precharge	Memory Timing Register 2	4.2.1.2
CPW	CAS# Pulse Width (Read & Write)	Memory Timing Register 2	4.2.1.2
RAH	Row Address Hold (min)	Memory Timing Register 1	4.2.1.1
RCD	RAS# to CAS# Delay (min)	Memory Timing Register 2	4.2.1.2
RP	RAS# Precharge	Memory Timing Register 1	4.2.1.1
RPW	RAS# Pulse Width	Memory Timing Register 1	4.2.1.1

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.

4.2.1.1 Memory Timing Register 1

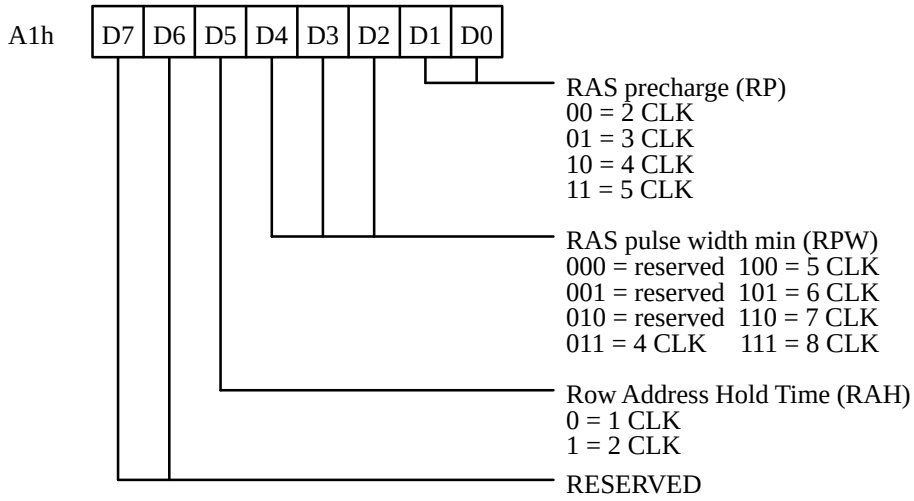
Index A1	Read/Write	Reset to 3Fh

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 4.2.1.

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.

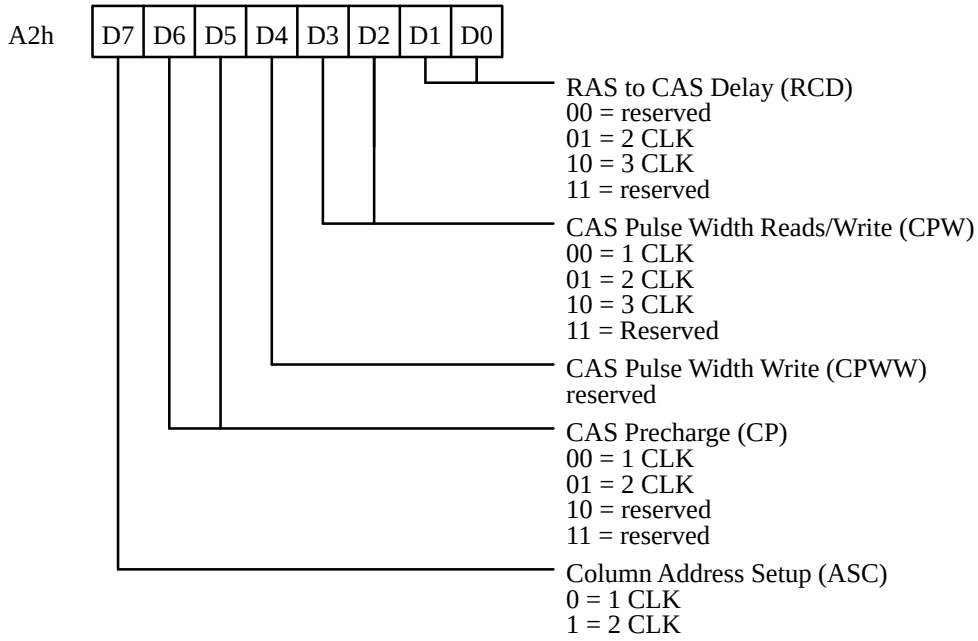
Bit 5 This bit controls the number of CPU clocks that the row address is held following the assertion of RAS#.



4.2.1.2 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 4.2.1.



4.2.1.3 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.

4.2.1.4 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 4.2.1 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), 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. The timing recommendations in this section apply to all MCM configurations. Because the MCM uses synchronous SRAM, it is not affected by the 660 DRAM special case timing restrictions.

Each of the nine equations below lists a register or register bits that govern 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 4-1 and Table 4-1.

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

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.

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].

4.2.1.5 DRAM Timing Examples

This section presents example DRAM timing calculations based on the equations found in Section 4.2.1.4. Except as noted in Section 4.2.1.4, 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.

4.2.1.6 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 values MTR1=32h, MTR2=0Eh 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).

4.2.1.7 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).

4.2.1.8 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).

4.2.1.9 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).

4.2.1.10 Aggressive Timing Summary

Table 4-2 contains a summary of recommended general case aggressive page mode DRAM timing, the required control register settings, and 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 top section of Table 4-2 shows the settings of the memory controller BCRs. The next 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 lowest 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 4-2. Page Mode DRAM Aggressive Timing Summary (1)

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

Notes:

1. Refresh Rate set to 0208h. RAS# watchdog timer BCR set to 53h.
2. 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#.
3. 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#.

Table 4-3 contains a summary of recommended general case aggressive EDO DRAM timing, the required control register settings, and the resulting performance of the memory controller.

Table 4-3. EDO DRAM Aggressive Timing Summary (1)

Transfer	70ns EDO Aggressive	60ns EDO Aggressive	50ns EDO Aggressive	Note
Memory Timing Register 1	32	11	2C	
Memory Timing Register 2	06	05	02	
Bridge Chipset Options 3	0C	0C	0C	
Initial Read Burst	11-3-3-3	10-3-3-3	10-2-2-2	
Initial Write Burst	5-4-3-3	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	
Page Hit Write	-3-3-3-3	-3-3-3-3	-3-2-2-2	
Page Miss & Bank Miss Read	-8-3-3-3	-7-3-3-3	-7-2-2-2	(2)
Page Miss & Bank Hit Read	-10-3-3-3	-8-3-3-3	-7-2-2-2	(3)
Page Miss & Bank Miss Write	-3-3-3-3	-3-3-3-3	-3-3-2-2	(2)
Page Miss & Bank Hit Write	-3-5-3-3	-3-3-3-3	-3-3-2-2	(3)
For a pipelined burst transfer immediately following a write:				
Page Hit Read	-9-3-3-3	-9-3-3-3	-7-2-2-2	
Page Hit Write	-5-3-3-3	-5-3-3-3	-4-2-2-2	
Page Miss & Bank Miss Read	-12-3-3-3	-11-3-3-3	-10-2-2-2	(2)
Page Miss & Bank Hit Read	-14-3-3-3	-12-3-3-3	-10-2-2-2	(3)
Page Miss & Bank Miss Write	-5-5-3-3	-5-4-3-3	-4-5-2-2	(2)
Page Miss & Bank Hit Write	-5-7-3-3	-5-5-3-3	-4-5-2-2	(3)

Notes: See Table 4-2 for notes.

4.2.1.11 Conservative Timing Summary

Table 4-4 contains a summary of recommended general case conservative page mode 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 4-4. Page Mode DRAM Conservative Timing Summary (1)

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

Notes: See Table 4-2 for notes.

Table 4-5 contains a summary of recommended general case aggressive EDO DRAM timing, the required control register settings, and the resulting performance of the memory controller.

Table 4-5. EDO DRAM Conservative Timing Summary (1)

Transfer	70ns EDO Conservative	60ns EDO Conservative	50ns EDO Conservative	Note
Memory Timing Register 1	32	31	0C	
Memory Timing Register 2	0A	06	05	
Bridge Chipset Options 3	0C	0C	0C	
Initial Read Burst	12-4-4-4	11-3-3-3	10-3-3-3	(2)
Initial Write Burst	5-4-4-4	5-4-3-3	5-3-3-3	
For a pipelined burst transfer immediately following a read:				
Page Hit Read	-5-4-4-4	-5-3-3-3	-5-3-3-3	(2)
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	-7-3-3-3	(1,3)
Page Miss & Bank Hit Read	-11-3-3-3	-9-3-3-3	-7-3-3-3	(1,4)
Page Miss & Bank Miss Write	-3-3-4-4	-3-3-3-3	-3-3-3-3	(3)
Page Miss & Bank Hit Write	-3-5-4-4	-3-4-3-3	-3-3-3-3	(4)
For a pipelined burst transfer immediately following a write:				
Page Hit Read	-12-4-4-4	-9-3-3-3	-9-3-3-3	(2)
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	-11-3-3-3	(2,3)
Page Miss & Bank Hit Read	-17-4-4-4	-13-3-3-3	-11-3-3-3	(2,4)
Page Miss & Bank Miss Write	-6-6-4-4	-5-5-3-3	-5-4-3-3	(3)
Page Miss & Bank Hit Write	-6-8-4-4	-5-6-3-3	-5-4-3-3	(4)

Notes: See Table 4-2 for notes.

4.2.1.12 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.

4.2.1.13 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.

4.2.1.14 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.

4.3 System Memory Addressing

4.3.1 DRAM Logical Organization

The DRAM system implemented by the 660 is logically arranged as shown in Figure 4-2. Each block shown in Figure 4-2 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# (aka MRE#) lines strobe in the row address. The CAS# (aka MCE#) 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.

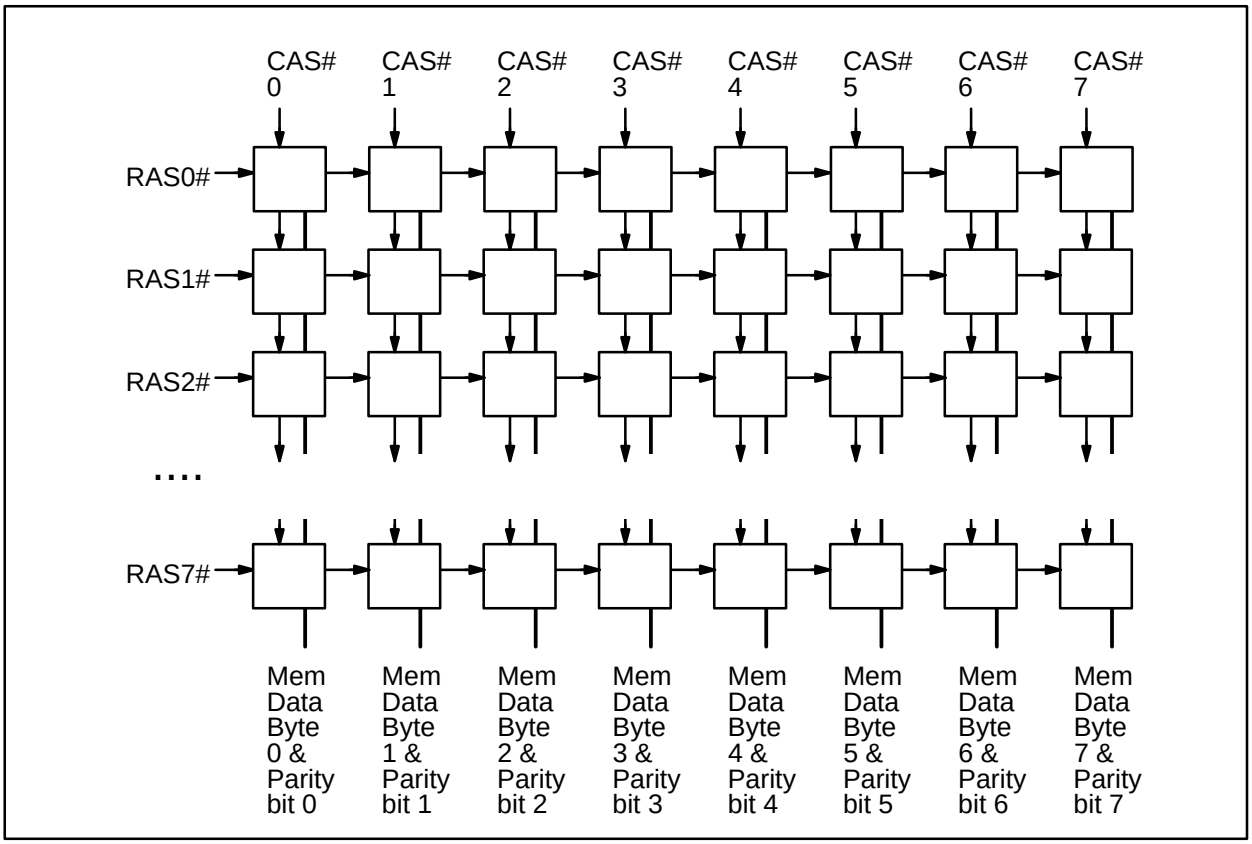


Figure 4-2. DRAM Logical Implementation

4.3.1.1 SIMM Topologies

Table 4-6 shows the various memory module (SIMM) topologies that the 660 supports. Each memory bank can be populated with any supported SIMM.

Table 4-6. 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:
1. See BCR(A4 to A7) in Section 4.3.1.6

The 660 supports the 168-pin 8-byte SIMMs shown in Table 4-6, 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 660 also supports the 72-pin 4-byte SIMMs shown in Table 4-6, 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.

4.3.1.2 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 4-7 and Table 4-8 show which CPU address bits are driven onto the memory address bus during CPU to memory transfers. Table 4-9 and Table 4-10 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 4.3.1.6). The addressing mode of each bank of memory is individually configurable.

Table 4-7. 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 4-8. 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 4-9. 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 4-10. 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.

4.3.1.3 DRAM Pages

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

4.3.1.4 Supported Transfer Sizes and Alignments

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

4.3.1.5 Unpopulated Memory Locations

Physical memory does not occupy the entire address space assigned to system memory in the memory map. While the Memory Select Error Enable bit = 0:

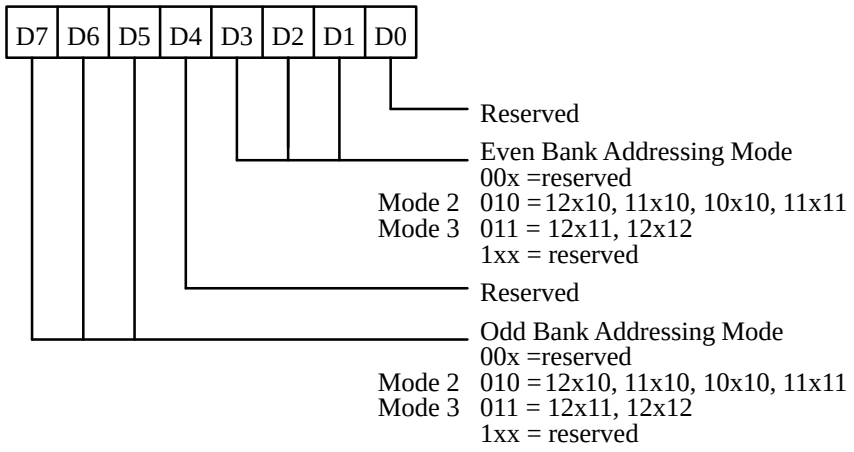
- When the CPU reads an unpopulated location, the bridge returns all-ones, and completes the transfer normally.
- When the CPU writes to an unpopulated location, the bridge signals normal transfer completion to the CPU, but does not write the data to memory.

The memory select error bit in the error status 1 register (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 the Memory Select Error bit =1, a read or write to an unpopulated memory location is reported to the initiator as an error.

4.3.1.6 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.



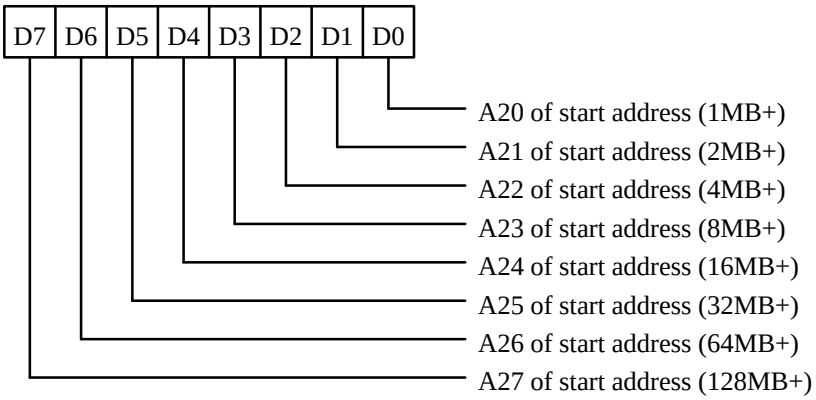
Register	Bits	Memory Bank	Bits	Memory Bank
A4h	3:0	0	7:4	1
A5h	3:0	2	7:4	3
A6h	3:0	4	7:4	5
A7h	3:0	6	7:4	7

4.3.1.7 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.

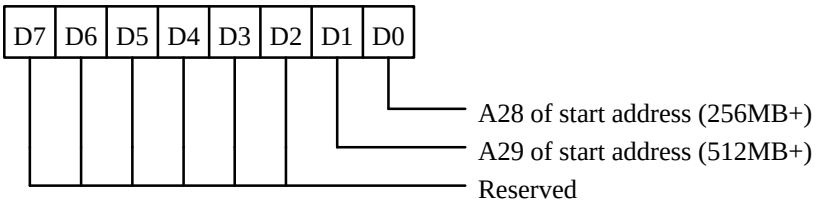
Program the banks in ascending order, such that (for n = 0 to 6) the starting address of bank n+1 is higher than the starting address of bank n. Each bank must be located in the 0 to 1G address range. See section 4.3.1.12.



4.3.1.8 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.

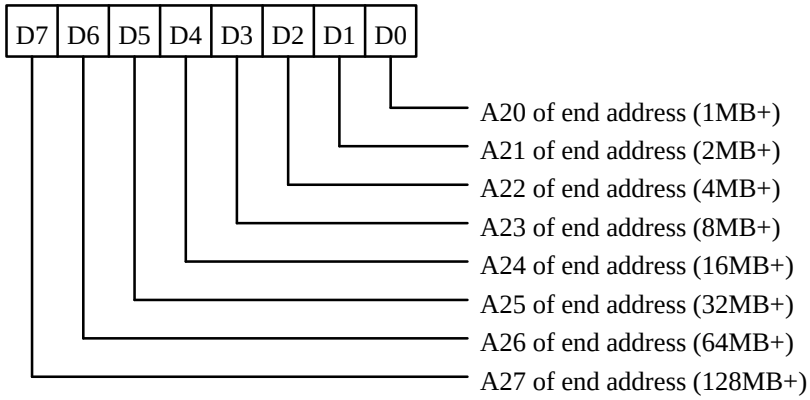


4.3.1.9 Memory Bank Ending Address BCRs

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 high-

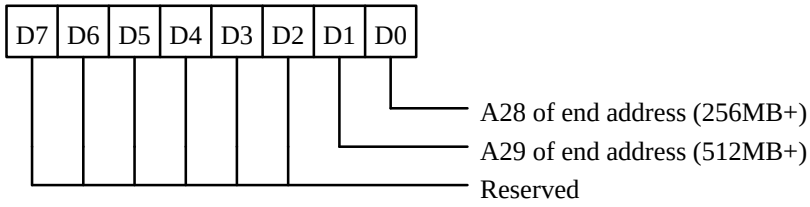
est memory location minus 1, with the least significant 20 bits truncated. Each bank must be located in the 0 to 1G address range. These BCRs must be programmed in conjunction with the ending address and extended ending address registers. See section 4.3.1.12.



4.3.1.10 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.



4.3.1.11 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.

4.3.1.12 Memory Bank Configuration Example

In the example memory bank configuration shown in Figure 4-3, 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.

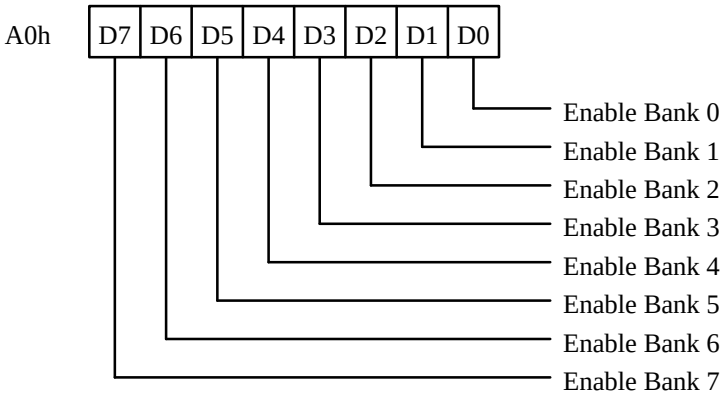


Table 4-11. Example Memory Bank Addressing Mode Configuration

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

4.3.1.13 Memory Bank Enable BCR

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

4.3.1.14 Memory Bank Addressing Mode

As shown in Table 4-11, 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).

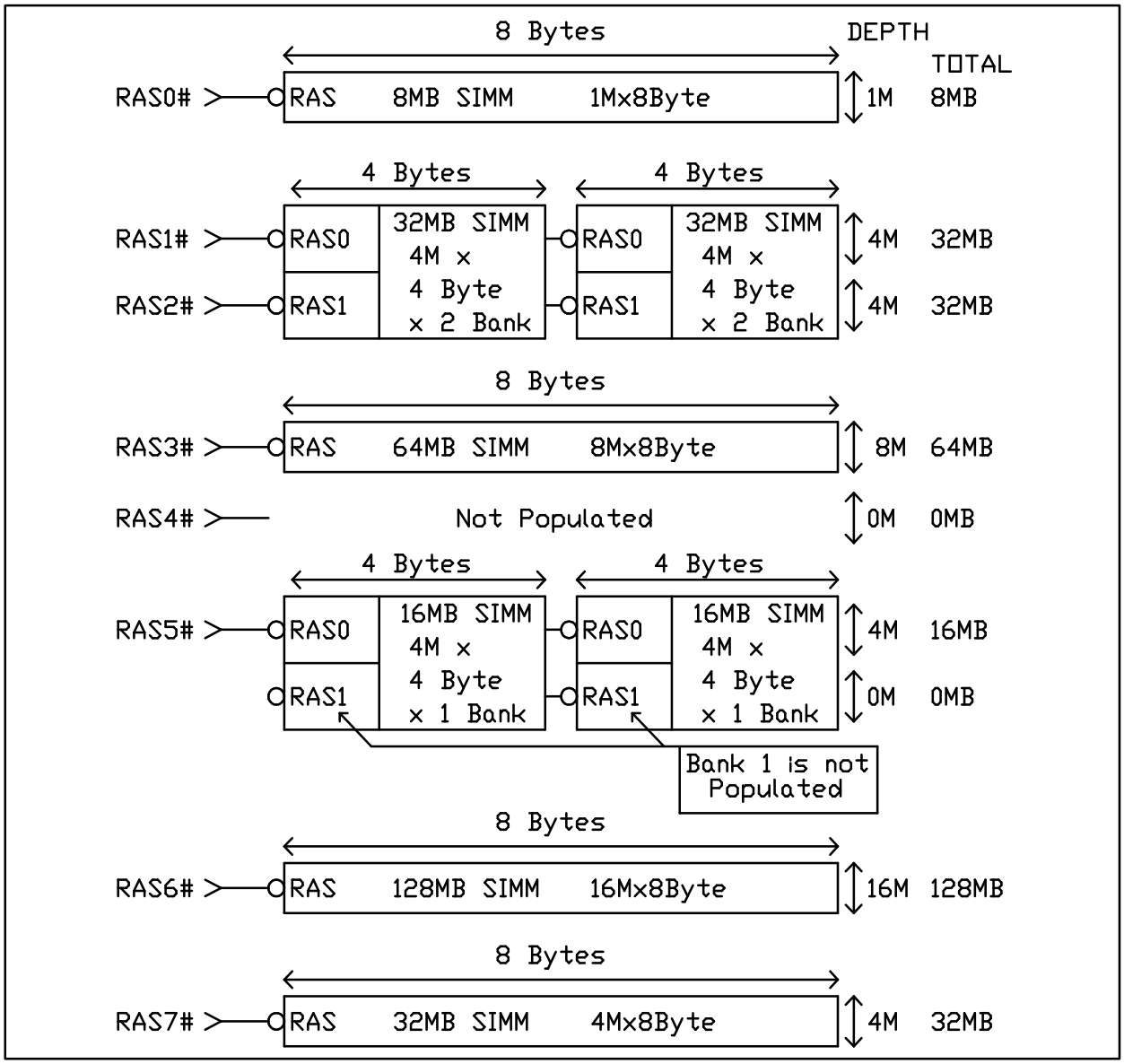


Figure 4-3. Example Memory Bank Configuration

4.3.1.15 Starting and Ending Addresses

As shown in Table 4-12, 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 4-12. 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

4.4 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 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

4.4.1 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#.

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.

4.4.1.1 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 busmaster 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 busmaster is requesting less than eight bytes.

While parity is enabled, the bridge also executes memory writes of less than eight bytes in response to busmaster 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.

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 busmaster 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.

4.4.1.2 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 busmaster 8-byte writes to memory, the bridge writes eight data bytes and one check byte to memory. When the busmaster 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.

4.4.1.3 Memory Reads

Figure 4-4 shows a simplified data flow in a 660 system during CPU to memory read transfers. Figure 4-5 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 4-4 through Figure 4-8 is considerably simplified and does not show the posted write buffers and other internal details.

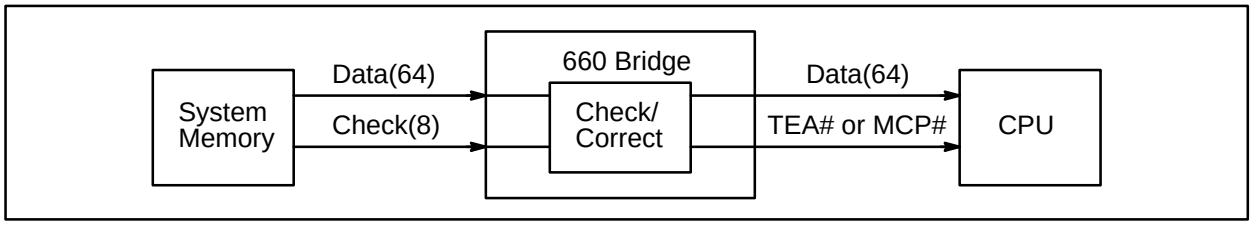


Figure 4-4. CPU Read Data Flow

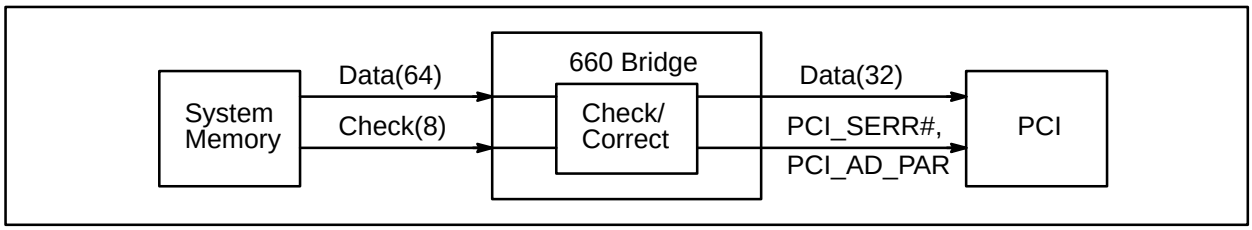


Figure 4-5. PCI Read Data Flow

4.4.1.4 Eight-Byte Writes

Figure 4-6 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.

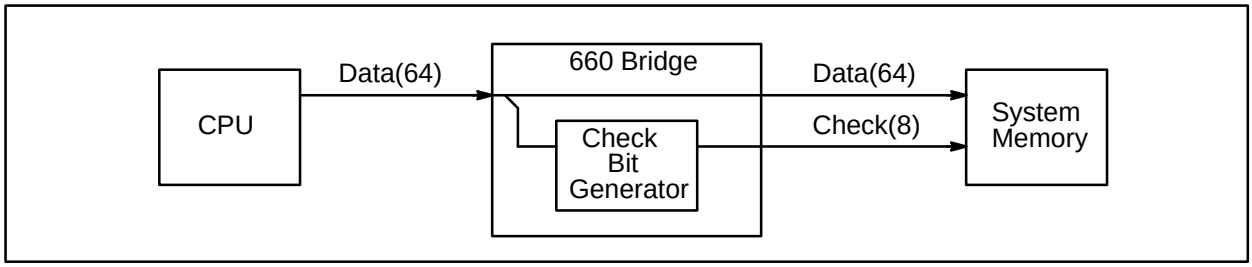


Figure 4-6. CPU 8-Byte Write Data Flow

Figure 4-7 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.

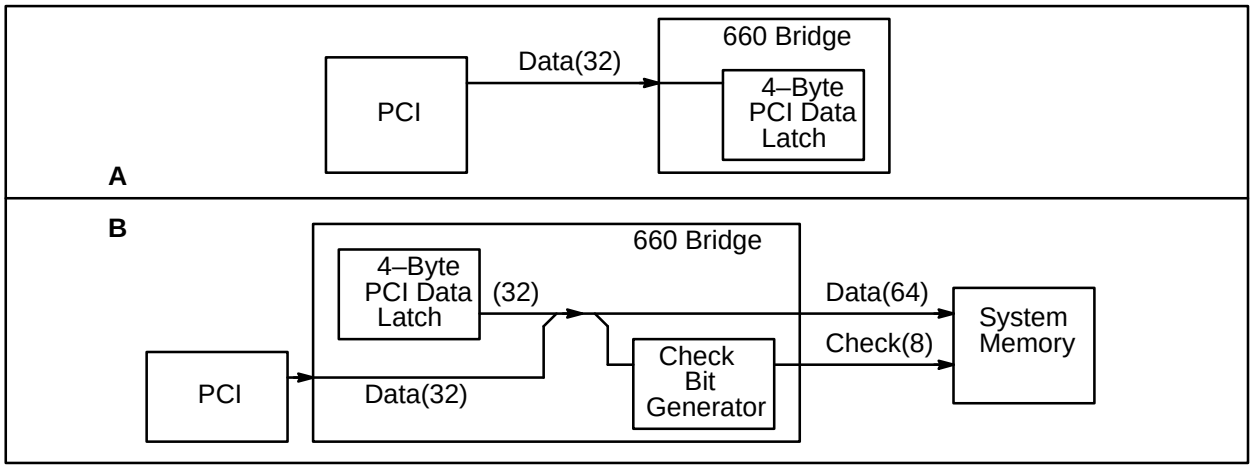


Figure 4-7. PCI 8-Byte Write Data Flow

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.

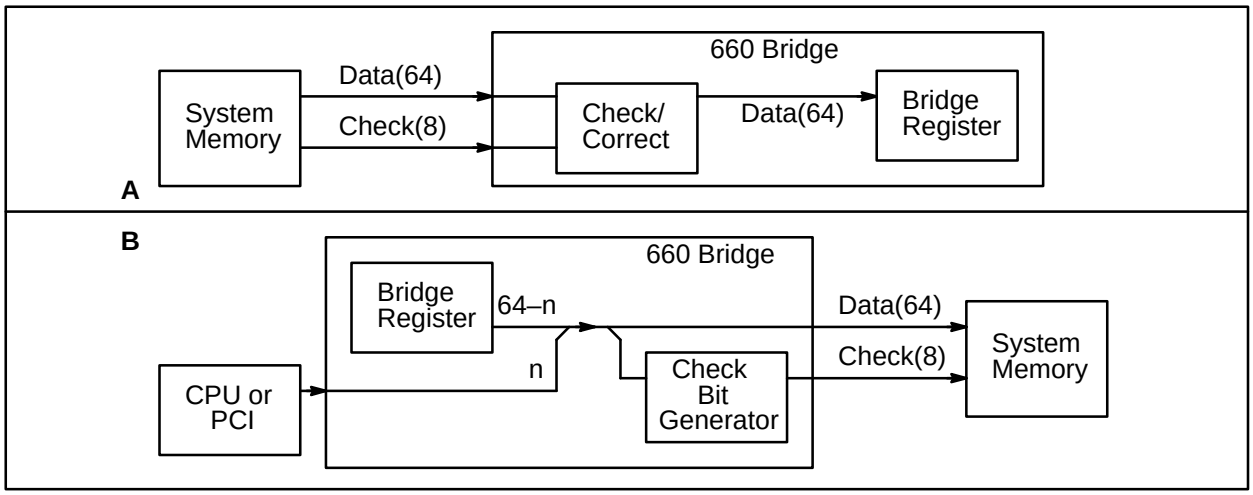


Figure 4-8. PCI or CPU Read-Modify-Write Data Flow

4.4.1.5 Less-Than Eight-Byte Writes

Figure 4-8 shows the simplified data flow during a CPU or PCI busmaster to memory write of less than eight bytes, during which the bridge executes a RMW cycle. In Figure 4-8(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 4-8(B), this data is modified by the bridge, which replaces the appropriate memory data bytes with write data from the busmaster. The bridge then recomputes all of the check bits and writes the new data and check bits to memory.

4.4.1.6 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.

4.4.1.7 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.

4.4.1.8 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.

4.4.1.9 PCI to Memory Read in ECC Mode

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

4.4.1.10 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 4-13. 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 4-13. 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 4-14) 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 4-14. 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 4-15 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 4-15.

Table 4-15. 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 4-16. 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 4-16 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.

4.4.1.11 Check Bit Calculation

The 660 generates the check bits based on Table 4-17 (which is shown using little-endian bit numbering).

Table 4-17. 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

4.4.1.12 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 4-18, 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 4-18. Syndrome Decode

S7	S6	S5	S4	S0 S1 S2 S3	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		cb4			d8			d24			d0			d16			
0	0	1	0		cb5			d9			d25			d1			d17			
0	0	1	1			d40	d41		d42			d44	d43			d45		d46	d47	
0	1	0	0		cb6			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		cb7			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																	

4.5 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 4-9).

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.

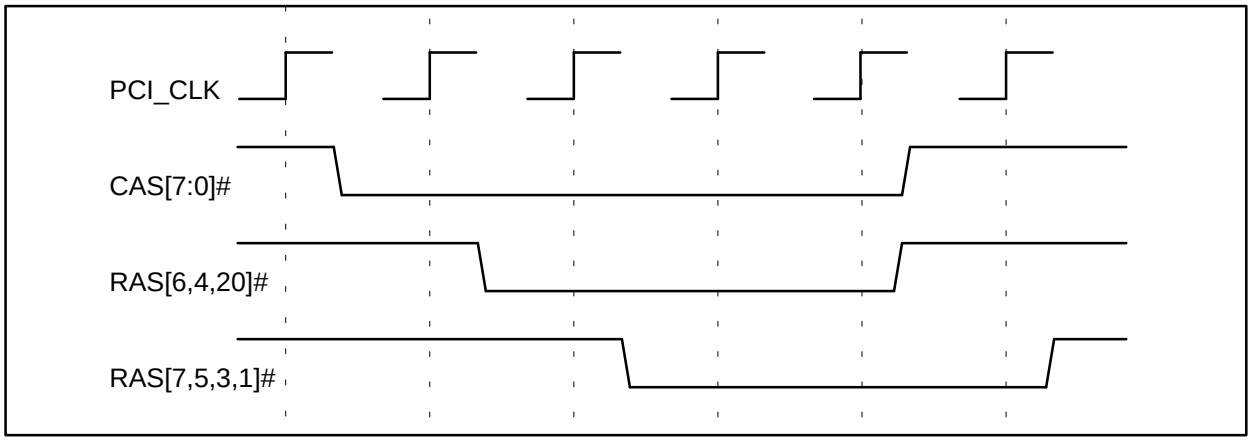


Figure 4-9. DRAM Refresh Timing Diagram

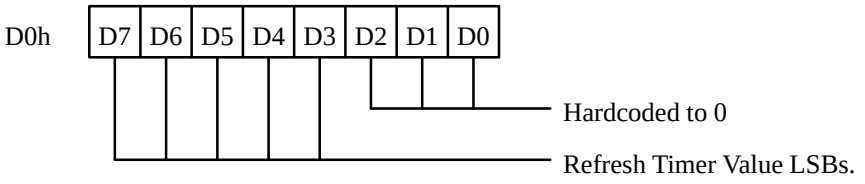
Refresh requests are generated internally by dividing down the PCI_CLK. The divisor value is programmed in the refresh timer divisor register. This register is initialized to 504, a value that provides a refresh rate of 15.1us when PCI_CLK rate is 33MHz. For other PCI_CLK frequencies, the refresh rate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

Refresh continues to occur even if CPU_CLK is stopped.

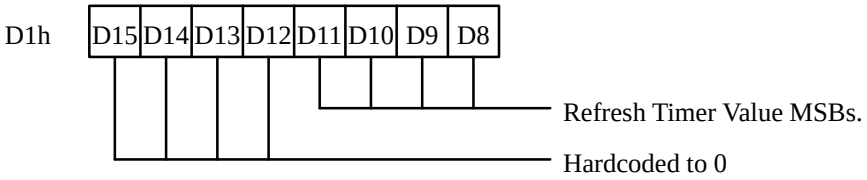
4.5.1 Refresh Timer Divisor Register

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.

4.6 Atomic Memory Transfers

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

4.6.1 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 busmasters. See Section 4.6.2.

PCI busmasters 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 4.6.3.

4.6.2 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.

4.6.3 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 busmasters and other PCI busmasters from reading or writing within a block of memory which is locked by a PCI busmaster. CPU busmaster accesses to a locked block are retried with ARTRY#. PCI busmaster accesses to a locked block are retried with the PCI bus retry protocol. PCI and CPU busmaster 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.

4.6.4 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.

4.7 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 4-19 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).

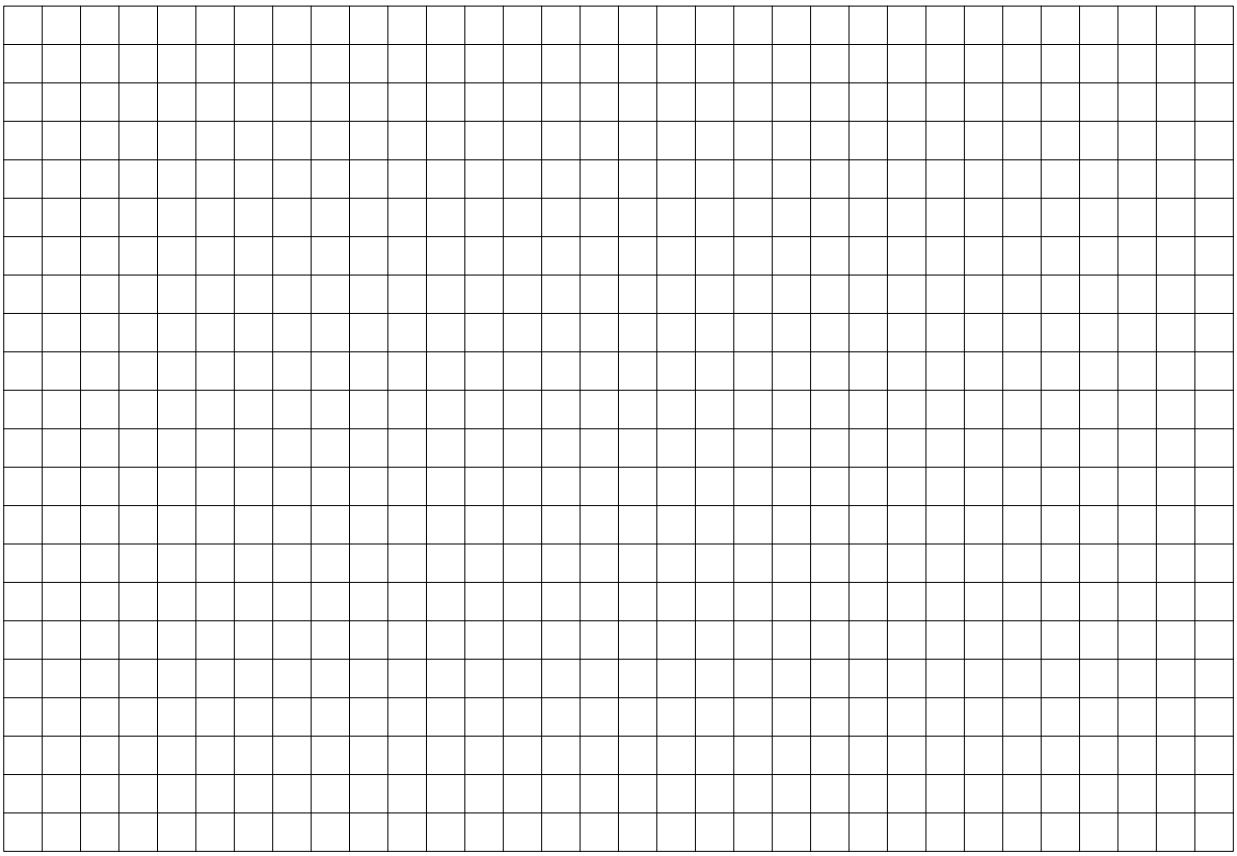
Table 4-19. Typical DRAM Module Maximum Input Capacitance

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

4.8 Related Bridge Control Registers

Information on these registers is contained in the *660 Bridge Manual.*

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



Section 5
PCI Bus

The MCM includes a 32-bit PCI bus interface that runs at frequencies up to 33MHz. The MCM PCI interface is a host PCI bridge that is compliant with the PCI Specification, revisions 2.0 and 2.1 (see Section 5.5) for the 3.3v and 5v signalling environments.

The MCM PCI bus interface is designed to run at one half of the CPU bus frequency.

PCI bus activity initiated by the CPU is discussed in section 3. This section describes PCI bus transactions initiated by a (non-660-bridge) PCI busmaster. PCI busmasters can initiate arbitrary length read and write bursts to system memory.

Whenever a PCI busmaster accesses system memory, the 660 broadcasts a snoop cycle onto the CPU bus. See the 660 User's Manual CPU Bus section for details.

The MCM supports ISA Master operations to system memory.

5.1 PCI Transaction Decoding

When a PCI busmaster initiates a transaction on the PCI bus, the transaction either misses all of the targets and is master aborted, or it is claimed by a PCI target. The target can be either MCM system memory (via the 660) or another PCI target.

5.1.1 PCI Transaction Decoding By Bus Command

Table 5-1 shows the response of the 660 and other agents to various PCI bus transactions initiated by a PCI busmaster other than the 660. 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 5-1. MCM Response to PCI_C[3:0] Bus Commands

C[3:0]	PCI Bus Command	Can a PCI busmaster 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.	None	Yes. The ISA bridge is intended to be the target.
0001	Special Cycle	Yes	None	Yes
0010	I/O Read	Yes	None	Yes
0011	I/O Write	Yes	None	Yes
0100	Reserved	No. Reserved	None	n/a
0101	Reserved	No. Reserved	None	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	None	n/a
1001	Reserved	No. Reserved	None	n/a
1010	Configuration Read	No. Only 660.	None	Yes
1011	Configuration Write	No. Only 660.	None	Yes
1100	Memory Read Multiple	Yes	System memory read	Yes, if no address conflict.
1101	Dual Address Cycle	Yes	None	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.

PCI busmasters are not able to access the boot ROM or the BCRs in the 660.

PCI busmasters can not initiate transactions to the CPU bus. CPU bus devices can, however, initiate transfers to the PCI bus. Large amounts of data are typically moved between the PCI bus and memory by bursts initiated by a PCI busmaster.

See Section 8.1.3 and the PCI specification for information on PCI Interrupt Acknowledge transactions.

5.1.2 PCI Memory Transaction Decoding By Address Range

When a PCI busmaster transaction is decoded by bus command as a system memory read or write, the 660 checks the address range. Table 5-2 shows the address mapping of PCI busmaster memory accesses to system memory. This is the mapping that the 660 uses when it decodes the bus command to indicate a system memory access.

Table 5-2. Mapping of PCI Memory Space, Part 1

PCI Bus Address	Other Conditions	Target Cycle Decoded	Target Address	Notes
0 to 2G	IGN_PCI_AD31 Deasserted	Not Decoded	N/A	No Response.
	IGN_PCI_AD31 Asserted	System Memory *	0 to 2G	Snooped by caches.
2G to 4G		System Memory *	0 to 2G	Snooped by caches.

* Memory does not occupy this entire address space. Accesses to unoccupied space are not decoded.

Unless the IGN_PCI_AD31 signal is asserted, PCI memory accesses in the 0 to 2G address range are ignored by the 660. There is no system memory access, no snoop cycle, and the 660 does not claim the transaction. When the IGN_PCI_AD31 signal is asserted, the 660 maps PCI memory accesses from 0 to 2G directly to system memory at 0 to 2G. PCI memory accesses from 2G to 4G are mapped to system memory from 0 to 2G.

PCI memory accesses that are mapped to system memory cause the 660 to claim the transaction, access system memory, and arbitrate for the CPU bus and broadcast a snoop operation on the CPU bus. A detailed description of the snoop process is presented in the 660 User's Manual.

Table 5-3 gives a more detailed breakdown of the MCM response to PCI memory transactions in the 0 to 2G range. Note that the preferred mapping of PCI memory, so that it can be accessed both by the CPU and by PCI busmasters, is from 16M to 1G–2M.

Table 5-3. Mapping of PCI Memory Space, Part 2

PCI Bus Address	Target Resource	System Memory Address	Snoop Address
2G to 4G	System memory (1)	0 to 2G	0 to 2G
1G–2M to 2G	Reserved (2)	No system memory access. The 660 ignores PCI memory transactions in this range.	No snoop.
16M to 1G–2M	PCI Memory		
0 to 16M	PCI/ISA Memory (3)		

Notes:

1. The 660 maps PCI busmaster memory transactions in the 2G to 4G range to system memory, and the CPU is unable to initiate PCI memory transactions to this address range, so do not map devices to this PCI memory address range.
2. The CPU (thru the 660) can not access the 1G–2M to 2G address range, so do not map PCI devices herein unless the CPU will not access them.
3. Transactions initiated on the PCI bus by the ISA bridge on behalf of an ISA busmaster only (IGN_PCI_AD31 asserted for an SIO), are forwarded to system memory and broadcast snooped to the CPU bus from 0 to 16M. If this is not an ISA busmaster transaction, then the 660 ignores it. Note that the 660 will also forward PCI transactions from 16M to 2G if IGN_PCI_AD31 is asserted during an ISA-bridge-mastered transaction, and that this capability is not normally used.

5.1.3 PCI I/O Transaction Decoding

The 660 initiates PCI I/O transactions on behalf of the CPU. Other PCI busmasters are also allowed to initiate PCI I/O transactions. Table 5-4 shows the MCM mapping of PCI I/O transactions. The 660 ignores PCI I/O transactions.

PCI/ISA I/O is mapped to PCI I/O space from 0 to 64K. The ISA bridge subtractively decodes these transactions (and also PCI memory transactions from 0 to 16M). Other devices may actively decode and claim these transactions without contention.

PCI I/O is assigned from 16M to 1G–8M.

Table 5-4. Mapping of PCI Master I/O Transactions

PCI Bus Address	Target Resource	Other System Activity
1G–8M to 4G	Reserved (1)	The 660 ignores I/O transactions initiated by PCI busmasters.
16M to 1G–8M	PCI I/O devices	
8M to 16M	Reserved (1)	
64K to 8M	Reserved (2)	
0 to 64K	PCI/ISA I/O	

Note:

1. The CPU (thru the 660) can not access this address range, so do not map PCI devices herein unless the CPU will not access them.
2. In contiguous mode, the CPU (thru the 660) can create PCI I/O addresses in the 64K to 8M range. In non–contiguous mode, the CPU can only access PCI addresses from 0 to 64K.

5.1.4 ISA Master Considerations

In systems that implement IGN_PCI_AD31 and use an Intel SIO, memory transactions produced on the PCI bus (by the SIO on behalf of an ISA master) are forwarded to system memory at the corresponding address (0 —16M).

If ISA masters are utilized and the SIO is programmed to forward their cycles to the PCI bus, then no other PCI device (e.g. video) is allowed to be mapped at the same addresses because contention would result.

The SIO contains registers to control which ranges of ISA addresses are forwarded to the PCI bus.

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 SIO and the 660. This will assert IGN_PCI_AD31 (during a PCI transaction) only while either the 660 or the SIO is the initiator (and the 660 knows when it is the initiator).

The required connectivity of IGN_PCI_AD31 prevents the SIO from initiating peer to peer PCI memory transactions in the 0 to 2G range. ISA masters cannot access any PCI memory. The SIO is allowed to initiate PCI memory transactions from 2G to 4G, and other PCI transaction types (I/O &etc.).

5.2 PCI Transaction Details

Further details of the MCM implementation of various PCI transactions, are found in the 660 User's Manual.

5.2.1 Memory Access Range and Limitations

PCI memory reads and writes by PCI busmasters 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 if 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.

5.2.2 Bus Snooping on PCI to Memory Cycles

Each time a PCI (or ISA) busmaster accesses memory, (and once again for each time a PCI burst crosses a cache block boundary) the 660 broadcasts a snoop operation on the CPU bus. If the CPU signals an L1 snoop hit by asserting ARTRY#, the 660 retries the PCI transaction. The ISA bridge then removes the grant from the PCI agent, who (according to PCI protocol) releases the bus for at least one cycle and then arbitrates again. Meanwhile, the 660 grants the CPU bus to the CPU, allowing it to do a snoop push. Then the PCI agent again initiates the original transaction.

During the transaction, the 660 L2 cache is monitoring the memory addresses. The L2 takes no action on L2 misses and read hits. If there is an L2 write hit, the L2 marks that block as invalid, does not update the block in SRAM, and does not affect the PCI transaction. L2 operations have no effect on PCI to memory bursts.

5.2.3 PCI to PCI Peer Transactions

Peer to peer PCI transactions are supported consistent with the memory maps of Table 5-1, Table 5-2, Table 5-3, and Table 5-4, which together show the ranges of different bus command transactions that are supported. If the ISA_MASTER signal is used with the Intel SIO, then the SIO is not allowed to perform peer to peer PCI memory transactions in the 0 to 2G range. No other transaction types are affected.

5.2.4 PCI to System Memory Transactions

Single and burst transfers are supported. Bursts are supported without special software restrictions. That is, bursts can start at any byte address and end on any byte address and can be of arbitrary length.

As per the PCI specification, the byte enables are allowed to change on each data phase. This has no practical effect on reads, but is supported on writes. The memory addresses linearly increment by 4 on each beat of the PCI burst. All PCI devices must use only linear burst incrementing.

In ECC mode, PCI to memory transactions that result in less than 8-byte writes, cause the memory controller in the 660 to execute a read-modify-write operation, during which 8 by-

tes of memory data are read, the appropriate bytes are modified, the ECC byte is modified, and then the resulting 8-byte doubleword is written to memory.

5.2.5 PCI to Memory Burst Transfer Completion

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

- The initiating PCI busmaster disconnects. The 660 handles all master disconnects correctly.
- The 660 target disconnects on a 1 M boundary. The 660 disconnects on all 1M boundaries.
- The 660 target disconnects because the PCI disconnect timer has timed out.
- The CPU retries the snoop cycle that the 660 broadcast on the CPU bus. In this case, the 660 target retries the PCI busmaster. (Note that L2 hits do not affect the PCI to memory transaction. Read hits have no effect on the L2, and write hits cause the L2 to invalidate the block.)
- The 660 will target disconnect the PCI busmaster if the refresh timer 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.

5.2.6 PCI to 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. See Section 3 for more L2 information.

5.2.7 PCI to Memory Writes

During PCI to memory burst writes, the 660 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), 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).

5.2.7.1 Detailed Write Burst Sequence Timing

The detailed write sequence is affected by several factors, such as refresh requests, memory arbitration delays, page and/or bank misses, and cache boundary alignment. Table 5-5 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 5-5 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 5-5 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 5-5. PCI to Memory Write Burst Sequence Timing

								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**.

5.2.8 PCI to Memory Reads

During PCI to memory burst reads, the 660 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), 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).

5.2.8.1 Detailed Read Burst Sequence Timing

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 5-6 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 5-6 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 5-6 are for ECC, parity, or no-error-checking operation.

Table 5-6. 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.

5.2.9 PCI BE# to CAS# Line Mapping

Table 5-7 shows which CAS# lines are activated when a PCI master writes memory. Note that CAS[0]# refers to byte addresses 0 mod 8, CAS[1]# refers to byte addresses 1 mod 8, etc.. For read cycles, eight bytes of memory data are read on each access, but the master receives only the desired 4 bytes. The bytes are read or written to memory independently of BE or LE mode (the endian mode byte swappers are situated between the CPU and the rest of the system, not between the PCI and the rest of the system).

Table 5-7. Active CAS# Lines – PCI to Memory Writes, BE or LE Mode

PCI_ AD[2]	Byte Enables BE[ ]#				Column Address Selects CAS[ ]#							
	3	2	1	0	0	1	2	3	4	5	6	7
0	1	1	1	1								
0	1	1	1	0	X							
0	1	1	0	1		X						
0	1	1	0	0	X	X						
0	1	0	1	1			X					
0	1	0	1	0	X		X					
0	1	0	0	1		X	X					
0	1	0	0	0	X	X	X					
0	0	1	1	1				X				
0	0	1	1	0	X			X				
0	0	1	0	1		X		X				
0	0	1	0	0	X	X		X				
0	0	0	1	1			X	X				
0	0	0	1	0	X		X	X				
0	0	0	0	1		X	X	X				
0	0	0	0	0	X	X	X	X				
1	1	1	1	1								
1	1	1	1	0					X			
1	1	1	0	1						X		
1	1	1	0	0					X	X		
1	1	0	1	1							X	
1	1	0	1	0					X		X	
1	1	0	0	1						X	X	
1	1	0	0	0					X	X	X	
1	0	1	1	1								X
1	0	1	1	0					X			X
1	0	1	0	1						X		X
1	0	1	0	0					X	X		X
1	0	0	1	1							X	X
1	0	0	1	0					X		X	X
1	0	0	0	1						X	X	X
1	0	0	0	0					X	X	X	X

Notes:

X = active. Blank = inactive. Byte enables would normally represent contiguous addresses. This table shows what would happen for all cases.

5.3 Bus Arbitration Logic

The MCM 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.

The 100 MHz PPC 603e MCM example planar uses the Intel SIO as the PCI bus arbiter. The PCI arbiter sees the 660 as one of several PCI agents. The order of priority for PCI arbitration is programmable, and is initially set to be:

1. 660 (the SIO normally parks the bus on the 660)
2. PCI slot 1
3. PCI slot 2
4. PCI slot 3

For more information on arbitration, see Section 9 on planar initialization, and see the PCI Arbitration Controller section of the SIO data book. See the example planar schematics for the connection of the various PCI requests and grants.

There may be concurrency of cycles on the ISA bus (caused by DMA or ISA masters) with PCI or CPU transactions as long as the ISA bus operations are not forwarded to the PCI bus. Forwarding of ISA bus operations must wait for the ISA bridge to grant the PCI bus to its ISA interface.

5.4 PCI Lock

The MCM 660 does not set PCI locks, but does honor them. Also see Section 4.6.

5.4.1 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 busmasters 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 busmaster 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.

5.4.2 CPU Bus Locking

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

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.

5.5 PCI 2.1 Compliance

The MCM contains a highly programmable system component, the 660 bridge. The 660 can be programmed to meet a wide range of application requirements. Along with this flexibility comes the ability to shoot oneself in the foot. This section discusses ways to program the 660 to ensure compliance with the PCI 2.1 specification.

5.5.1 PCI Target Initial Latency

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 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. Program the memory controller for the fastest settings that are consistent with good design practice.

5.5.2 Transaction Ordering Rules

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.

5.5.2.1 The Problem

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.

5.5.2.2 Solution 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.

5.5.2.3 Solution 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.

- 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.
- 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.

5.5.2.4 Solution 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).

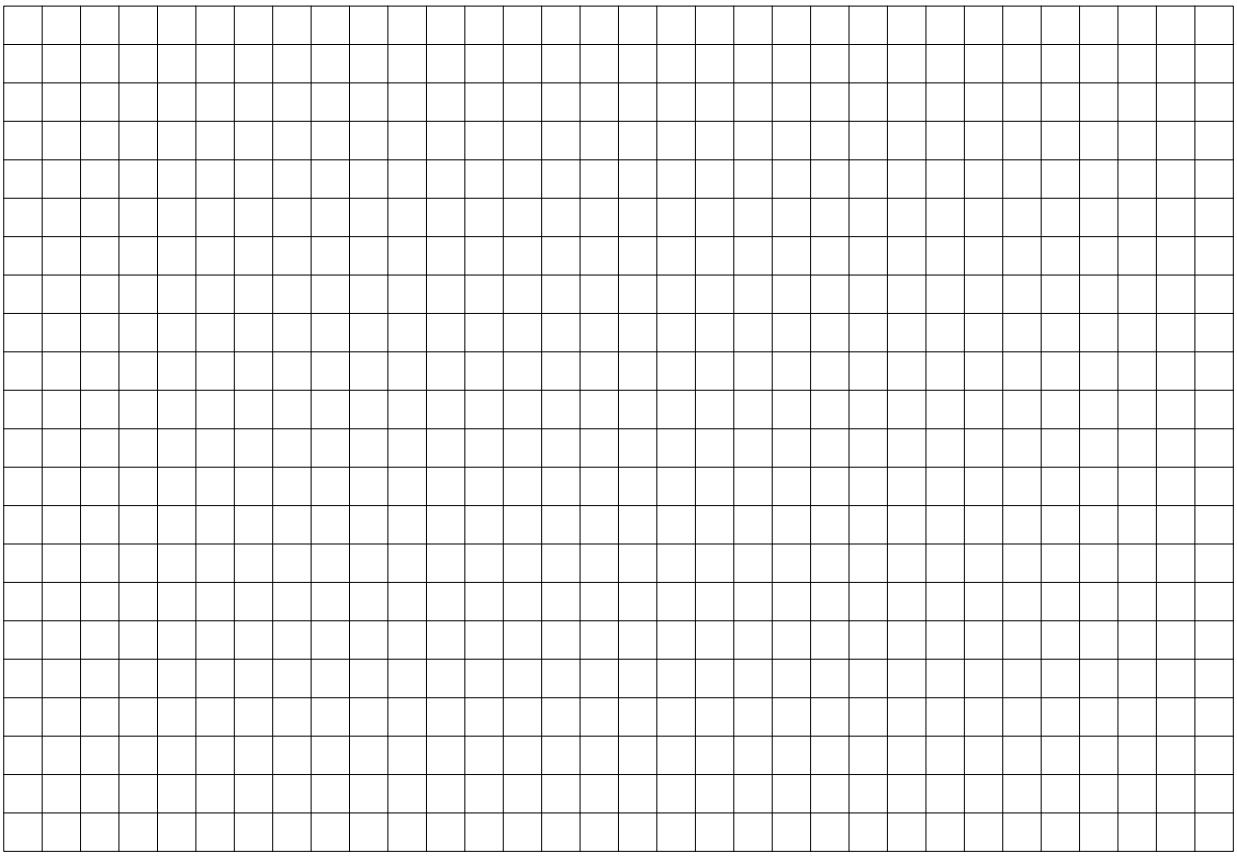
- While the Consumer is receiving the data set as a PCI IO target, the 660 does not post the IO writes, and so each write completes in order, even if some are retried. This technique forces the data set writes to complete before the flag write completes.
- If the Consumer is receiving the data set by 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.

5.6 Related Bridge Control Registers

Information on these registers is contained in the *660 Bridge Manual.*

Bridge Control Register	Index	R/W	Bytes
Memory Controller Misc	8000 0821	R/W	1
PCI/BCR Configuration Address	8000 0CF8	R/W	4
PCI/BCR Configuration Data	8000 0CFC	R/W	4
PCI Type 0 Configuration Addresses IBM27–82650 Compatible	8080 08xx thru 80C0 00xx	R/W	4
PCI Vendor ID	Index 00 – 01	R	2
PCI Device ID	Index 02 – 03	R	2
PCI Command	Index 04 – 05	R/W	2
PCI Device Status	Index 06 – 07	R/W	2
Revision ID	Index 08	R	1
PCI Standard Programming Interface	Index 09	R	1
PCI Subclass Code	Index 0A	R	1
PCI Class Code	Index 0B	R	1
PCI Cache Line Size	Index 0C	R	1
PCI Latency Timer	Index 0D	R	1
PCI Header Type	Index 0E	R	1
PCI Built-in Self-Test (BIST) Control	Index 0F	R	1
PCI Interrupt Line	Index 3C	R	1
PCI Interrupt Pin	Index 3D	R	1
PCI MIN_GNT	Index 3E	R	1
PCI MAX_LAT	Index 3F	R	1
PCI Bus Number	Index 40	R	1
PCI Subordinate Bus Number	Index 41	R	1
PCI Disconnect Counter	Index 42	R/W	1
PCI Special Cycle Address BCR	Index 44 –45	R	2
Error Enable 1	Index C0	R/W	1
Error Status 1	Index C1	R/W	1
Error Enable 2	Index C4	R/W	1
Error Status 2	Index C5	R/W	1
PCI Bus Error Status	Index C7	R/W	1
CPU/PCI Error Address	Index C8 – CB	R/W	4



Section 6
ROM

The MCM implements a 2M ROM space from 4G–2M to 4G (on the CPU bus) by means of the 660 bridge. 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.

The ROM mode is indicated to the 660 on the strapping pin configuration bits during power-on-reset (POR). See section 8.3.2.2.

The ROM access method used by the example planar is referred to as the direct-attach ROM mode; the ROM attaches directly to the 660 (MCM) using the PCI_AD lines. This mode is required when using the Intel™ SIO ISA bridge (as the example planar does) 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 example planar uses an AMD AM29F040-120 Flash™ ROM to contain the POST and boot code. This is a 512K device located at 4G–2M. It is recommended that Vital Product Data (VPD) such as the motherboard speed and native I/O complement be programmed into in this device. It is possible to program the Flash before or during the manufacturing process.

The other ROM access method (remote ROM mode – see section 6.2) attaches the ROM device to an external PCI agent which supports the PowerPC Reference Platform ROM space map and access protocol. CPU busmaster 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.

At power-on, the 603/604 CPU comes up in BE Mode with the L1 cache disabled, and begins fetching instructions (using 8-byte single beat reads) at address FFF0 0100 (4G – 1M + 100h). The example planar logic also resets to BE mode.

6.1 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 busmaster 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 6-1, 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.

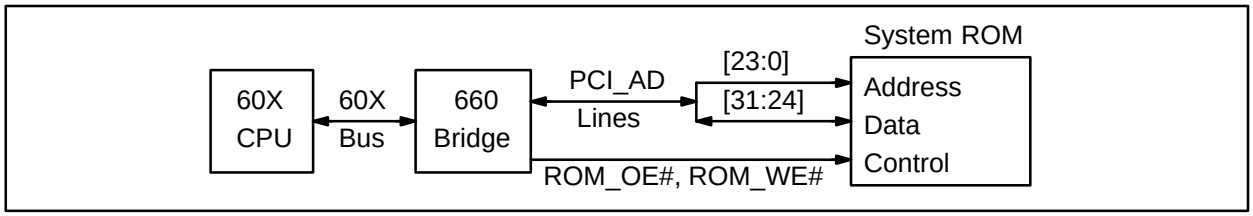


Figure 6-1. ROM Connections

6.1.1 ROM Reads

When a CPU busmaster 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.

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 6.1.1.3.

6.1.1.1 ROM Read Sequence

Figure 6-2 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.

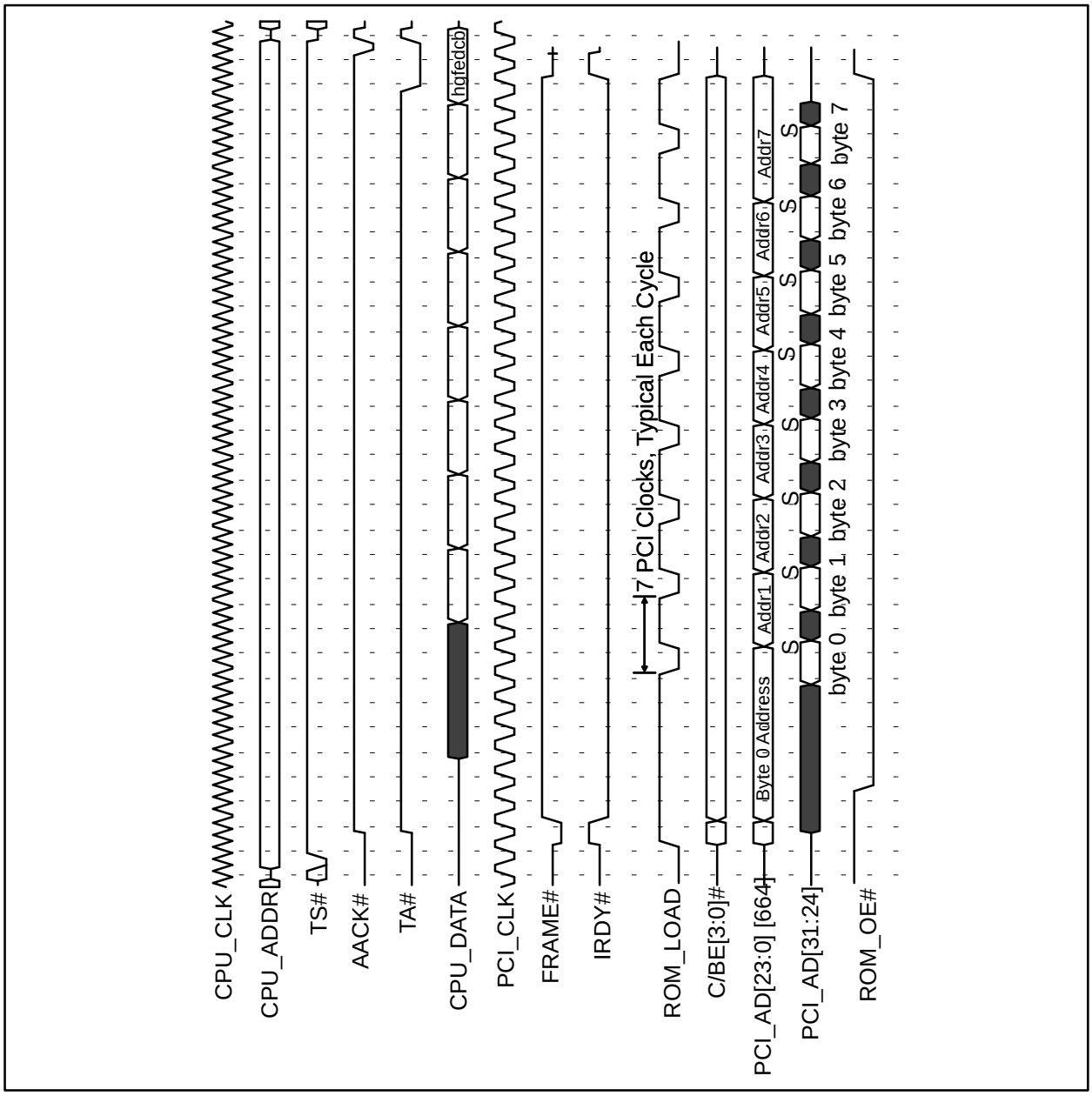


Figure 6-2. ROM Read Timing Diagram

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 6-1 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 6-1. 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

6.1.1.2 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 6-1 could have been caused by a CPU memory read to FF80 0100h, FF80 0101h, or FF80 0105h.

6.1.1.3 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 6-1.

6.1.1.4 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.

6.1.2 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 6.3.1.

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 6.1.2.3. In direct-attach mode, the ROM is attached to the 660 as shown in Figure 6-3.

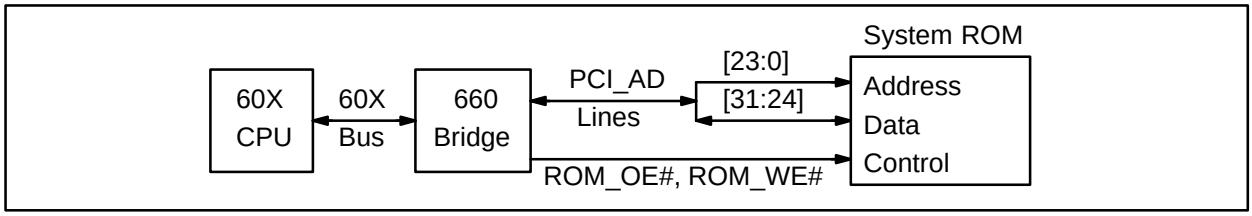


Figure 6-3. ROM Connections

6.1.2.1 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_DEVSEL#, PCI_TRDY#, PCI_STOP#, and ROM_OE# are negated throughout the transaction.

6.1.2.2 Write Protection

Write protection for direct-attach ROM is provided through the ROM lockout BCR (see Section 6.3.2). ROM write-lockout operations are compatible with the 650 bridge.

When a CPU busmaster 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.

6.1.2.3 Data Flow In Little-Endian Mode

Figure 6-4 and Table 6-2 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).

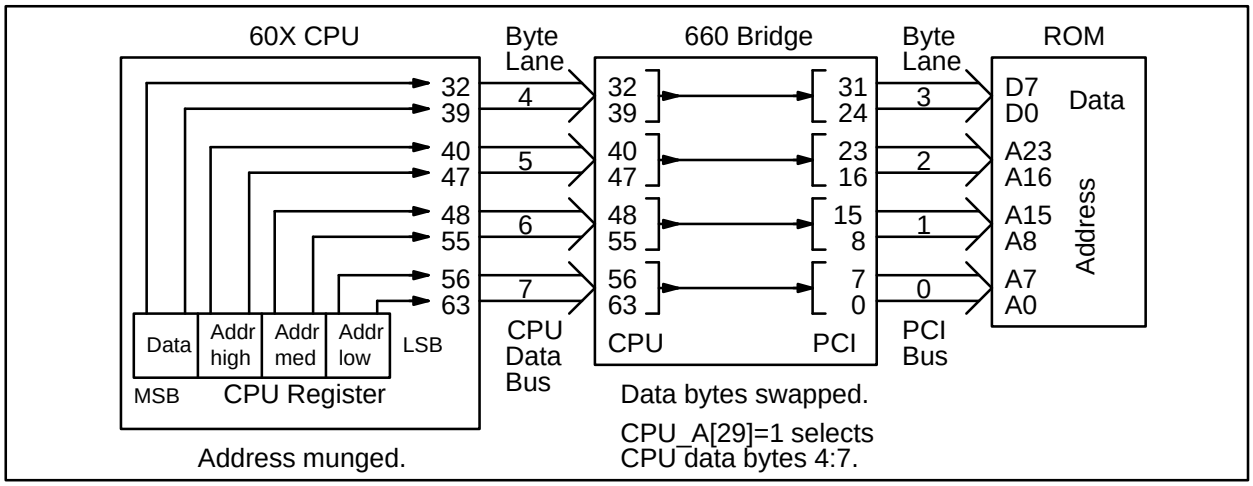


Figure 6-4. ROM Data and Address Flow In Little Endian Mode

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 6-2. 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]

6.1.2.4 Data Flow In Big-Endian Mode

Figure 6-5 and Table 6-3 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).

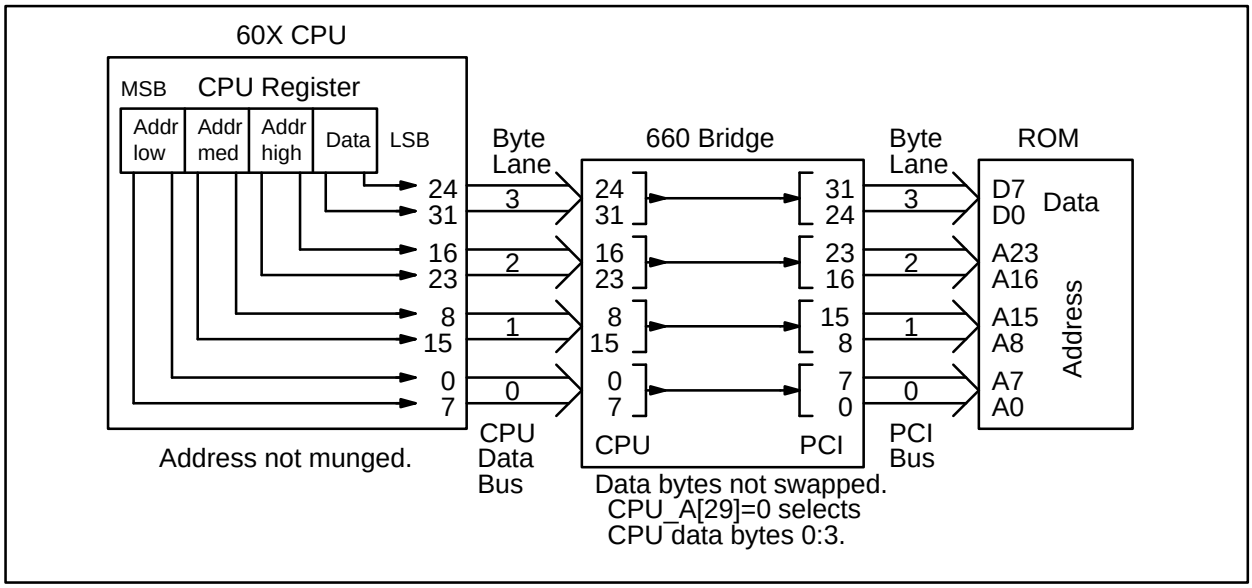


Figure 6-5. ROM Data and Address Flow In Big Endian Mode

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 6-3. 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]

6.2 Remote ROM Mode

In a system that uses the remote ROM mode, the ROM device attaches to a PCI agent. When a CPU busmaster 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 6-6, 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.

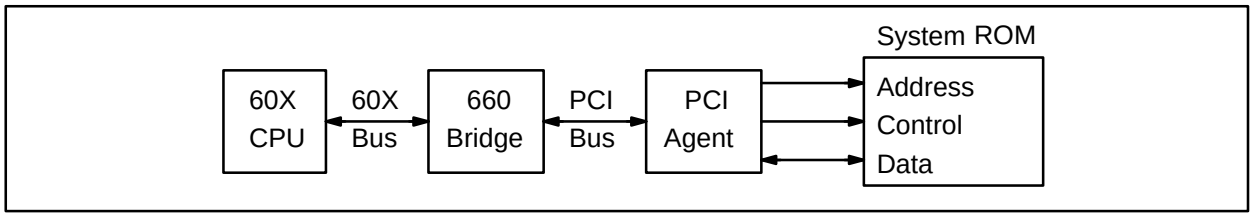


Figure 6-6. Remote ROM Connections

6.2.1 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 6-7 shows the beginning of the operation, including the first two PCI transactions. Figure 6-8 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.

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

6.2.1.1 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 6-4 for a CPU read from FFE0 0000h to FFFF FFFF. Note that the last column in Table 6-4 shows the effect of little-endian mode operation. See Section 6.2.1.4.

The address of the first transaction is the low-order byte of the double-word pointed to by the CPU address (see Section 6.2.1.2). 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 6-4. Note that the byte enables are incrementing within each 4-byte word pointed to by the PCI address.

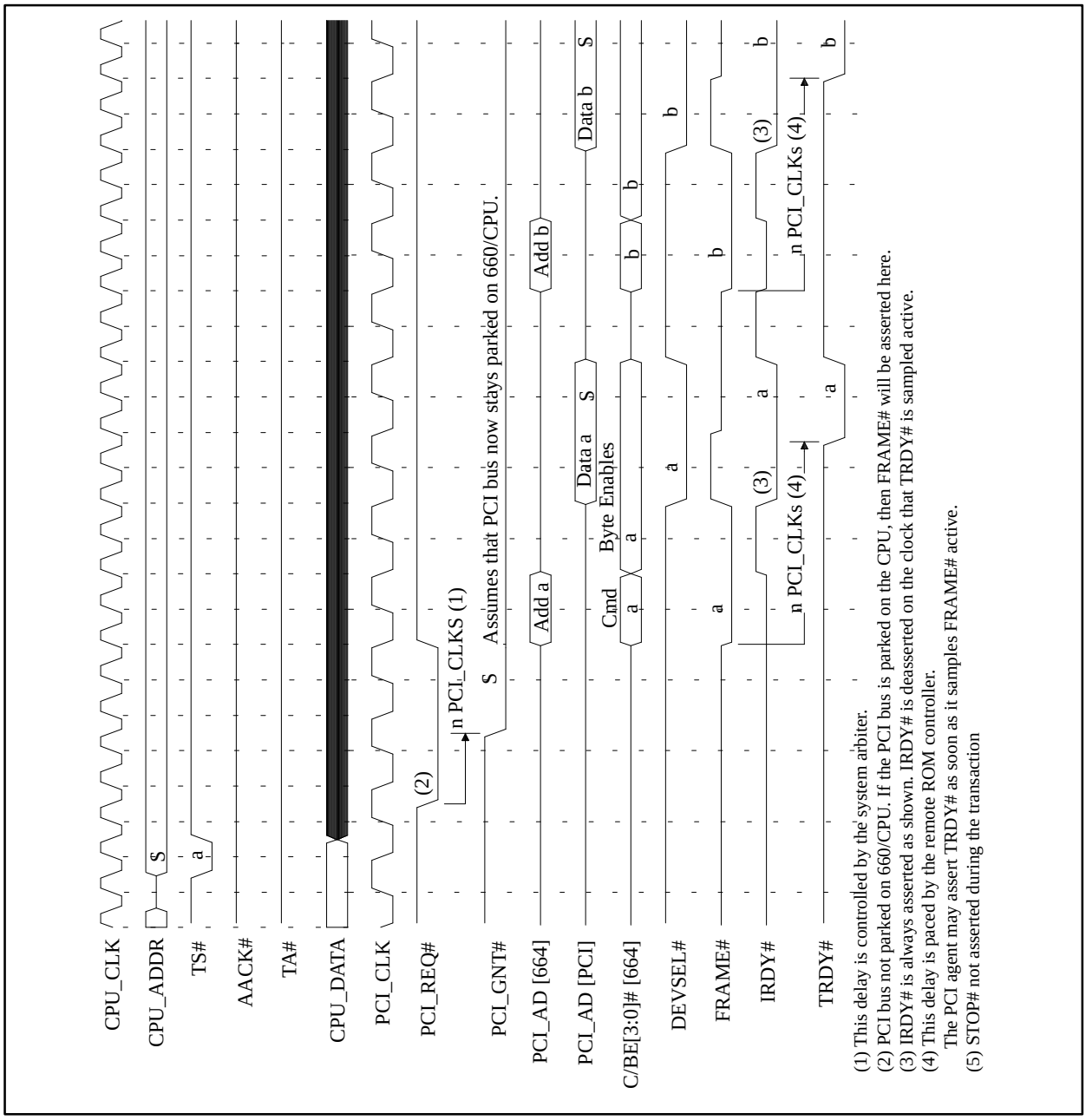


Figure 6-7. Remote ROM Read – Initial Transactions

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.

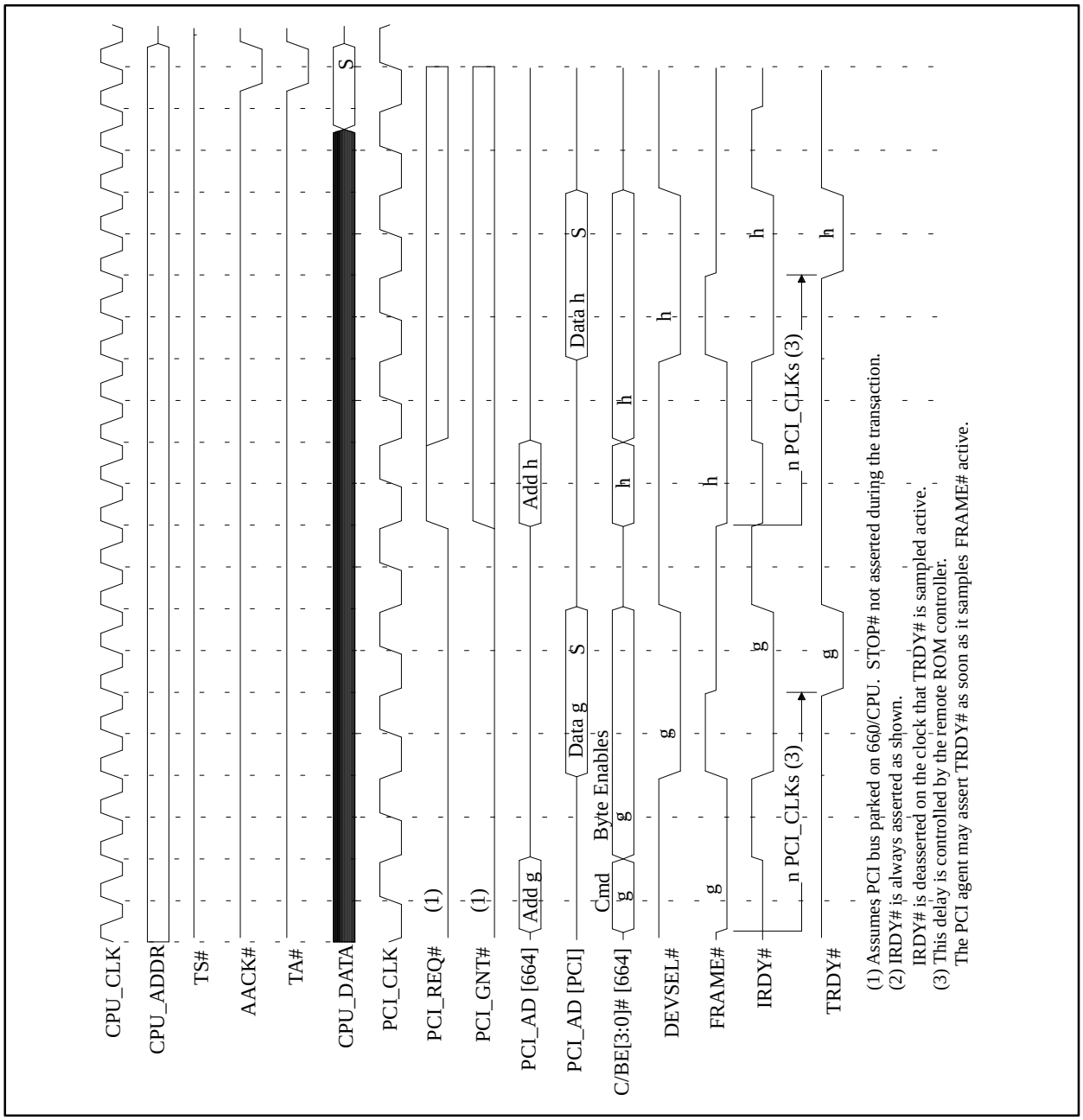


Figure 6-8. Remote ROM Read – Final Transactions

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 6-4. 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

6.2.1.2 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 6-4 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.

6.2.1.3 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.

6.2.1.4 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 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 6-4.

6.2.1.5 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.

6.2.2 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.

6.2.2.1 Write Sequence

A CPU busmaster 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 (FF80 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.

6.2.2.2 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).

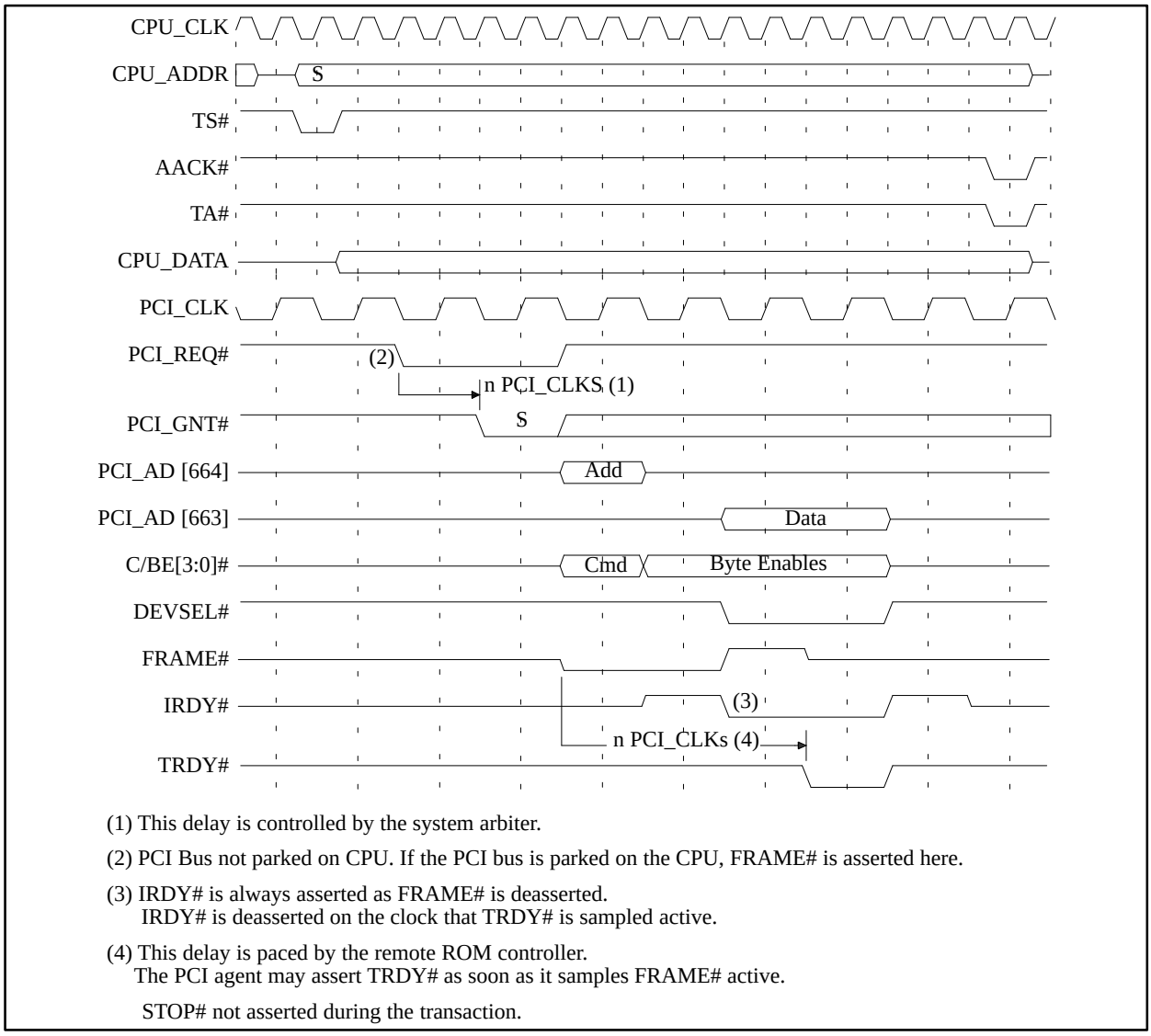
6.2.2.3 Address, Size, Alignment, and Endian Mode

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 busmaster 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. In contrast, in direct-attach ROM mode, the 660 forwards PCI busmaster 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 6-9. Remote ROM Write

6.3 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.

6.3.1 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 6.1.2). This register must be written by means of a 4-byte transfer. Bits are shown with little-endian labels.

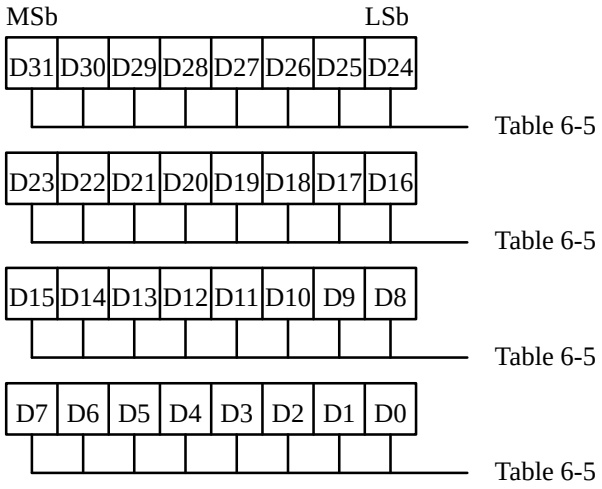


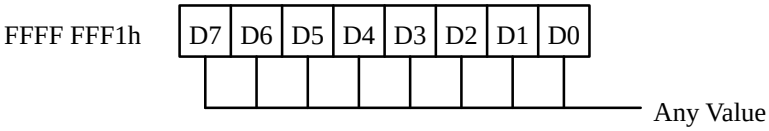
Table 6-5. ROM Write BCR Contents

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

6.3.2 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.

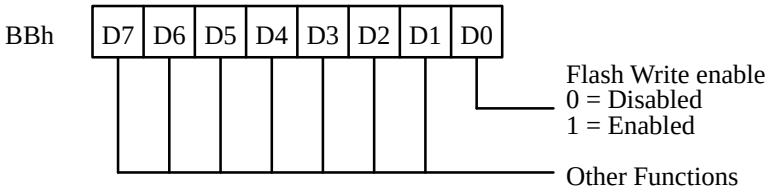


Bits 7:0 Writing any value to the register prevents all future writes to a ROM that is connected directly to the 660 through the PCI_AD lines.

6.3.3 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

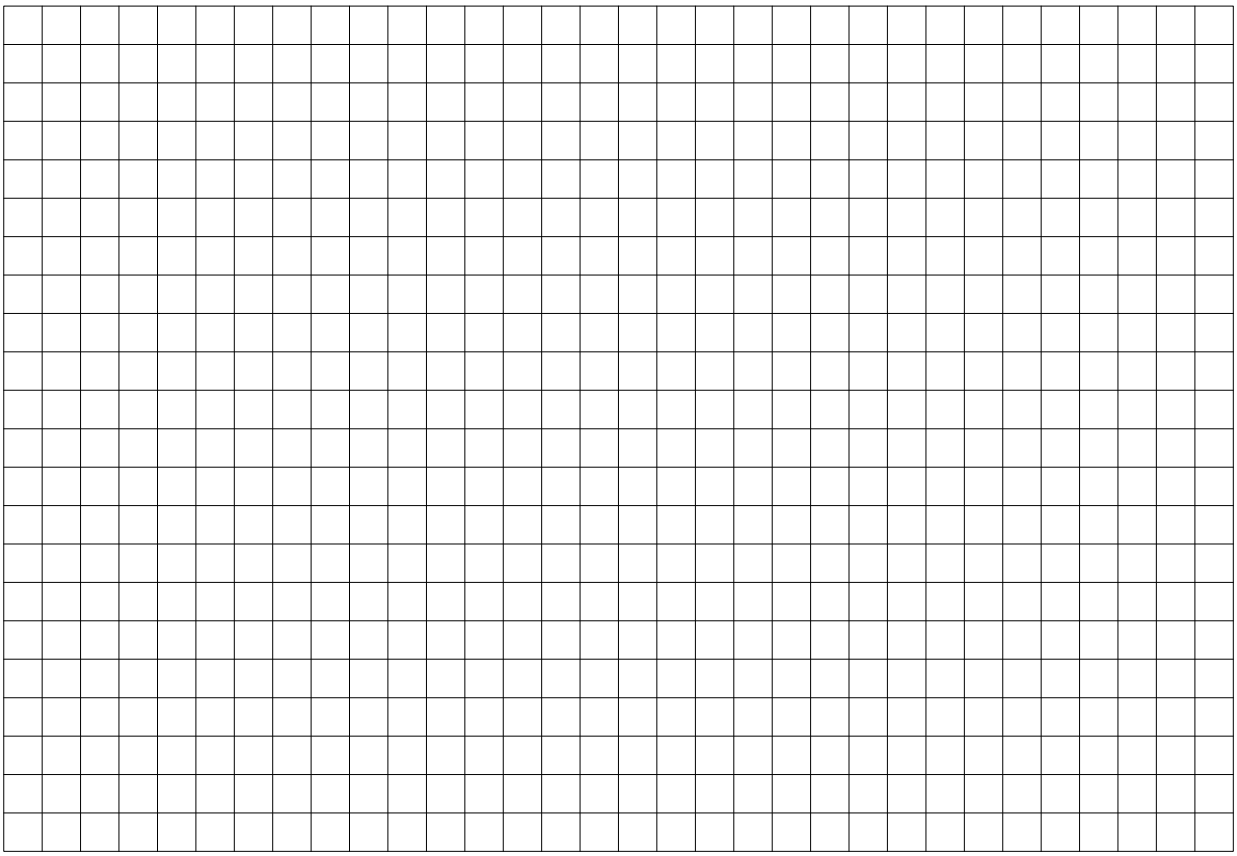


Bit 0 Flash write enable: When the ROM is remotely attached, this bit controls write access to the flash ROM address space (4G – 2M to 4G). When enabled, writes to this space are forwarded to the PCI memory space at the same address. When disabled, writes to this space are treated as no-ops and an error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

6.3.4 Other Related BCRs

Information on these registers is contained in the *660 Bridge Manual.*

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



Section 7
Clocks

The 100 MHz PPC 603e MCM provides a separate clock signal for each CPU bus and PCI bus agent in the system. The clock signals are generated by a Motorola™ MPC970. The MPC970 inputs are brought off of the MCM to allow maximum programming flexibility. See the data sheet for more information on MPC970 programming, capabilities, and characteristics. See Figure 7-1.

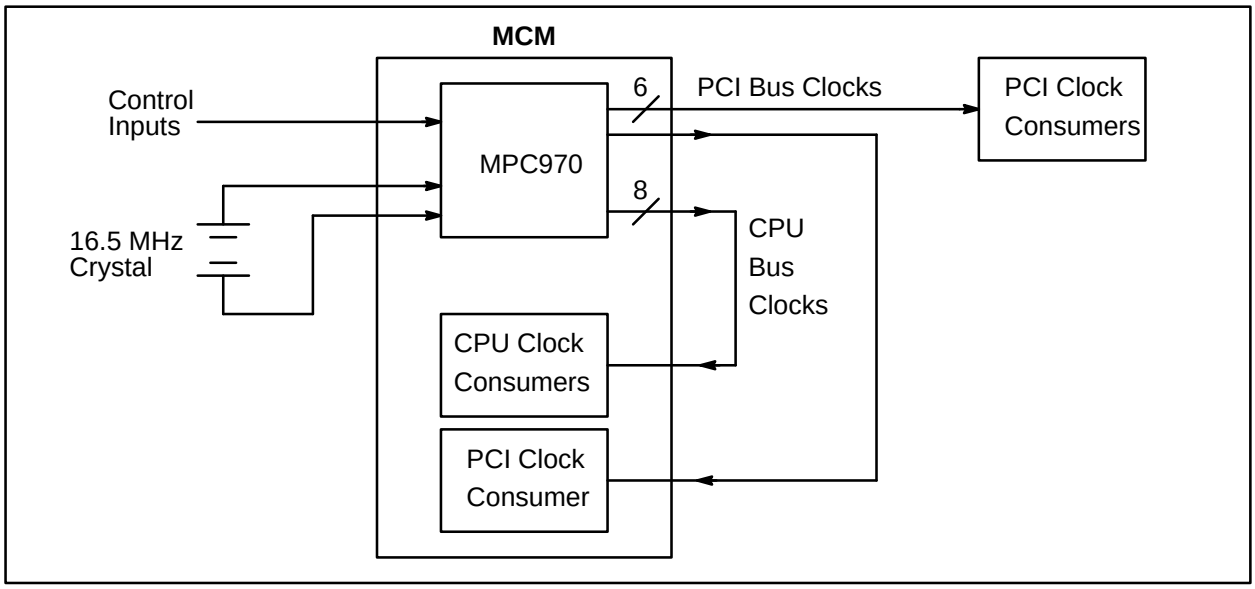


Figure 7-1. MCM Clocks

The example planar configures the MPC970 to produce 8 CPU bus clocks, all of which are consumed by the MCM components. When required, additional CPU bus clocks can be generated using a 'zero delay' clock repeater such as the Motorola MPC930 PLL, using one of the supplied CPU clocks as the seed clock.

The planar also configures the MPC970 to produce seven PCI bus clocks, two of which are consumed by the MCM components. The other PCI clocks are available for use on the planar. The MPC 970 is configured to interface with a 16.5 MHz crystal. An oscillator internal to the component then produces the root frequency used to create all clock outputs.

7.1 CPU Clock Physical Design Rules

The MCM and the example planar were physically designed with careful attention to the fact that, at PowerPC operating frequencies, the circuit board itself becomes a component that materially affects circuit behavior. Clock nets are the most critical wiring on the board. Their wiring requirements should be given priority over the requirements of other groups of signals. The following design rules are helpful in designing low-noise and low-skew clock nets:

1. Clock nets are to have a minimum number of vias.
2. No clock wires may be routed closer than one inch to the edge of the board. Consider adding EMC caps to the far end of the clock trace.
3. Clock nets should not have more than two nodes. Daisy chains, stubs, and star fanouts are not allowed.
4. Clock nets are to be routed as much as possible on internal signal planes.
5. Route a ground trace as a shield in the adjacent wiring channel on both sides of the clock trace. It is a good practice to periodically (every inch or so) connect these shield traces to the ground plane. Completely surround the clock trace with shield traces. Avoid impedance bumps.
6. Series (source) termination resistors are required. Choose them according to the MPC970 data sheet recommendations and place them as close as possible to the clock source pin of the MCM.
7. To minimize clock skew on the planar, design the circuit board such that the combined length (MCM plus planar) of each of the clocks is the same. In other words, determine the required total length of the longest clock trace, and then make all of the other traces the same total length (see Figure 7-2).

   Inside the MCM, the PCI clock traces run about 1 inch, and the CPU clock traces run about 3.5 inches. Thus the planar runs of the clocks should be adjusted accordingly.

   Suppose in Figure 7-2, one of the PCI clocks planar runs is initially the longest, at 8.5 inches. Added to the about 1 inch of MCM run, the total length is about 9.5 inches. Correct design will then stretch all of the other PCI clock lines to 8.5 inches. The required planar run length of the CPU lines is then 9.5 – 3.5 = 6 inches. See Table 7-1 and Table 7-2.

Table 7-1. Clock Net Lengths

Net Length	Type	Name	Net Topology	
13.7500	CLK	XTAL1	J1.U30	U4.12
11.7071	CLK	XTAL2	J1.T29	U4.13
32.5000	CLK	FRZ_CLK	J1.B11	U4.3
18.6036	CLK	FRZ_DATA	J1.A22	U4.5
8.5866	CLK	PCLK_60X	J1.E18	U4.34
66.3536	CLK	TAG_BCLK	J1.N30	U4.36
29.9142	CLK	CLK_EXT_FB	J1.A17	U4.14
27.0000	CLK	CLK_FB_SEL	J1.A15	U4.9
27.5000	CLK	CLK_PLL_EN	J1.D33	U4.7
67.5000	CLK	SRAM_BCLK0	J1.W30	U4.44

Table 7-1. Clock Net Lengths (Continued)

Net Length	Type	Name	Net Topology	
66.5000	CLK	SRAM_BCLK1	J1.AE30	U4.46
9.1624	CLK	SRAM_BCLK2	J1.AG04	U4.48
8.6624	CLK	SRAM_BCLK3	J1.AA04	U4.50
22.6036	CLK	CLK_COM_FRZ	J1.A18	U4.6
29.6036	CLK	CLK_REF_SEL	J1.B13	U4.8
24.4571	CLK	CLK_TTL_CLK	J1.A26	U4.11
28.0000	CLK	CLK_VCO_SEL	J1.A13	U4.52
1.7500	CLK	663_CPU_CLK	J1.G04	U4.38
63.6036	CLK	664_CPU_CLK	J1.G30	U4.42
24.1036	CLK	CLK_BCLK_DIV0	J1.C16	U4.31
22.2500	CLK	CLK_BCLK_DIV1	J1.B17	U4.27
26.3536	CLK	CLK_FRZ_ STROBE	J1.F33	U4.4
14.8536	CLK	CLK_MPC601_ CLKS	J1.K27	U4.40
25.3536	CLK	CLK_ MR/TRISTATE	J1.A16	U4.2
61.8536	CLK	664_PCI_CLK	J1.J30	U4.18
20.6036	CLK	CLK_PCI_DIV0	J1.B21	U4.20
20.0607	CLK	CLK_PCI_DIV1	J1.C20	U4.26
38.1036	CLK	USER_PCICLK1	J1.AE01	U4.16
29.5000	CLK	USER_PCICLK2	J1.N01	U4.21
42.8536	CLK	USER_PCICLK3	J1.C01	U4.23
31.1036	CLK	USER_PCICLK4	J1.A11	U4.25
25.4571	CLK	USER_PCICLK5	J1.A14	U4.29
29.7500	CLK	USER_PCICLK6	J1.E33	U4.32

Table 7-2. Clock Net Calculations

Net	MCM Run	Total Run	Required Planar Run	Tolerance (Inch)
CPU Clocks	3.5 (nom)	x	x – 3.5	1
664_PCI_CLK	1 (nom)	x	x – 1	1
PCI_CLK[4:1]	1 (nom)	x	x – 1	1

In our experience, a maximum clock trace length skew of two inches is acceptable. It is the responsibility of the designer to determine the appropriate amount of allowed clock trace length variation for the individual application.

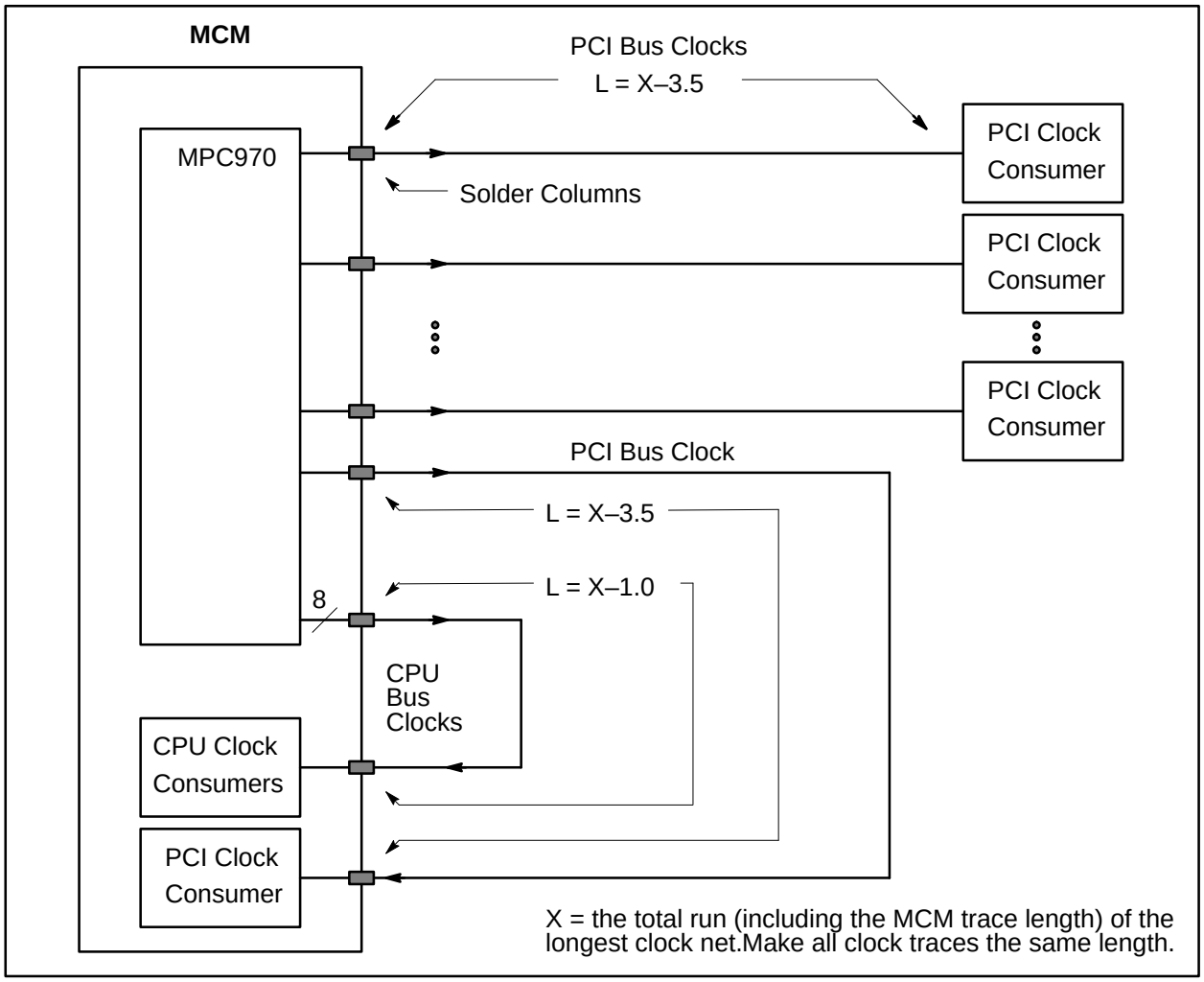


Figure 7-2. MCM Clocks

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 7-3.

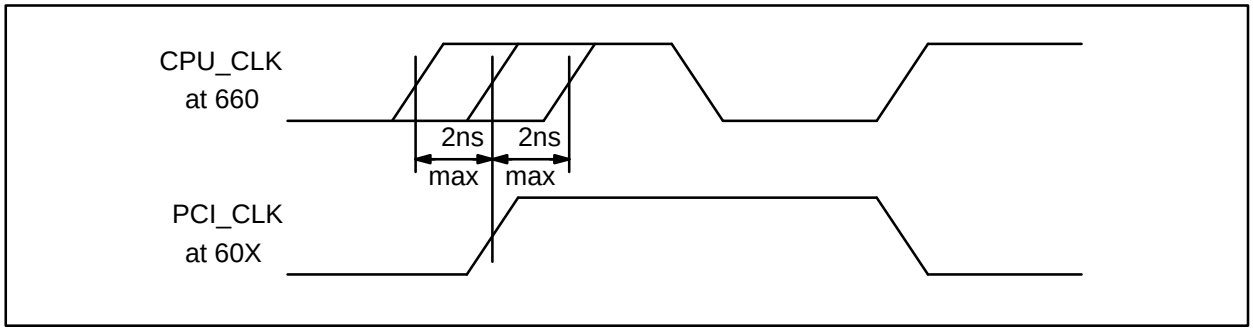


Figure 7-3. CPU_CLK to PCI_CLK Skew

If the design rules laid out in Section 7.1 are followed, the requirements are met.

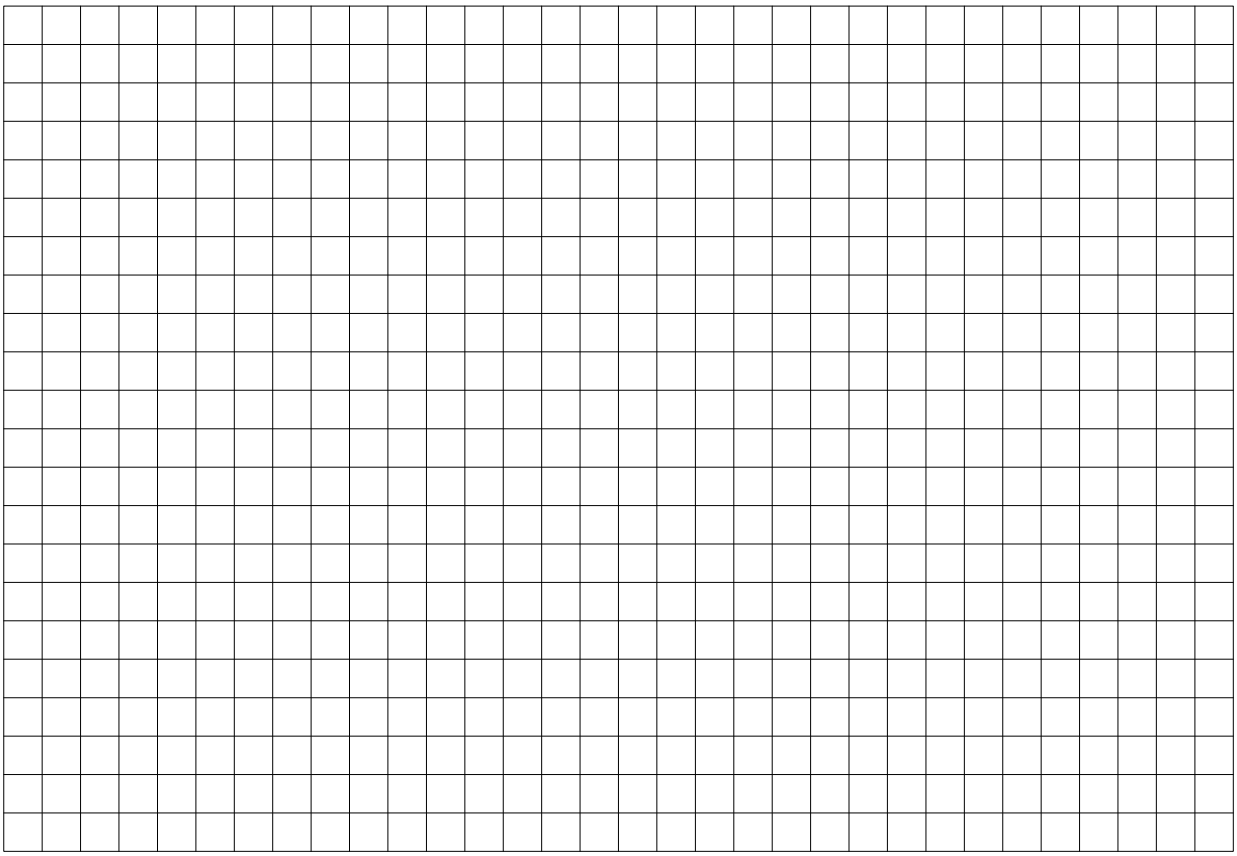
7.2 Clock Freezing

Some of the PCI clocks may not be required. In this case, the ability of the MPC970 to stop (freeze) the unused clocks is useful to reduce run-time power consumption and EMI emissions.

The MPC970 can freeze any set of output clocks in the low state, while allowing the other clocks to continue running. The freeze command is given to the MPC970 via a two wire synchronous serial interface that originates in the system EPLD. See the clock freeze register description in the System EPLD section for more information on activating this feature.

Firmware can read the presence detect bits of the PCI slots to determine which devices are present. For PCI slots which are not populated, firmware can disable the clocks for those slots.

This function can also be used by power management functions to further reduce power consumption of the motherboard by freezing the clocks of devices which have been placed in a power managed mode.



Section 8
Exceptions

This section contains information on interrupts, errors, resets, and test modes. Some of the functionality of each area is implemented on the MCM, and some of that functionality is located on the planar. To simplify the material, both the MCM and the planar parts of each system are presented in this section.

8.1 Interrupts

8.1.1 Planar Interrupt Handler

Example planar interrupts are handled primarily by the interrupt controller in the ISA bridge, the 660, the CPU, and the firmware.

There are two 8259 type interrupt controllers located in the ISA bridge. These controllers receive and prioritize example planar interrupts, which can be asserted by motherboard logic, PCI devices, or ISA devices. The interrupt controller then asserts an interrupt to the 660.

The interrupt controller is programmed to handle both ISA and PCI interrupts using the correct protocols, under software control. Much of the operation of the interrupt controller is programmable. See the SIO data book for more information.

8.1.2 MCM (660) INT_REQ and INT_CPU#

The 660 features two interrupt inputs, INT_REQ and NMI_REQ.

As shown in Figure 8-1, the 660 inverts INT_REQ and passes it thru to the CPU as INT_CPU#. While the 660 is in 601 error reporting mode, it also uses INT_CPU# to report certain error conditions, but this function is not used by the MCM.

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 to be 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. This could have been done on the example planar; the current connectivity illustrates the use of the 660 pins.

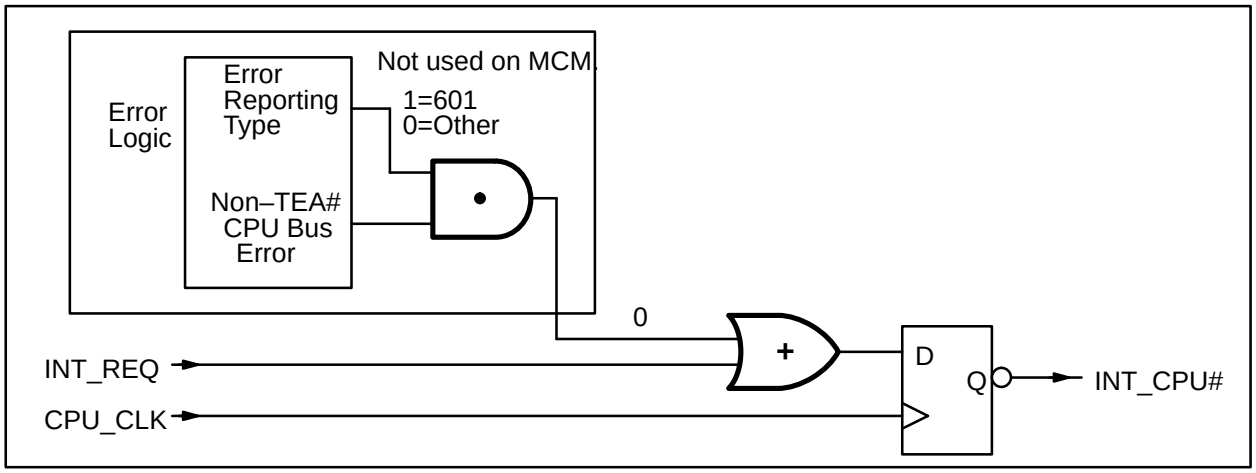


Figure 8-1. Conceptual Block Diagram of INT Logic

8.1.3 Interrupt Acknowledge Transactions

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 busmasters can initiate interrupt acknowledge transactions.

8.1.4 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 8.2.7.1 for more information on NMI_REQ.

8.1.5 Interrupt Handling

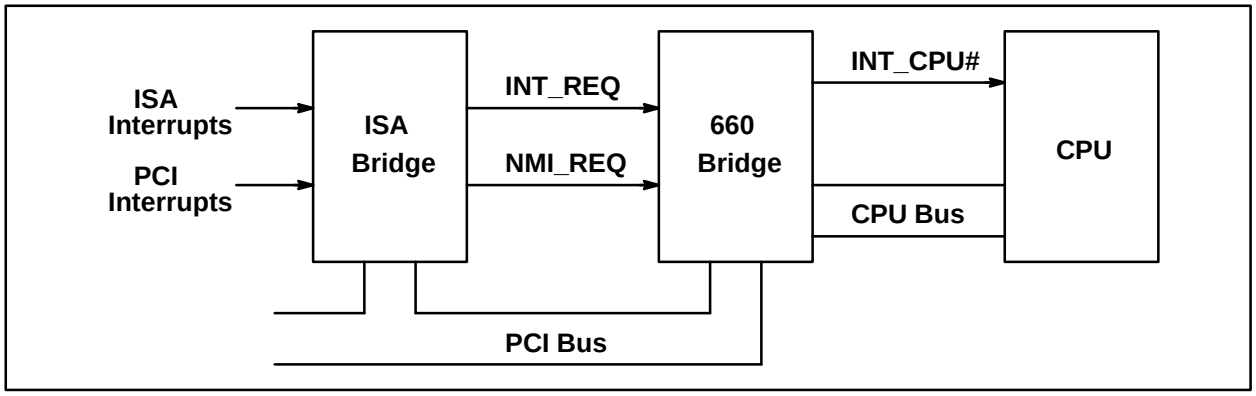


Figure 8-2. Interrupt Handling

As shown in Figure 8-2, the 100 MHz PPC 603e MCM interrupts are routed to the interrupt controller located inside the ISA bridge. When a device signals an interrupt (which is not masked in the interrupt controller), then for error sources, the interrupt controller is pro-

grammed to assert NMI_REQ. If the input to the interrupt controller signals an interrupt, then:

1. The ISA bridge asserts INT_REQ to the 660.
2. The 660 asserts INT_CPU# to the CPU.
3. The CPU recognizes the interrupt signal (INT#) immediately (or as soon as the MSR(EE) interrupt enable bit in the CPU is set to 1), saves its state, and then takes a precise external interrupt exception, branching to either 500h or FFF0 0500h, depending upon the Exception Prefix (EP) bit in the MSR. The MSR(EE) bit is automatically set to 0 at this time.
4. The code at the vector location requests a single-byte read of memory address BFFF FFF0h.
5. In response to the read, the 660 arbitrates for the PCI bus and then generates an interrupt acknowledge transaction on the PCI bus.
6. The ISA bridge decodes and claims the PCI interrupt acknowledge transaction, and returns the 8-bit vector which has been preprogrammed for the active interrupt, and then negates the interrupt output.
7. The 660 accepts the interrupt vector on the PCI bus, returns it to the CPU, and signals TA# to terminate the CPU transfer normally.

Since the CPU does not require that the interrupt signal (INT_CPU#) be deactivated between interrupts, another interrupt is allowed to occur as soon as software sets the MSR(EE) bit back to 1. For this reason, software should enable interrupts as soon as possible after receiving the vector. Note that the load instruction that fetches the interrupt vector is subject to out-of-order execution; eieio as required. After servicing the interrupt, execute a return from interrupt (RFI) instruction to return to the program that was interrupted. For more information on interrupts, see the Exceptions section of the 603 User's Manual.

Note that other PCI busmasters can initiate interrupt acknowledge transactions, but this may have unpredictable effects.

8.1.6 Planar Interrupt Assignments

In general, program ISA interrupts as edge sensitive. Program PCI interrupts as level sensitive. Interrupts are assigned to priority levels per ISA conventions. Table 8-1 shows the interrupt assignments of the planar. IRQ[0:7] connect to the master controller, and IRQ[8:15] connect to the cascaded controller. Figure 8-3 shows the connection of the PCI interrupts.

Table 8-1. Mapping of PCI Memory Space, Part 1

SIO IRQ #	Connects to	Priority	Assignment or (Comment)
0	no pin	1	Timer 1 Counter 0 (Internal to SIO).
1	Sys I/O EPLD	2	Keyboard
2	no pin	(3-10)	Cascade from controller 2
3	ISA IRQ3	11	(COM 2 or COM 4)
4	ISA IRQ4	12	(COM 1 or COM 3)
5	ISA IRQ5	13	(Parallel LPT 1 or 2)
6	ISA IRQ6	14	(Floppy)
7	ISA IRQ7	15	(Parallel LPT 2 or 3)
8#	RTC	3	TOD (aka Real Time Clock)
9	ISA IRQ9	4	
10	ISA IRQ10	5	(Audio)
11	ISA IRQ11	6	
12/M	ISA IRQ12	7	(Mouse)
13/FERR	—	8	Pulled up.
14	ISA IRQ14	9	(IDE)
15	ISA IRQ15	10	
PIRQ2#	PCI_INTC#		See Figure 8-3.
PIRQ1#	PCI_INTB#		See Figure 8-3.
PIRQ0#	PCI_INTA#		See Figure 8-3.

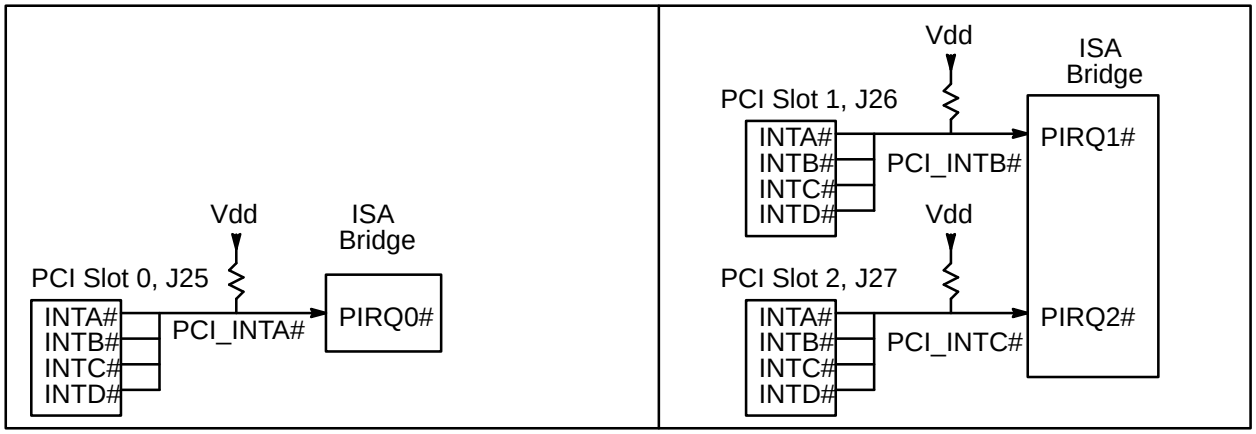


Figure 8-3. PCI Interrupt Connections

8.1.7 Scatter-Gather Interrupts

Where possible, set up the scatter-gather function to use the ISA bridge end of process (EOP output) indicator for the termination of ISA bus DMA in which scatter-gather is employed. The planar is initially configured to use this scheme. The EOP signal from the ISA bridge is used as the terminal count (ISA_TC) signal on the ISA bus.

8.2 Errors

Errors are handled primarily by the 660, the CPU, and the firmware. 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. There are two methods which the MCM (660) uses to report errors to the CPU, the TEA# method, and the MCP# method. Errors related to a PCI bus transaction are reported to the PCI by means of the PCI_PERR# or the PCI_SERR# signals.

Errors that are detected while the CPU is running a cycle that can be terminated immediately are reported using TEA#. 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#.

Errors that are detected while a CPU transfer is not in progress, and errors that occur because of a CPU transfer but which are detected too late to be reported using TEA#, and errors that are not a direct result of the current CPU transfer, are reported using MCP#. For example, memory parity errors occurring while a PCI busmaster is accessing memory are reported using MCP#.

There are three separate data error checking systems in the 660; CPU bus, memory, and PCI bus. The 660 does not generate or check CPU address bus parity.

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. For more information on error handing, see the 660 User's Manual.

8.2.1 CPU Bus Related Errors

8.2.1.1 CPU Bus Error Types

- **Errors Reported With TEA# (cycle still active)**
  - CPU bus unsupported transfer type
  - CPU bus unsupported transfer size
  - CPU bus XATS# asserted
- **Errors Reported With MCP# (cycle has ended)**
  - 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
  - PCI master abort generated (660 is PCI busmaster) during CPU to PCI transaction.

8.2.1.2 CPU Bus 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.

8.2.2 PCI Bus Related Errors

8.2.2.1 PCI Bus 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
- PCI memory select error
- Memory parity error
- Memory single–bit ECC error trigger exceeded
- Memory multi-bit ECC error

8.2.2.2 PCI Bus 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.
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 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.

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.

8.2.2.3 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.

8.2.3 CPU Bus Transaction Errors

8.2.3.1 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 8-2). An error is not generated if the cycle is claimed by another CPU device (CPU_BUS_CLAIM# asserted).

Table 8-2. 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 an 8-byte 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.

8.2.3.2 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.

8.2.3.3 CPU to Memory Writes

During CPU to memory writes, the CPU drives data parity information onto the CPU data bus. Correct parity is then generated in the 660 and written to DRAM memory along with the data. The L2 SRAM is updated (when required) with the data and the parity information that the CPU drove onto the CPU data bus.

During CPU to memory writes, the 660 checks the data parity sourced by the CPU, and normally reports any detected parity errors via TEA#.

8.2.3.4 CPU to Memory Reads

The MCM is initially configured to check memory data using parity. However, the 660 can be programmed to execute an error checking and correction (ECC) algorithm on the memory data, by generating ECC check bits during memory writes, and checking-correcting the data during memory reads. ECC can be implemented using normal parity DRAM.

Note that for each memory read operation, eight bytes of memory are read, and parity on eight bytes is checked regardless of the transfer size. Therefore, all of memory must be initialized (at least up to the end of any cache line that can be accessed). For the same reasons, memory must be initialized while ECC memory data error checking is in use.

When the CPU reads from memory, the data and accompanying parity information can come from either the L2 SRAM or from DRAM memory. If the data is sourced from the L2, the parity information also comes from the L2.

If the data is sourced by memory, the parity information also comes from memory. The L2 SRAM is updated (when required) using the data and parity from memory.

During CPU to memory reads, the 660 samples the DPE# output of the CPU to determine parity errors, and reports them back to the CPU via MCP#. The particular memory read data beat will be terminated normally with TA#.

8.2.3.5 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 8.2.3.6) 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.

8.2.3.6 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.

8.2.3.7 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 6.1.2 and 6.1.2.2). 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.

8.2.4 CPU to PCI Bus Transaction Errors

8.2.4.1 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#.

8.2.4.2 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. The CPU cycle is terminated with a TEA#, the Error Address register is held, and the Illegal Transfer Error register is set.

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.

8.2.4.3 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.

The 660 master aborts if no agent responds with DEVSEL# within eight clocks after the start of a CPU to PCI cycle. The cycle is ended with a TEA# response to the CPU, all 1's data is returned on reads, the Illegal Transfer Error register is set, and the Error Address register is held.

Note that some operating systems intentionally access unused addresses to determine what devices are located on the PCI bus. These operating systems do not expect an error to be generated by these accesses. When using such an operating system it is necessary to leave this error masked.

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.

The 660 also checks for bus hung conditions. If a CPU to PCI cycle does not terminate within approximately 60 usec. after the PCI is owned by the CPU, the cycle is terminated with TEA#. This is true for all CPU to PCI transaction types except configuration transactions. This feature may be disabled via a 660 control register.

In the case of configuration cycles that do not receive a DEVSEL# (no device present at that address), the PCI cycle is master aborted, and TA# (normal response) is returned.

Write data is thrown away and all 1's are returned on read cycles. No error register is set and no address is captured in the error address register.

8.2.5 PCI to Memory Transaction Errors

8.2.5.1 PCI to Memory Writes

During PCI to memory writes, the 660 generates the data parity that is written into DRAM memory. The bridge also checks the parity of the data, and asserts PCI_PERR# if it detects a data parity error.

8.2.5.2 PCI to Memory Reads

During PCI to memory reads, the 660 checks the parity of the memory data, and then generates the data parity that is driven onto the PCI bus. If there is a parity error in the data/parity returned to the 660 from the DRAM, the bridge drives PCI_PAR incorrectly to propagate the parity error (and also reports the error to the CPU via MCP#). The data beat with the bad parity is not target aborted because doing so would slow all data beats for one PCI clock (TRDY# is generated before the data is known good). However, if the agent is bursting and there is another transfer in the burst, the next cycle is stopped with target abort protocol.

During PCI to memory reads, the 660 also samples the PCI_PERR# signal, which other agents can be programmed to activate when they detect a PCI parity error.

8.2.5.3 Out of Bounds PCI Memory Accesses

If a PCI busmaster runs a cycle to a system memory address above the top of physical memory, no one will respond, and the initiator master aborts the cycle. The initiating busmaster must be programmed to notify the system of master aborts as needed. The system logic does not notify the CPU.

8.2.5.4 PCI Address Bus 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:

- 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.)
- 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.
- 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. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

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

8.2.5.5 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:

- 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.
- 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 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. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

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.

8.2.5.6 Errant Masters

Both PCI and ISA masters can access certain planar and ISA bridge registers. For example, various control registers such as the I/O Map Type register, the BE/LE mode bit, the Memory Control registers, etc. are accessible. Faulty code in the PCI or ISA masters can defeat password security, read the NVRAM, and cause the system to crash without recovery. Take care when writing device drivers to prevent these events.

8.2.6 Memory Transaction Errors

8.2.6.1 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 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.

8.2.6.2 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.

8.2.6.3 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 and the system memory address is saved in the single-bit ECC error address register (register CCh). If the count in the single-bit error counter register exceeds the value in the single-bit error trigger level register (B9h), the 660 generates a single–bit ECC trigger exceeded error.

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.

8.2.6.4 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.

8.2.7 SERR, I/O Channel Check, and NMI Errors

The PCI bus defines a signal called SERR# which any agent can pulse. This signal is to report error events within the devices, not bus parity errors. The signal is wired to the ISA bus bridge on the planar. The ISA bus signal IOCHCK is also wired to the ISA bridge. If either of these lines activate, the ISA bridge asserts NMI to the 660 unless the condition is masked by a register within the ISA bridge. The NMI signal causes the 660 to generate an interrupt to the CPU, and to assert MCP# to the CPU. The ISA bridge contains status registers to identify the NMI source. Software may interrogate the ISA bridge and other devices to determine the source of the error.

8.2.7.1 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.

8.2.8 Error Reporting Protocol

In general, when the 660 recognizes an error condition, it sets various status BCRs, saves address and control information (for 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.

8.2.8.1 Error Reporting With MCP#

The following errors are reported to the CPU using MCP#:

- NMI errors,
- Errors that occur because of a CPU transfer but which are detected too late to be reported using TEA#, and
- Errors that are not a direct result of the current CPU transfer.

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.

8.2.8.2 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#. 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.

8.2.8.3 Error Reporting With PCI_SERR#

The 660 asserts PCI_SERR# for one PCI clock when a 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.

8.2.8.4 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.

8.2.9 Error Status Registers

Error status registers in the 660 may be read to determine the types of outstanding errors. Errors are not accumulated while an error is outstanding; however, there will be one TEA# or MCP# for each error that occurs. For example, if an illegal transfer error causes a TEA#, a memory parity error can occur while the CPU is processing the code that handles the TEA#. The second error can occur before the error status registers are read. If so, then the second error status is not registered, but the MCP# from the memory parity error is asserted.

8.2.10 Error-Related Bridge Control Registers

Information on these registers is contained in the *660 Bridge Manual.*

Bridge Control Register	Index	R/W	Bytes
System Control 81C	8000 081C	R/W	1
Memory Parity Error Status	8000 0840	R	1
L2 Error Status	8000 0842	R	1
L2 Parity Error Read and Clear	8000 0843	R	1
Unsupported Transfer Type Error	8000 0844	R	1
System Error Address	BFFF EFF0	R	4
PCI Command	Index 04 – 05	R/W	2
PCI Device Status	Index 06 – 07	R/W	2
Single-Bit Error Counter	Index B8	R/W	1
Single-Bit Error Trigger Level	Index B9	R/W	1
Bridge Options 1	Index BA	R/W	1
Error Enable 1	Index C0	R/W	1
Error Status 1	Index C1	R/W	1
CPU Bus Error Status	Index C3	R	1
Error Enable 2	Index C4	R/W	1
Error Status 2	Index C5	R/W	1
PCI Bus Error Status	Index C7	R/W	1
CPU/PCI Error Address	Index C8 – CB	R/W	4
Single-Bit ECC Error Address	Indx CC – CF	R/W	4
Bridge Chip Set Options 3	Index D4	R/W	1

8.2.11 Special Events Not Reported as Errors

- A PCI to memory cycle at any memory address above that programmed into the top of memory register.

  The 660 ignores this cycle and the initiator master aborts. No data is written into system memory on writes, and the data returned on reads is indeterminate. The busmaster must be programmed to respond to a target abort by alerting the host.

- CPU to PCI configuration cycles to which no device responds with a DEVSEL# signal within 8 clocks (no device at this address)

  The data returned on a read cycle is all 1's and write data is discarded. This allows software to scan the PCI at all possible configuration addresses, and it is also consistent with the PCI specification.

- A CPU read of the IACK address having a transfer size other than 1, or having other than 4–byte alignment.

  These conditions return indeterminate data. The ISA bridge requires the byte enables, CBE#3:0, to be 1110 in order to place the data on the correct byte lane (0). Accesses other than one byte at the address BFFF FFF0h are undefined.

- A read of the IACK address when no interrupt is pending

  A DEFAULT 7 vector is returned in this case. This is the same vector that is returned on spurious interrupts.

- Parity error in Flash/ROM.

  Parity is not stored in the Flash ROM. Therefore the memory parity error signal and the DPE signal are ignored during ROM reads. The Flash or ROM should include CRC with software checking to insure integrity.

- Write to Flash with TSIZ other than 4.

  This will cause indeterminate data to be written into the Flash at an indeterminate address.

- Caching ROM space.

  An L1 or CB–L2 cast out will cause indeterminate results.

- Running any cycle to the PCI configuration space with an undefined address.

  Some of these could potentially cause damage. See the warning under the PCI configuration cycle section.

- Accessing any ISA device with the wrong data size for that device.

  Indeterminate results will occur.

8.3 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#.

8.3.1 CPU Reset

For information of CPU SRESET# and HRESET#, see the 603e user manual.

8.3.2 660 Reset

The entire 660 is reset to the correct initial state by the proper assertion of the RESET# input of the 664 (POWER_GOOD/RESET#). 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 8.3.2.3.

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 8.3.2.1, all 660 outputs maintain their reset state until an external stimulus (CPU bus activity) forces them to change.

8.3.2.1 Reset State of 660 Pins

The following symbols are used in Table 8-3 and Table 8-4:

—	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 8-3. 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

664 Signal	State
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

664 Signal	State
RESET#	0
ROM_LOAD	0
ROM_OE#	1
ROM_WE#	1
SBE#	—
SHD#	Z
SRAM_ADS#	0
SRAM_ALE	1
SRAM_CNT_EN#	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. Drive TEST# continuously high and MIO_TEST continuously low for correct operation.

Table 8-4. 663 Pin Reset State

663 Signal	State
AOS_RR_MMRS	—
C2P_WRL_OPEN	—
CPU_CLK	—
CPU_DATA[00:63]	Z
CPU_DATA_OE#	—
CPU_DPAR[0:7]	Z
CPU_PAR_ERR#	U
CPU_RDL_OPEN	—
CRS_C2PWXS	—
DUAL_CTRL_REF	—
ECC_LE_SEL	—

663 Signals	State
MEM_BE[0:1]	—
MEM_BE[2:3]	—
MEM_CHECK[0:7]	Z
MEM_DATA[63:0]	Z
MEM_DATA_OE#	—
MEM_ERR#	U
MEM_RD_SMPL	—
MEM_WRL_OPEN	—
MIO_TEST	0
MWS_P2MRXS	—

663 Signals	State
PCI_AD[31:0]	Z
PCI_AD_OE#	—
PCI_EXT_SEL	—
PCI_IRDY#	—
PCI_OL_OPEN	—
PCI_OUT_SEL	—
PCI_TRDY#	—
ROM_LOAD	—
SBE#	U
TEST#	1

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

8.3.2.2 660 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 8-5 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 8-5, 603(e) refers to either a 603 or a 603e.

Table 8-5. 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

8.3.2.3 660 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 8.3.2, 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:

  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.

  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.

8.4 Test Modes

8.4.1 CPU Test

The three test pins of the 603e (LLSD_MODE#, L1_TST_CLK#, & L2_TST_CLK#) are only supported for use during internal factory testing. They are not supported for customer use. Refer to the RISCWatch documentation for use of the JTAG port for debugging and diagnostic purposes.

8.4.2 660 Test

Tie the MIO_TEST input of both the 663 and the 664 low during normal operation. 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. MIO test mode is enabled on the 664 (asynchronously) by asserting MIO_TEST to the 664. IBM uses LSSD test mode to verify the switching levels of the inputs of the 663 and the 664.

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.

8.4.2.1 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 8-6 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 8-6. 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]

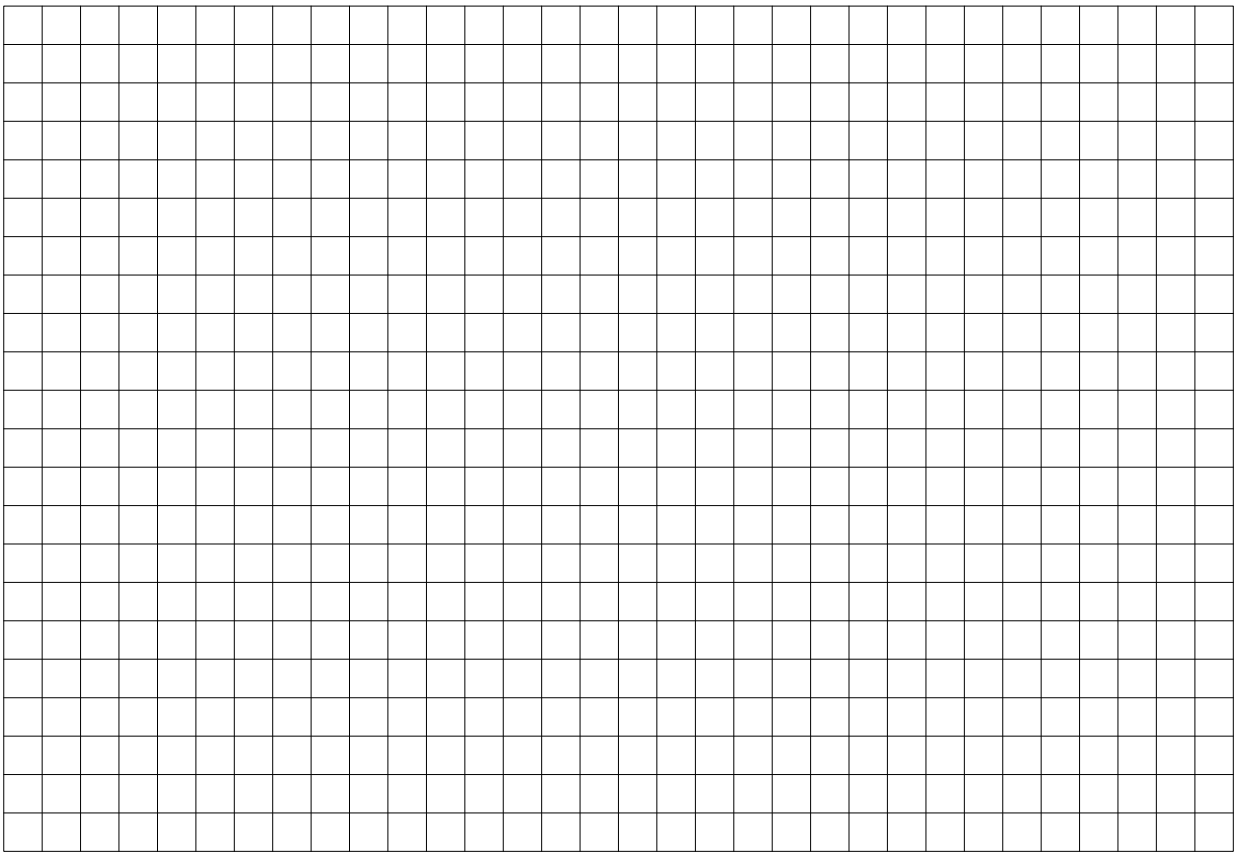
In LSSD test mode, never assert more than one of TEST_ACLK#, TEST_BCLK#, TEST_CCLK#, and RESET# at the same time, as this may damage the device by provoking excessive internal current flows.

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.

No further information on the use of the 660 test pins is expected to be released.



Section 9
Set Up and Registers

The following subsections represent activities carried out early in the boot firmware. On the MCM, all initialization registers are contained in the 603e nd the 660 bridge controller chips.

9.1 CPU Initialization

The 603 CPU exits the reset state with the L1 cache disabled and bus error checking disabled.

All memory pages 2G to 4G must be marked as non-cacheable.

The Segment Register T bit, bit 0, defaults to 0 which is the normal storage access mode. It must be left in this state for the hardware to function. Direct store (PIO) segments are not supported.

Set the bit that controls ARTRY# negation, HID0[7], to 0 to enable the precharge of ARTRY#. It may be necessary set HID0[7] to 1 to disable the precharge of ARTRY# for 100 MHz PPC 603e MCM configurations having a CPU bus agent (such as an added L2) that drives the ARTRY# line. Software must set this bit before allowing any CPU bus traffic to which the CPU agent might respond. Note that PCI to memory transactions cause the 660 bridge to broadcast snoop operations on the CPU bus.

HID0 bit 0, Master Checkstop Enable, defaults to 1 which is the enabled state. Leave it in this state so that checkstops can occur.

MCM errors are reported through the 660 by way of the TEA# and MCP# pins. Because of this, the bus error checking in the CPU must be disabled by setting HID0 bits 2 and 3 to zero.

9.2 660 Bridge Initialization

Before DRAM memory operations can begin, the software must:

1. Read the SIMM presence detect and SIMM type registers.
2. Set up and check the memory-related registers in the 660 (see the *660 Bridge User's Manual*).
3. Program the timer in the ISA bridge register which controls ISA refresh timing. In SIO compatible bridges it should be programmed to operate in Mode 2 with an interval of approximately 15 usec.
4. Make sure 200 usec has elapsed since starting the refresh timer so that sufficient refresh cycles have occurred to properly start the memory. This will be hidden if

approximately 120 Flash accesses occur after the timer is started and before the memory initialization starts.

5. Initialize all of memory so that all parity bits are properly set. (The CPU may cache unnecessary data; hence, all of memory must be initialized.) The 660 does not require reconfiguration when port 4Dh in the ISA bridge is utilized to reset the native I/O and the ISA slots.
6. Read the L2 cache presence detect bits and set register xx for 512K of synchronous SRAM cache.

9.3 PCI Configuration Scan

The 660 bridge in the MCM enables the software to implement a scan to determine the complement of PCI devices present. This is because the system returns all ones rather than an error when no PCI device responds to initialization cycles. The software may read each possible PCI device ID to determine devices present.

Table 9-1. Configuration Address Assignments

Device	IDSEL Line	60X Address*	PCI Address
ISA bus bridge (SIO)	A/D 11	8080 08XXh	080 08XX
PCI Slot 1	A/D 12	8080 10XXh	080 10XX
PCI Slot 2	A/D 13	8080 20XXh	080 20XX
PCI Slot 3	A/D 14	8080 40XXh	080 40XX

Note: ***This address is independent of contiguous I/O mode.

Software must use only the addresses specified. Using any addresses that causes more than one IDSEL to be asserted (high) can cause bus contention, because multiple PCI agents will be selected.

In systems that contain the Intel SIO chip, it must be configured prior to any other PCI bus agent. The SIO PCI arbiter is automatically enabled upon power-on reset. During power-on reset, the SIO drives the A/D(31:0), C/BE#(3:0), and PAR signals on the PCI bus.

9.3.1 Multi-Function Adaptors

The 660 supports multi-function adapters. It passes, unmodified, the address of the load or store instruction that causes a PCI configuration cycle. The only exception is that the three low-order bits are unmunged in little endian mode, and the two low-order address bits are set to zero in either endian mode; therefore, addresses may be selected with non-zero CPU address bits (21:23)—corresponding to PCI bits (10:8)—to configure multi-function adaptors. For example, to configure device 3 in slot 1, use address 80C0 03XXh. To configure device 7 in slot 2, use address 8084 07XXh.

9.3.2 PCI to PCI Bridges

The 660 supports both Type 0 and Type 1 configuration cycles.

9.3.3 Indexed BCR Summary

Table 9-2 contains a summary listing of the indexed BCRs in the 660. Access to these registers is described in the *660 Bridge User's Manual*. The values shown in the Set To column are for reference only, and may not apply to a particular application.

Table 9-2. 660 Bridge Indexed BCR Listing

Bridge Control Register	Index	R/W	Bytes	Set To (1)
PCI Vendor ID	Index 00 – 01	R	2	1014h
PCI Device ID	Index 02 – 03	R	2	0037h

Table 9-2. 660 Bridge Indexed BCR Listing (Continued)

Bridge Control Register	Index	R/W	Bytes	Set To (1)
PCI Command	Index 04 – 05	R/W	2	—
PCI Device Status	Index 06 – 07	R/W	2	—
Revision ID	Index 08	R	1	02h
PCI Standard Programming Interface	Index 09	R	1	0
PCI Subclass Code	Index 0A	R	1	0
PCI Class Code	Index 0B	R	1	06h
PCI Cache Line Size	Index 0C	R	1	0
PCI Latency Timer	Index 0D	R	1	0
PCI Header Type	Index 0E	R	1	0
PCI Built-in Self-Test (BIST) Control	Index 0F	R	1	0
PCI Interrupt Line	Index 3C	R	1	0
PCI Interrupt Pin	Index 3D	R	1	0
PCI MIN_GNT	Index 3E	R	1	0
PCI MAX_LAT	Index 3F	R	1	0
PCI Bus Number	Index 40	R	1	0
PCI Subordinate Bus Number	Index 41	R	1	0
PCI Disconnect Counter	Index 42	R/W	1	0
PCI Special Cycle Address BCR	Index 44 –45	R	2	0
Memory Bank 0 Starting Address	Index 80	R/W	1	Memory
Memory Bank 1 Starting Address	Index 81	R/W	1	Memory
Memory Bank 2 Starting Address	Index 82	R/W	1	Memory
Memory Bank 3 Starting Address	Index 83	R/W	1	Memory
Memory Bank 4 Starting Address	Index 84	R/W	1	Memory
Memory Bank 5 Starting Address	Index 85	R/W	1	Memory
Memory Bank 6 Starting Address	Index 86	R/W	1	Memory
Memory Bank 7 Starting Address	Index 87	R/W	1	Memory
Memory Bank 0 Ext Starting Address	Index 88	R/W	1	Memory
Memory Bank 1 Ext Starting Address	Index 89	R/W	1	Memory
Memory Bank 2 Ext Starting Address	Index 8A	R/W	1	Memory
Memory Bank 3 Ext Starting Address	Index 8B	R/W	1	Memory
Memory Bank 4 Ext Starting Address	Index 8C	R/W	1	Memory
Memory Bank 5 Ext Starting Address	Index 8D	R/W	1	Memory
Memory Bank 6 Ext Starting Address	Index 8E	R/W	1	Memory
Memory Bank 7 Ext Starting Address	Index 8F	R/W	1	Memory
Memory Bank 0 Ending Address	Index 90	R/W	1	Memory
Memory Bank 1 Ending Address	Index 91	R/W	1	Memory
Memory Bank 2 Ending Address	Index 92	R/W	1	Memory
Memory Bank 3 Ending Address	Index 93	R/W	1	Memory
Memory Bank 4 Ending Address	Index 94	R/W	1	Memory
Memory Bank 5 Ending Address	Index 95	R/W	1	Memory
Memory Bank 6 Ending Address	Index 96	R/W	1	Memory
Memory Bank 7 Ending Address	Index 97	R/W	1	Memory

Table 9-2. 660 Bridge Indexed BCR Listing (Continued)

Bridge Control Register	Index	R/W	Bytes	Set To (1)
Memory Bank 0 Ext Ending Address	Index 98	R/W	1	Memory
Memory Bank 1 Ext Ending Address	Index 99	R/W	1	Memory
Memory Bank 2 Ext Ending Address	Index 9A	R/W	1	Memory
Memory Bank 3 Ext Ending Address	Index 9B	R/W	1	Memory
Memory Bank 4 Ext Ending Address	Index 9C	R/W	1	Memory
Memory Bank 5 Ext Ending Address	Index 9D	R/W	1	Memory
Memory Bank 6 Ext Ending Address	Index 9E	R/W	1	Memory
Memory Bank 7 Ext Ending Address	Index 9F	R/W	1	Memory
Memory Bank Enable	Index A0	R/W	1	Memory
Memory Timing 1	Index A1	R/W	1	0001 0010
Memory Timing 2	Index A2	R/W	1	1000 1010
Memory Bank 0 & 1 Addressing Mode	Index A4	R/W	1	Mode 2
Memory Bank 2 & 3 Addressing Mode	Index A5	R/W	1	—
Memory Bank 4 & 5 Addressing Mode	Index A6	R/W	1	—
Memory Bank 6 & 7 Addressing Mode	Index A7	R/W	1	—
Cache Status	Index B1	R/W	1	—
Refresh Cycle Definition	Index B4	R	1	—
Refresh Timer B5 (Not used – see Indexed BCR D0)	Index B5	R	1	—
RAS Watchdog Timer	Index B6	R/W	1	53h
PCI Bus Timer (Not used)	Index B7	R	1	0
Single-Bit Error Counter	Index B8	R/W	1	0
Single-Bit Error Trigger Level	Index B9	R/W	1	0
Bridge Options 1	Index BA	R/W	1	07h
Bridge Options 2	Index BB	R/W	1	5Fh
Error Enable 1	Index C0	R/W	1	EDh
Error Status 1	Index C1	R/W	1	0
Error Simulation 1	Index C2	R/W	1	0
CPU Bus Error Status	Index C3	R	1	14h
Error Enable 2	Index C4	R/W	1	04h
Error Status 2	Index C5	R/W	1	04h
Error Simulation 2	Index C6	R/W	1	0
PCI Bus Error Status	Index C7	R/W	1	0Fh
CPU/PCI Error Address	Index C8–CB	R/W	4	—
Single-Bit ECC Error Address	Index CC – CF	R/W	4	—
Refresh Timer Divisor	Index D0 – D1	R/W	2	0180h
Suspend Refresh Timer	Index D2 – D3	R/W	2	01F8h
Bridge Chip Set Options 3	Index D4	R/W	1	88h

Notes:

1. In this column, a long dash — means that the initialization firmware does not write to this register. The register is either not used, not written to, or the value of it depends on changing circumstances.
   If the word Memory appears, please refer to the System Memory section of the 660 User's Manual.
2. The initialization firmware sets the Memory registers depending on the information reported by the DRAM presence detect registers.

Section 10
System Firmware

10.1 Introduction

The firmware on the 100 MHz PPC 603e MCM planar handles three major functions:

- Test the system in preparation for execution,
- Load and execute an executable image from a bootable device, and
- Allow user configuration of the system.

Section 10.2 briefly discusses the power on system test function.

Section 10.3 details a structure for boot records which can be loaded by the system firmware.

Section 10.4 describes the system configuration utility.

This information is included for reference only. Some of the information in this section concerns the example planar rather than the MCM.

To obtain a copy of the commented source code of the firmware on diskette, contact your IBM representative. This material is available free of charge with a signed license agreement.

10.2 Power On System Test

The Power On System Test (POST) code tests those subsystems of the reference board which are required for configuration and boot to ensure minimum operability. Tests also assure validity of the firmware image and of the stored system configuration.

10.2.1 Hardware Requirements

In addition to the reference board, the firmware requires the following peripherals to be installed as adapter cards:

• Serial Port 1	Address: 0x3F8 (COM1:)	Interrupt: IRQ 4
• Serial Port 2	Address: 0x2F8 (COM2:)	Interrupt: IRQ 3
• Floppy Controller	Address: 0x3F0 (Primary Floppy)	Mode: PC/AT or PS/2
• IDE Controller	Address: 0x1F0 (Primary IDE)	

10.3 Boot Record Format

The firmware will attempt to boot an executable image from devices specified by the user. See Section 10.4 for details on specifying boot devices and order.

The *PowerPC Reference Platform Specification* details a structure for boot records which can be loaded by the system firmware. This specification is described in the following sections.

10.3.1 Boot Record

The format of the boot record is an extension of the PC environment. The boot record is composed of a PC compatibility block and a partition table. To support media interchange, the PC compatibility block may contain an x86-type program. The entries in the partition table identify the PowerPC Reference Platform boot partition and its location in the media.

The layout of the boot record must be designed as shown in Figure 10-1. The first 446 bytes of the boot record contain a PC compatibility block, the next four 16-byte entries make up a partition table totalling 64 bytes, and the last 2 bytes contain a signature.

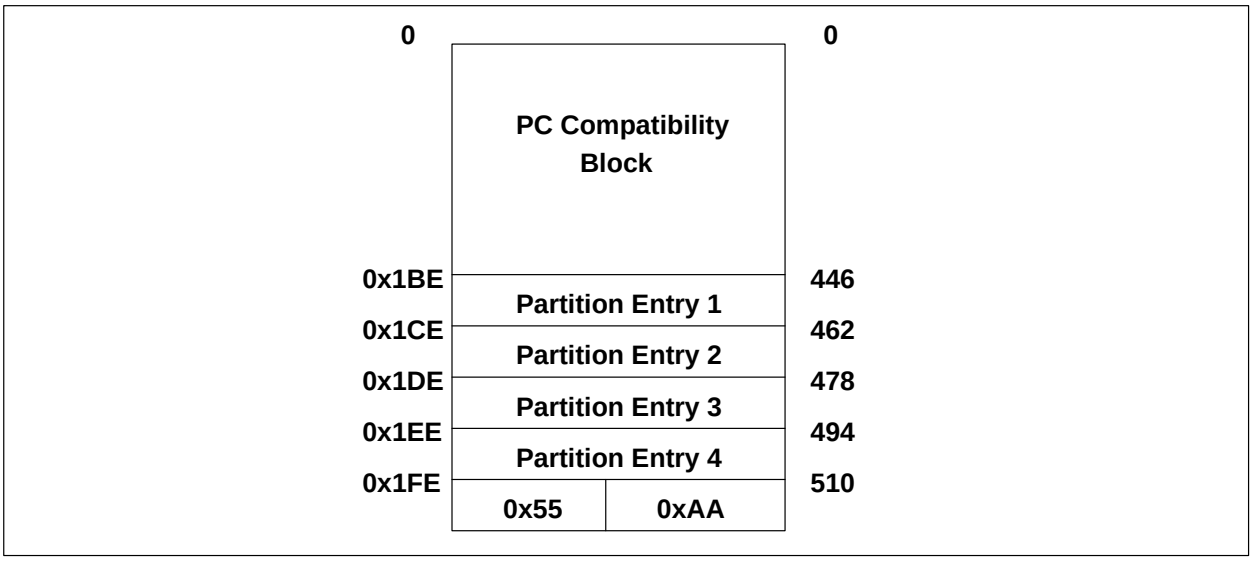


Figure 10-1. Boot Record

10.3.1.1 PC Partition Table Entry

To support media interchange with the PC, the PowerPC Reference Platform defines the format of the partition table entry based on that for the PC. This section describes the format of the PC partition table entry, which is shown in Figure 10-2.

	0				
Partition Begin	4	Boot Ind	Head	Sector	Cyl
Partition End	8	Sys Ind	Head	Sector	Cyl
Beginning Sector	12	Low Word (LE)		High Word (LE)	
Number of Sectors		Low Word (LE)		High Word (LE)	

LE = Little-Endian

Figure 10-2. Partition Table Entry

- Partition Begin — The beginning address of the partition in head, sector, cylinder notation.
- Partition End — The end address of the partition in cylinder, head, sector notation.
- Beginning Sector — The number of sectors preceding the partition on the disk. That is, the zero-based relative block address of the first sector of the partition.
- Number of Sectors — The number of sectors allocated to the partition.

The subfields of a partition table entry are defined as follows:

- Boot Ind — Boot Indicator. This byte indicates if the partition is active. If the byte contains 0x00, then the partition is not active and will not be considered as bootable. If the byte contains 0x80, then the partition is considered active.
- Head — An eight-bit value, zero-based.
- Sector — A six-bit value, one-based. The low-order six bits are the sector value. The high-order two bits are the high-order bits of the 10-bit cylinder value.
- Cyl — Cylinder. The low-order eight-bit component of the 10-bit cylinder value (zero-based). The high-order two bits of the cylinder value are found in the sector field.
- Sys Ind — System Indicator. This byte defines the type of the partition. There are numerous partition types defined. For example, the following list shows several:

0x00	Available partition
0x01	DOS, 12-bit FAT
0x04	DOS, 16-bit FAT
0x05	DOS extended partition
0x41	PowerPC Reference Platform partition.

10.3.1.2 Extended DOS Partition

The extended DOS partition is used to allow more than four partitions in a device. The boot record in the extended DOS partition has a partition table with two entries, but does not contain the code section. The first entry describes the location, size and type of the partition. The second entry points to the next partition in the chained list of partitions. The last partition in the list is indicated with a system indicator value of zero in the second entry of its partition table.

Because of the DOS format limitations for a device partition, a partition which starts at a location beyond the first 1 gigabyte is located by using an enhanced format shown in Figure 10-3.

Partition Begin	0	Boot Ind	–1	–1	–1
Partition End	4	Sys Ind	–1	–1	–1
Beginning Sector	8	32–bit start RBA (zero–based) (LE)			
Number of Sectors	12	32–bit RBA count (one–based) (LE)			

LE = Little-Endian

–1 = All ones in the field.
RBA = Relative Block Address in units of 512 bytes.

Figure 10-3. Partition Table Entry Format for an Extended Partition

10.3.1.3 PowerPC Reference Platform Partition Table Entry

The Power PC Reference Platform partition table entry (see Figure 10-4) is identified by the 0x41 value in the system indicator field. All other fields are ignored by the firmware except for the Beginning Sector and Number of Sectors fields. The CV (Compatible Value – not shown) fields must contain PC-compatible values (i.e. acceptable to DOS) to avoid confusing PC software. The CV fields, however, are ignored by the firmware.

Partition Begin	0	Boot Ind	Head	Sector	Cyl
Partition End	4	Sys Ind	Head	Sector	Cyl
Beginning Sector	8	32–bit start RBA (zero–based) (LE)			
Number of Sectors	12	32–bit RBA count (one–based) (LE)			

RBA = Relative Block Address in units of 512 bytes.

LE = Little-Endian

Figure 10-4. Partition Table Entry for PowerPC Reference Platform

The 32-bit start RBA is zero-based. The 32-bit count RBA is one-based and indicates the number of 512-byte blocks. The count is always specified in 512-byte blocks even if the physical sectoring of the target devices is not in 512-byte sectors.

10.3.2 Loading the Load Image

This section describes the layout of the PowerPC 0x41 type partition and the process of loading the load image.

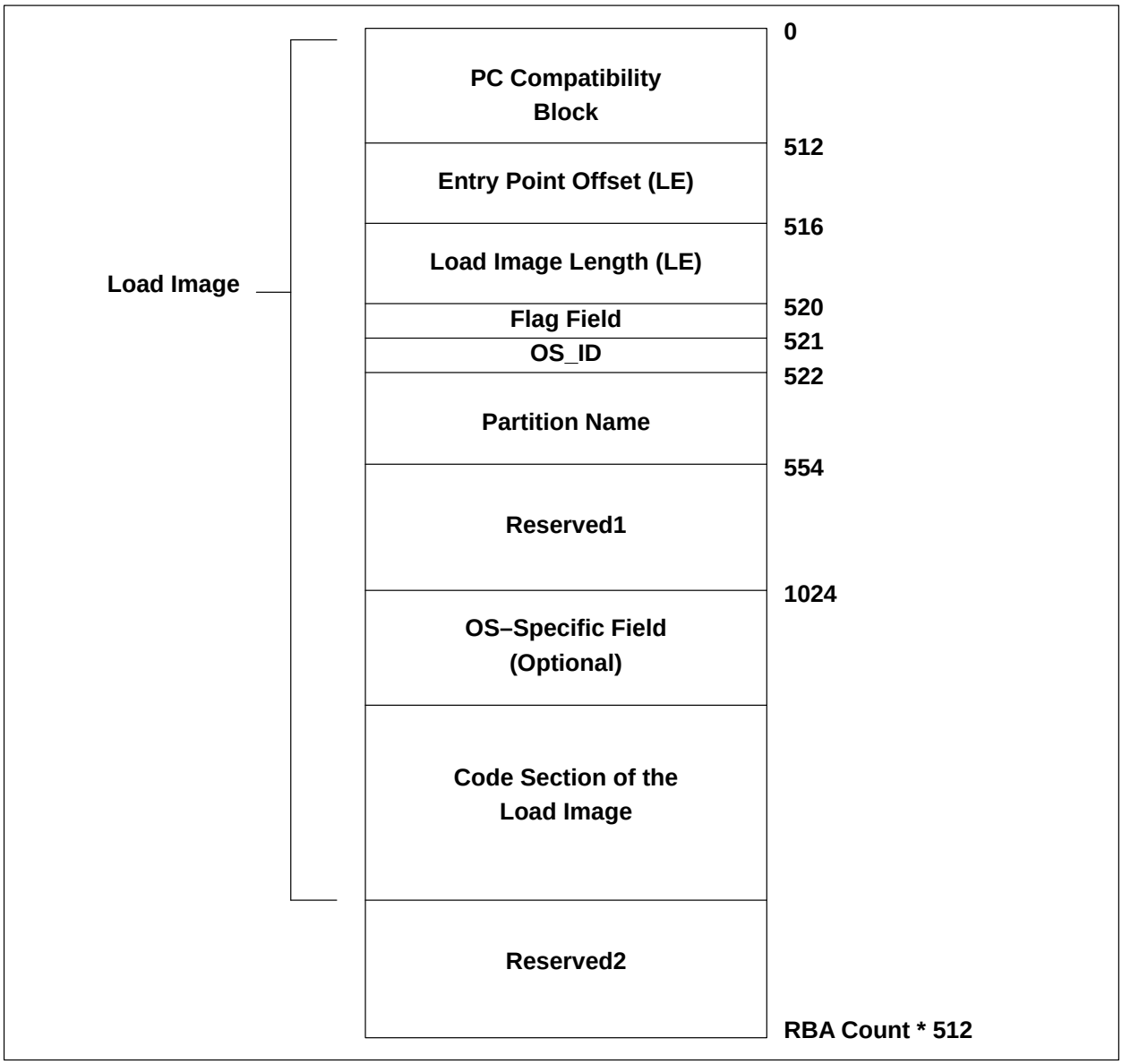


Figure 10-5. PowerPC Reference Platform Partition

The layout for the 0x41 type partition is shown in Figure 10-5. The PC compatibility block in the boot partition may contain an x86-type program. When executed on an x86 machine, this program displays a message indicating that this partition is not applicable to the current system environment.

The second relative block in the boot partition contains the entry point offset, load image length, flag field, operating system ID field, ASCII partition name field, and the reserved1 area. The 32-bit entry point offset (little-endian) is the offset (into the image) of the entry point of the PowerPC Reference Platform boot program. The entry point offset is used to allocate the Reserved1 space. The reserved1 area from offset 554 to Entry Point – 1 is reserved for implementation specific data and future expansion.

The 32-bit load image length (little-endian) is the length in bytes of the load image. The load image length specifies the size of the data physically copied into the system RAM by the firmware.

The flag field is 8 bits wide. The MSb in the field is allocated for the Open Firmware flag. If this bit is set to 1, the loader requires Open Firmware services to continue loading the operating system.

The second MSb is the endian mode bit. If the mode bit is 0, the code in the section is in big-endian mode. Otherwise, the codes is in little-endian mode. The implication of the endian mode bit is different depending on the Open Firmware flag. If the Open Firmware flag is set to 1, the mode bit indicates the endian mode of the code section pointed to by the load image offset, and the firmware has to establish the hardware endian mode according to this bit. Otherwise, this bit is just an informative field for firmware.

The OS_ID field and partition name field are used to identify the operating system located in the partition. The OS_ID field has the numeric identification value of the operating system located in the partition. The 32 byes of partition name field must have the ASCII notation of the partition name. The name and OS_ID can be used to provide to a user the identification of the boot partition during the manual boot process.

Once the boot partition is identified by the PowerPC Reference Platform boot partition table entry, the firmware:

- Reads into memory the second 512-byte block of the boot partition
- Determines the load image length for reading in the boot image up to but not including the reserved2 space
- Allocates a buffer in system RAM for the load image transfer (no fixed location)
- Transfers the load image into system RAM from the boot device (the reserved2 space is not loaded).

The load image must be fully relocatable, as it may be placed anywhere in memory by the system firmware. Once loaded, the load image may relocate itself anywhere within system RAM.

10.4 System Configuration

This section describes the utilities in the system firmware which allow the system to be customized. These utilities allow viewing of the system configuration, as well as the ability to change I/O device configurations, console selection, boot devices, and the date and time. These functions are described in the following sections.

10.4.1 System Console

The system console can be either a screen-oriented video display or a line-oriented serial terminal. The example screens shown in this section show the S3 video/keyboard interface. When using a serial terminal, the configuration utilities will prompt for numeric input for each prompt instead of using the arrow keys. All choices and options are the same as for the screen-oriented menus.

The configuration of the reference board as shipped is set for S3 video / Keyboard console. In the case that either the video adapter or the keyboard fails the power-on test, the system

console will default to serial port 1. The baud rate for the serial console is specified in the configuration menus. The value as shipped is 9600 baud.

10.4.2 System Initialization

The logo screen, shown in Figure 10-6, is displayed at power-on. The logo screen is active while the system initializes and tests memory and performs a scan of the SCSI bus to determine what SCSI devices are installed.

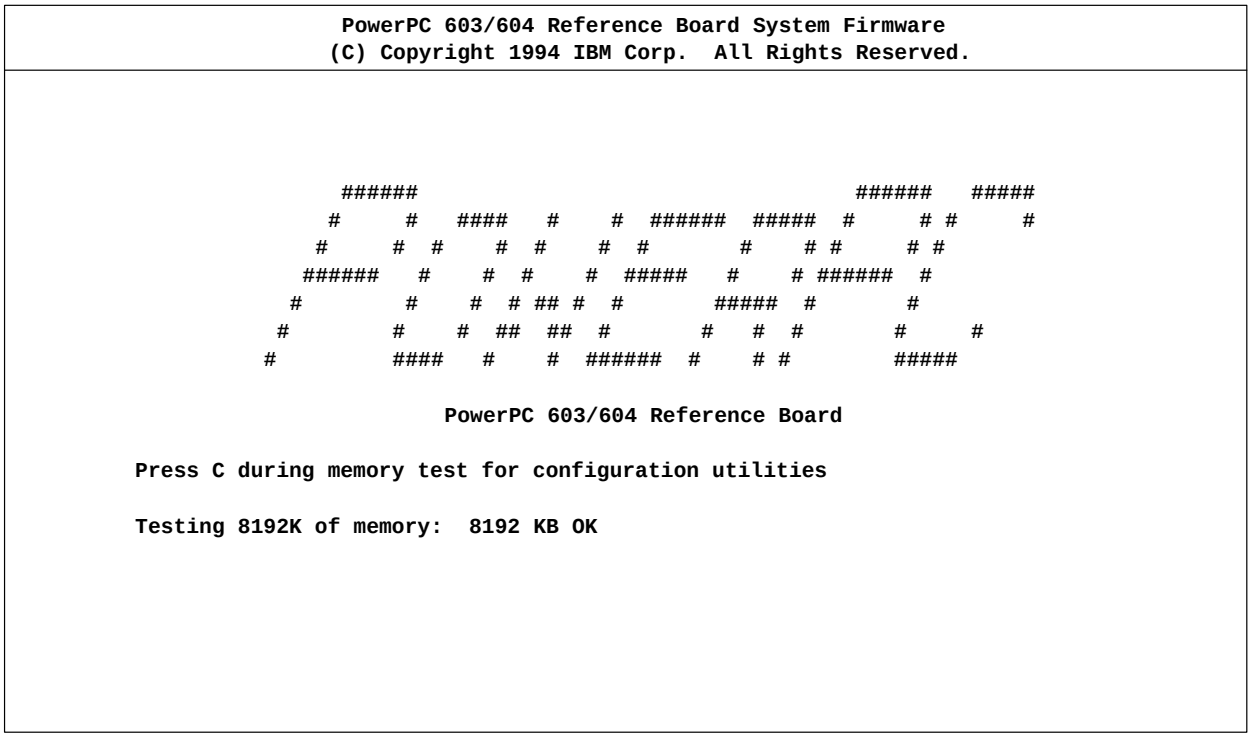


Figure 10-6. System Initialization Screen

While the logo screen is displayed, pressing the 'C' key on the console will enter the system configuration utility. The configuration menu will also be entered if there is no bootable device present, or if the configuration stored in the system non-volatile RAM is not initialized or is corrupt.

10.4.3 Main Menu

Figure 10-7 shows the main menu for the system configuration utility. Selections on the menu are highlighted by using the up and down arrow keys on the keyboard, and are chosen with the Enter key. Each choice is detailed in the following sections.

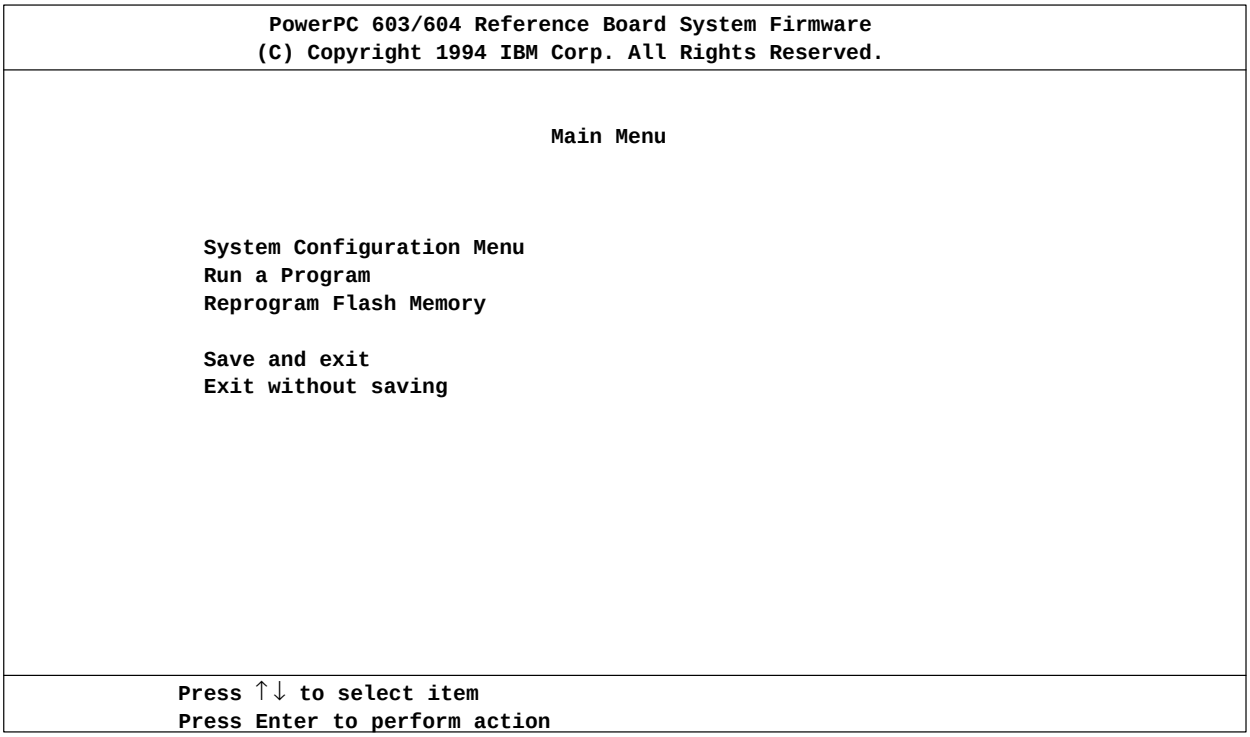


Figure 10-7. Configuration Utility Main Menu

10.4.3.1 System Configuration Menu

Figure 10-8 shows the System Configuration menu, which has choices to display and change the default state of the reference board on boot. Each menu item is discussed in the following sections.

```
                    PowerPC 604 Reference Board System Firmware
                (C) Copyright 1994 IBM Corp.  All Rights Reserved.

                                System Configuration Menu


        System Information
        Configure I/O Devices
        View SCSI Devices
        Set Boot Devices
        Set Date and Time

        Previous Menu









        Press ↑↓ to select item
        Press Enter to perform action
```

Figure 10-8. System Configuration Menu

System Information

The system configuration option shows the hardware configuration of the system at power-up—including processor, installed options, and firmware revision level. A sample screen is shown in Figure 10-9.

```
                PowerPC 603/604 Reference Board System Firmware
                (C) Copyright 1994 IBM Corp. All Rights Reserved.

                        System Configuration


          System Processor    PowerPC 604
          Installed Memory    8 MB
          Second-Level Cache   Not Installed
          Upgrade Processor    Not Installed
          Boot Firmware Revision 1.0

          Go to Previous Menu

          Press ↑↓ to select item
          Press Enter to perform action
```

Figure 10-9. System Information Screen

Configure I/O Devices

The configure I/O devices option allows the customization of system I/O ports and the system console. The menu is shown in Figure 10-10. Options are highlighted by using the up and down arrow keys on the keyboard and are changed with the left and right arrow keys. Options on the menu are discussed below.

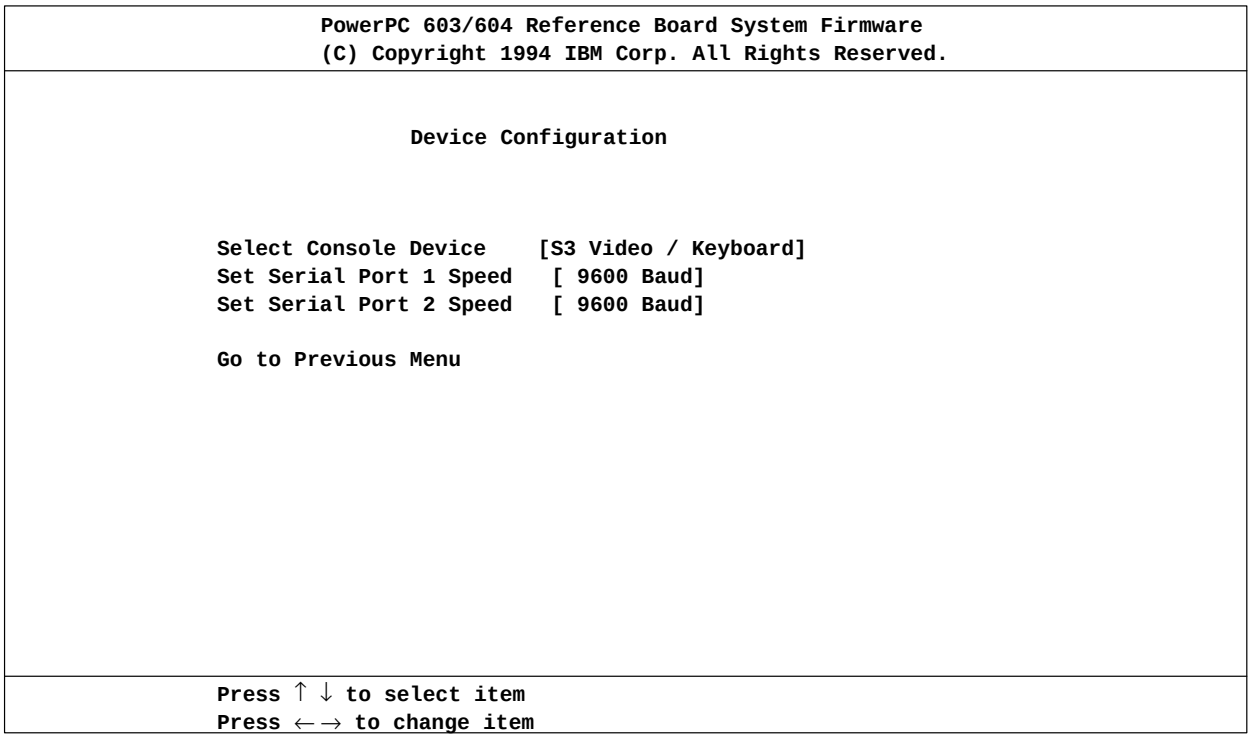


Figure 10-10. Device Configuration Screen

Any changes made in I/O device configuration are saved when the Save and Exit option on the main menu is selected. Exiting the system configuration utility in any other manner will cause device configuration changes to be lost.

Select Console Device

The console selection box allows the selection of an option for the system console

- Serial Port 1 or 2 — Console input and output will be transmitted and received through a serial port on an adapter card. Console input and output will be transmitted and received at the baud rate selected with Serial Port Speed.
- S3 Video/Keyboard — Console output will be displayed on a video monitor connected to an S3 PCI video adapter; console input will be received from a keyboard connected to the keyboard connector on the reference board

Set Serial Port 1 or 2 Speed

The serial port speed selection box sets the speed of each serial port. Baud rates for the two serial ports are independent. If a serial port is used as the system console, set this value to match the baud rate of the terminal.

View SCSI Devices

The SCSI devices screen shows the devices found on the SCSI bus during power-on initialization. The string shown is the SCSI device's response to the SCSI inquiry command. According to the SCSI specification, this data comprises the manufacturer's ID, device model number, and device revision level. A sample screen is shown in Figure 10-11.

```
                PowerPC 603/604 Reference Board System Firmware
                (C) Copyright 1994 IBM Corp. All Rights Reserved.

                              SCSI Devices


          SCSI Device 0    None
          SCSI Device 1    None
          SCSI Device 2    None
          SCSI Device 3    None
          SCSI Device 4    None
          SCSI Device 5    None
          SCSI Device 6    IBM MXT-540SL H

          Previous Menu

          Press ↑↓ to select item
          Press Enter to perform action
```

Figure 10-11. SCSI Devices Screen

Set Boot Devices

The boot devices menu allows the user to select which devices are queried for boot images and in what order they are selected for boot. Allowable selections are one of the two floppy disk drives, any of six SCSI drive ID numbers, either of two IDE disk drives, or no device selected. The default configuration is shown in Figure 10-12. In this configuration, the system will attempt to find a boot image on the first floppy disk drive. If this fails, the system will attempt to boot from the SCSI device programmed to SCSI ID 6. If this fails, the system will attempt to boot from IDE drive zero (master).

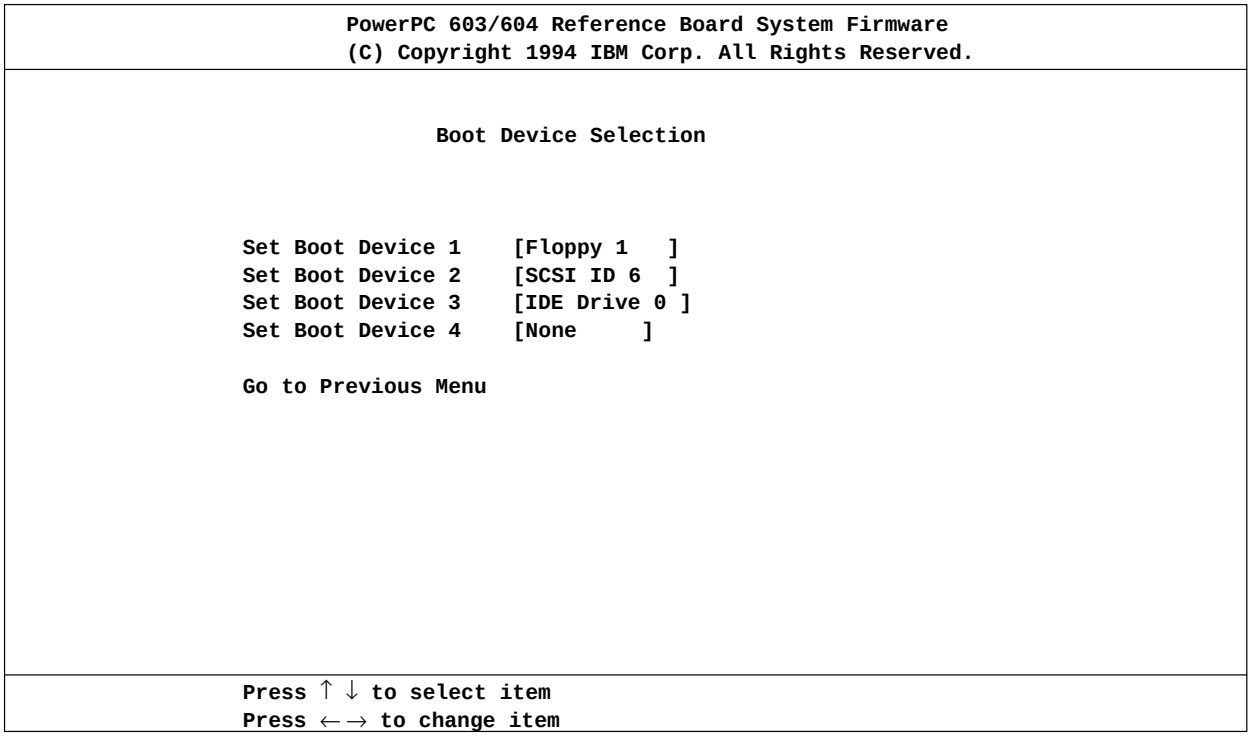


Figure 10-12. Boot Devices Screen

If the system fails to find a valid boot image (as discussed in Section 10.3) on any of the selected boot devices, or if no boot device is selected, the user will be prompted to enter the configuration menu to select a valid boot device.

Any changes made in boot device selection is saved when the Save and Exit option on the main menu is selected. Exiting the system configuration utility in any other manner will cause boot device changes to be lost.

Set Date and Time

The set date and time screen allows the date and time stored in the battery-backed real time clock to be updated. The screen is shown in Figure 10-13. To change the time, the left and right arrow keys are used to select the digit to modify, and the digit is then typed over with the number keys. The date or time will be updated when Enter or either the up or down arrow is pressed. Changing the date or time is immediate, and is not affected by either the Save and Exit or Exit Without Saving options on the main menu.

```
                PowerPC 603/604 Reference Board System Firmware
                (C) Copyright 1994 IBM Corp. All Rights Reserved.

                          Set Date and Time


          Set Date  [03/01/94]
          Set Time  [11:30:00]

          Go to Previous Menu

          Press ↑↓ to select item
          Enter data at cursor
```

Figure 10-13. Set Date and Time Screen

10.4.3.2 Run a Program

The Run a Program option on the main menu loads and executes a program from a FAT (DOS) disk or from a CD-ROM in ISO-9660 format. The program is loaded at location 0x00400000 (4 MB) and control is passed with a branch to the first address.

All boot devices specified in the Boot Devices Menu will be searched in order for FAT and CD-ROM file systems, and the first matching file on a boot device will be loaded.

The Run a Program screen is shown in Figure 10-14. To run a program, enter the file name in the Specify Program Filename field and select the Run the Program option.

```
PowerPC 603/604 Reference Board System Firmware
(C) Copyright 1994 IBM Corp. All Rights Reserved.

                          Run a Program


Specify Program Filename [          ]
Run the Program

Go to Previous Menu

Press ↑↓ to select item
Enter data at cursor
```

Figure 10-14. Run a Program Screen

10.4.3.3 Reprogram Flash Memory

The PowerPC 603/604 reference board stores its system firmware in a reprogrammable flash memory on the system board. The reprogram flash memory option on the main menu allows the reprogramming of the flash device with a DOS-formatted diskette. This allows future revisions of the system firmware to be provided on diskette without the need for removal of the device from the board.

If done improperly, reprogramming the flash memory can cause the system to become unusable until external means are available to reprogram the device. Use this option with care.

All boot devices specified in the Boot Devices Menu will be searched in order for FAT and CD-ROM file systems, and the first matching file on a boot device will be loaded.

The Reprogram the Flash Memory screen is shown in Figure 10-15. To reprogram the flash, enter the file name in the Specify Image Filename field and select the Reprogram the Memory option.

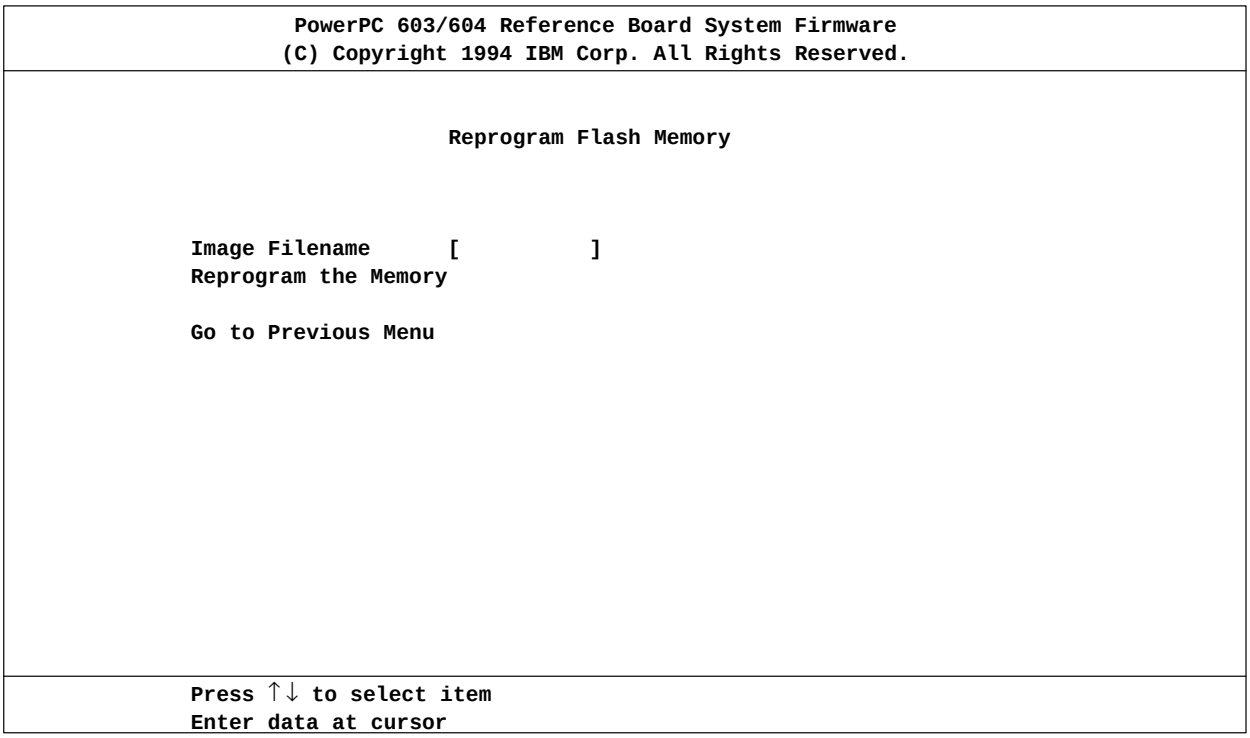


Figure 10-15. Reprogram the Flash Memory Screen

10.4.3.4 Exit Options

The two exit options at the bottom of the main menu leave the system configuration utility. The two options are:

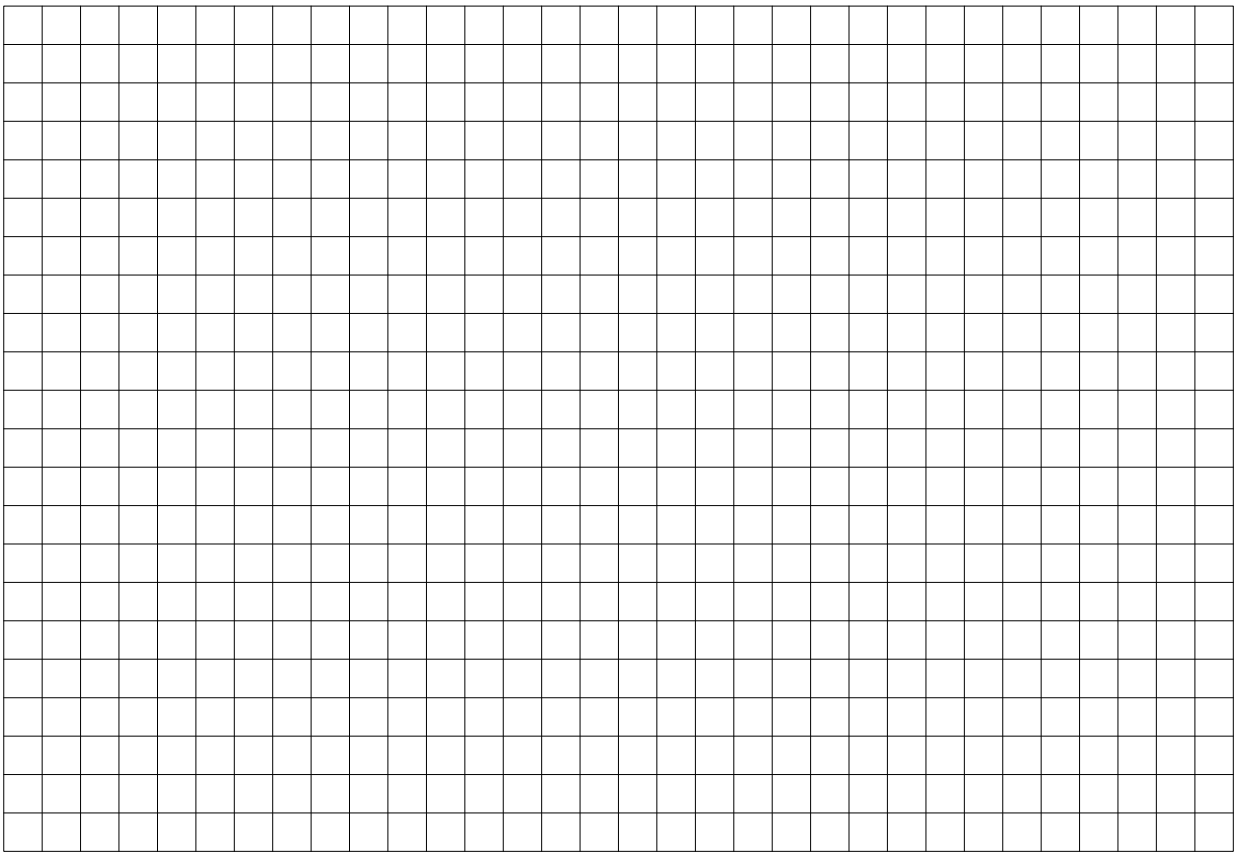
- Save and Exit — Saves any changes made in the Configure I/O Devices and Set Boot Devices screens, and restarts the system.
- Exit without Saving — Proceeds with the boot process as if the configuration utility had not been entered. Any changes made in Configure I/O Devices or Set Boot Devices are lost.

10.4.4 Default Configuration Values

When the PowerPC 603/604 reference board is shipped from the factory, it has the following default configuration:

- Console Device — S3 Video / Keyboard
- Serial Port 1 — 9600 Baud
- Serial Port 2 — 9600 Baud
- Boot Devices — Device 1 - Floppy 1
Device 2 - SCSI ID 6
Device 3 - IDE Drive 0

These default values also take effect whenever the system configuration in system nonvolatile RAM becomes corrupted.



Section 11
Endian Mode Considerations

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.

The 100 MHz PPC 603e MCM normally operates with big endian (BE) byte significance, which is the native mode of the PowerPC 603e 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 Bridge 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 MCM also supports this mode of operation.

When the MCM 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). Also, see the *603e CPU* and *660 Bridge User's Manuals*.

Table 11-1 summarizes the operation of the MCM in the two different modes.

Table 11-1. Endian Mode Operations

Mode	What the 603e 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. This is the native mode of the 603e CPU. In BE mode, 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.

11.1 What the 603e CPU Does

11.1.1 The 603e Address Munge

The 603e 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 11-2 shows the address transform rules for the allowed LE mode transfer sizes.

Table 11-2. 603e 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]

11.1.2 The 603e 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.

11.2 What the 660 Bridge Does

While the MCM 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 603e.

11.2.1 The 660 Bridge 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.

11.2.2 The 660 Bridge Data Swapper

The 660 contains a byte swapper. As shown in Figure 11-1, 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 MCM 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.

The 660 cannot tell the endian mode of the CPU directly, and so cannot automatically change endian mode to match the CPU. There is a control bit located in ISA I/O space (port 0092) that the CPU can write to in order to set the endian mode of the motherboard.

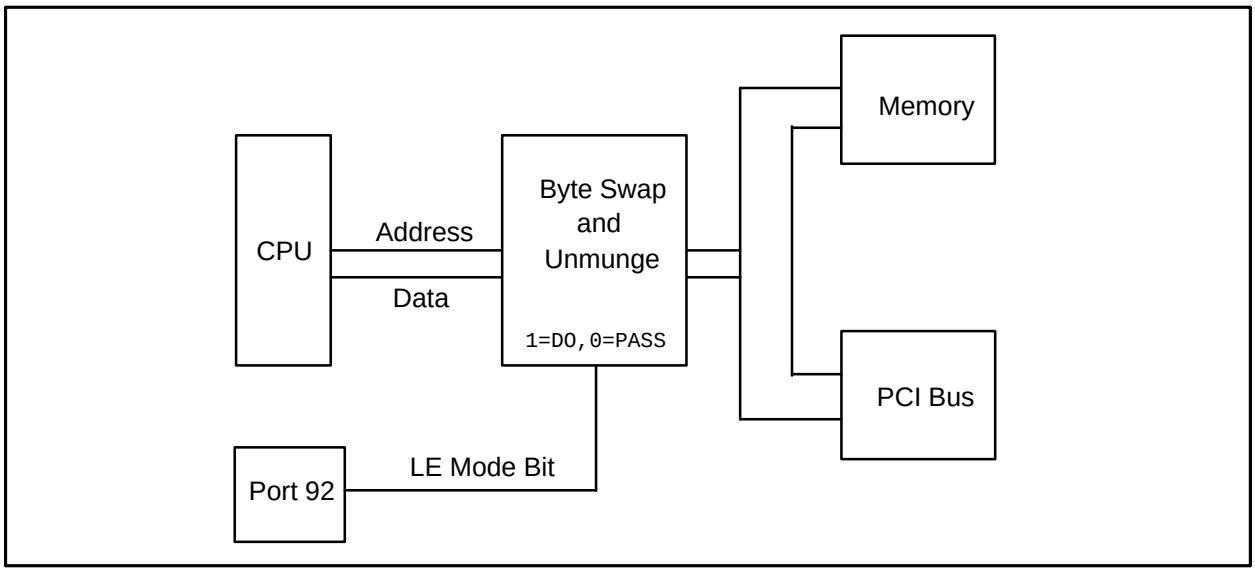


Figure 11-1. Endian Mode Block Diagram

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 11-3, 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.

Table 11-3. 660 Bridge Endian Mode Byte Lane Steering

CPU Byte Lane	BE Mode Connection	LE Mode Connection
CPU byte lane 0 (MSB)	Memory byte lane 0, PCI lane 0	Memory byte lane 7, PCI lane 7*
CPU byte lane 1	Memory byte lane 1, PCI lane 1	Memory Byte lane 6, PCI lane 6*
CPU byte lane 2	Memory byte lane 2, PCI lane 2	Memory byte lane 5, PCI lane 5*
CPU byte lane 3	Memory byte lane 3, PCI lane 3	Memory byte lane 4, PCI lane 4*
CPU byte lane 4	Memory byte lane 4, PCI lane 4*	Memory byte lane 3, PCI lane 3
CPU byte lane 5	Memory byte lane 5, PCI lane 5*	Memory byte lane 2, PCI lane 2
CPU byte lane 6	Memory byte lane 6, PCI lane 6*	Memory byte lane 1, PCI lane 1
CPU byte lane 7 (LSB)	Memory byte lane 7, PCI lane 7*	Memory byte lane 0, PCI lane 0

Note: * 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, as coded in the instruction. PCI byte lanes [7:4] refer to the data bytes associated with PCI_C/BE[3:0]# when PCI_AD[29] is a 1.

11.3 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 11-4 for both BE and LE mode operation.

Table 11-4. 660 Bit Transfer

CPU_DATA[ ]	BE Mode MEM_DATA[ ]	LE Mode MEM_DATA[ ]
0	7	63
1	6	62
2	5	61
3	4	60
4	3	59
5	2	58
6	1	57
7	0	56
8	15	55
9	14	54
10	13	53
11	12	52
12	11	51
13	10	50
14	9	49
15	8	48
16	23	47
17	22	46
18	21	45
19	20	44
20	19	43
21	18	42
22	17	41
23	16	40
24	31	39
25	30	38
26	29	37
27	28	36
28	27	35
29	26	34
30	25	33
31	24	32

CPU_DATA[ ]	BE Mode MEM_DATA[ ]	LE Mode MEM_DATA[ ]
32	39	31
33	38	30
34	37	29
35	36	28
36	35	27
37	34	26
38	33	25
39	32	24
40	47	23
41	46	22
42	45	21
43	44	20
44	43	19
45	42	18
46	41	17
47	40	16
48	55	15
49	54	14
50	53	13
51	52	12
52	51	11
53	50	10
54	49	9
55	48	8
56	63	7
57	62	6
58	61	5
59	60	4
60	59	3
61	58	2
62	57	1
63	56	0

11.4 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:

- 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.

11.5 603e CPU Alignment Exceptions In LE Mode

The CPU does not support a number of instructions and data alignments in the LE mode that it supports in BE mode. When it encounters an unsupportable situation, it takes an internal alignment exception (machine check) and does not produce an external bus cycle. See the latest 603e CPU documentation for details. Examples include:

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

11.6 Single-Byte Transfers

Figure 11-2 is an example of byte write data a at address xxxx xxx0**.**

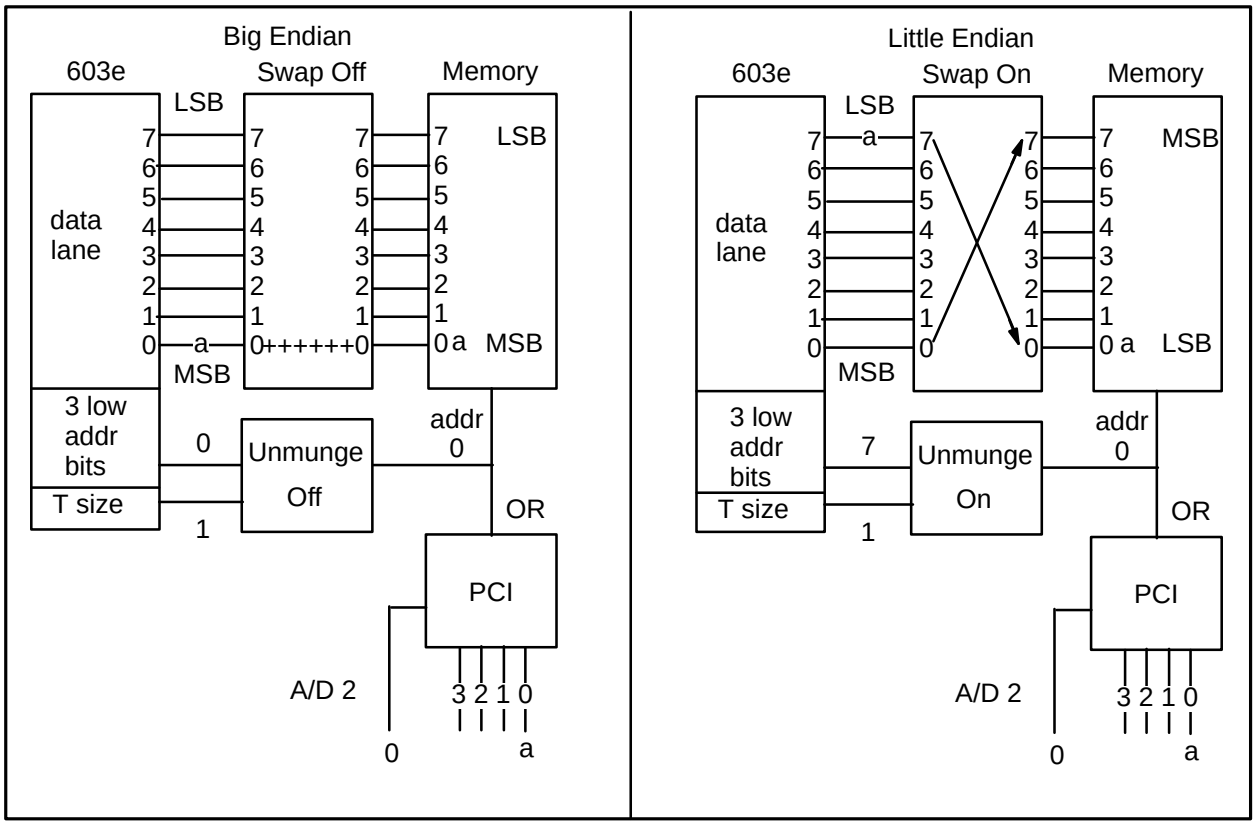


Figure 11-2. Example at Address xxxx xxx0

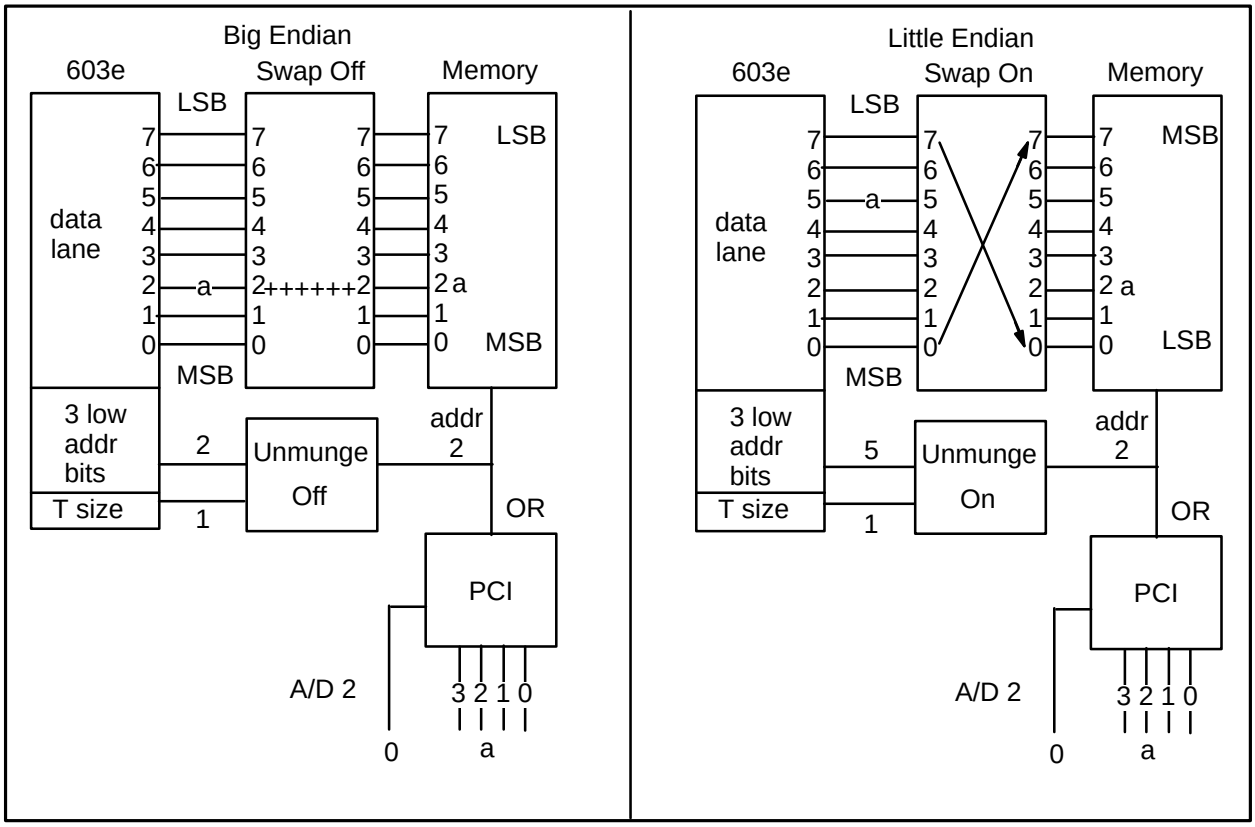


Figure 11-3. Example at Address xxxx xxx2

Figure 11-3 is an example of byte write data a at address xxxx xxx2.

For single byte accesses to memory in BE mode, Table 11-5 applies.

Table 11-5. Memory in BE Mode

603e A31 30 29	603e add	BYTE LANE	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

Note: *At the CPU side.

For single byte accesses to memory in LE mode, Table 11-6 applies.

Table 11-6. Memory in LE Mode

603e A31 30 29	603e add	BYTE LANE	BYTE LANE*	663 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			ON	SWAP	UNMUNGED

Note: *At the CPU side.

For single byte accesses to PCI in BE mode, Table 11-7 applies.

Table 11-7. PCI in BE Mode

603e A31 30 29	603e add	BYTE LANE	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	

Note: **AD[0:1] set to 00 for all PCI transactions except I/O cycles.

For single byte accesses to PCI in LE mode, Table 11-8 applies.

Table 11-8. PCI in LE Mode

603e A31 30 29	603e add	663* BYTE LANE	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	

Notes:
***** At the CPU side.
****** AD[0:1] set to 00 for all PCI transactions except I/O cycles.

11.7 Two-Byte Transfers

Figure 11-4 gives an example of double byte write data ab at address xxxx xxx0.

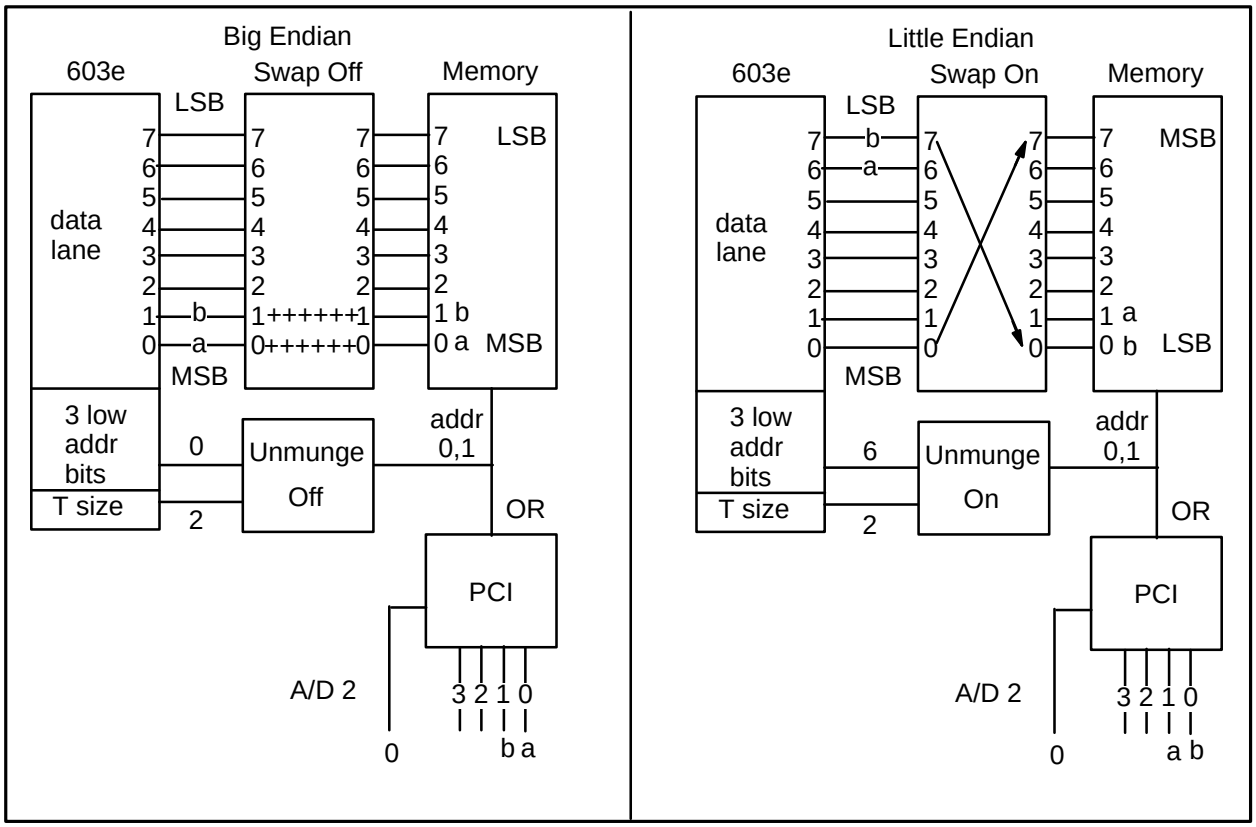


Figure 11-4. Double Byte Write Data ab at Address xxxx xxx0

Table 11-9 and Table 11-10 illustrate all cases that can occur. The columns of Table 11-9 have these meanings:

- The first column indicates target address (e.g. the address of the byte coded into a store half-word instruction).
- The next two columns show the state of the address pins for BE mode.
- The next two columns show the state of the address pins for the same target data when the machine is in LE mode.
- The remaining columns show the CASs and the PCI byte enables associated with the target data.
- The notes indicate which combinations either do not occur at the pins because of internal exceptions, or are not supported externally.

For 2-byte transfers, Table 11-9 holds:

Table 11-9. Two Byte Transfer Information

PROG TARG ADDR	BE MODE 603e add	BE a29:31	LE MODE (x or w 110) 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	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

Notes:
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 603e bus) in LE mode

Table 11-10 contains the same information as found in Table 11-9, 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 603e and as a function of BE/LE mode.

Table 11-10. Rearranged Two-Byte Transfer Information

2 BYTE XFERS 60X ADDRESS PINS	BE CAS#0:7		BE PCI CBE#		LE CAS#0:7		LE PCI	CBE#
	0	7	A2	3210	0	7	AD2	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

Notes:
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 603e bus) in LE mode

11.8 Four-Byte Transfers

Figure 11-5 gives an example of Word (4-BYTE) Write of 0a0b0c0dh AT ADDRESS xxxx xxx4.

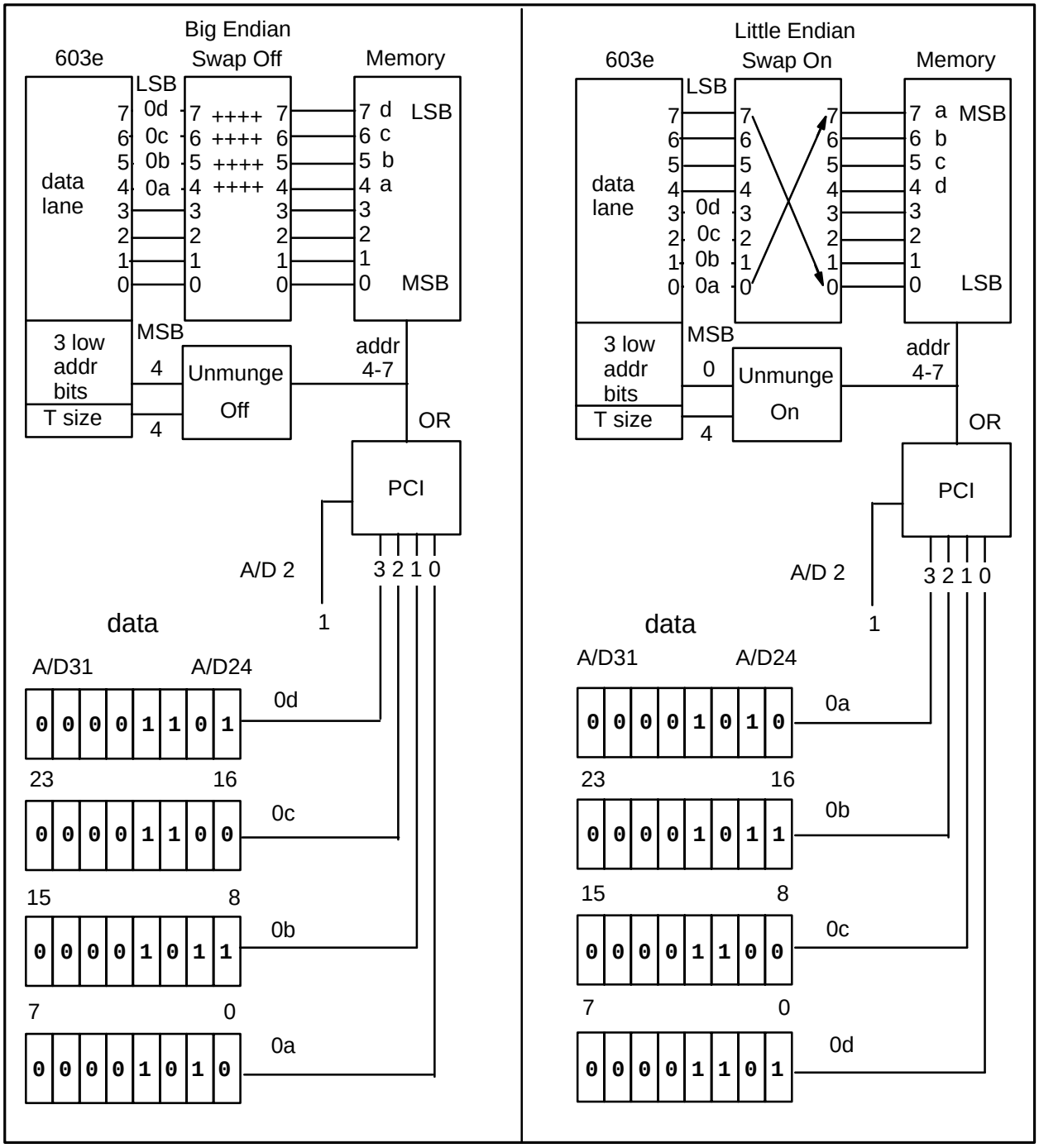


Figure 11-5. Word (4-Byte) Write of 0a0b0c0dh at Address xxxx xxx4

Table 11-11 and Table 11-12 illustrate the cases that can occur. The columns of Table 11-11 have these meanings:

- The first column indicates the target address (e.g. the address of the byte coded into a store word instruction).
- The next two columns show the state of the address pins for BE mode.
- The next two columns show the state of the address pins for the same target data when the machine is in LE mode.
- The remaining columns show the CASs and the PCI byte enables associated with the target data.
- The notes indicate which combinations either do not occur at the 603e pins because of internal exceptions, or are not supported externally.

Table 11-11. Four-Byte Transfer Information

PROG TARG ADDR	BE MODE 603e 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

Notes:
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 603e bus) in LE mode

Table 11-12 contains the same information as found in Table 11-11, 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 603e and as a function of BE/LE mode.

Rearranging Table 11-12 for 4-byte transfers:

Table 11-12. Rearranged Four-Byte Transfer Information

4 BYTE XFERS		BE		BE		LE		LE	
60X ADDRESS PINS		CAS#0:7		PCI CBE#		CAS#0:7		PCI CBE#	
		0	7	A2	3210	0	7	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

Notes:
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 603e bus) in LE mode
X= not supported in memory controller (crosses 4-byte boundary

11.9 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.

11.10 Instruction Fetches and Endian Modes

Most instruction fetching is with cache on. Therefore memory is fetched eight bytes wide. Figure 11-6 shows the instruction alignment.

Example: 8 byte instruction fetch I1=abcd, I2=efgh at address xxxx xxx0

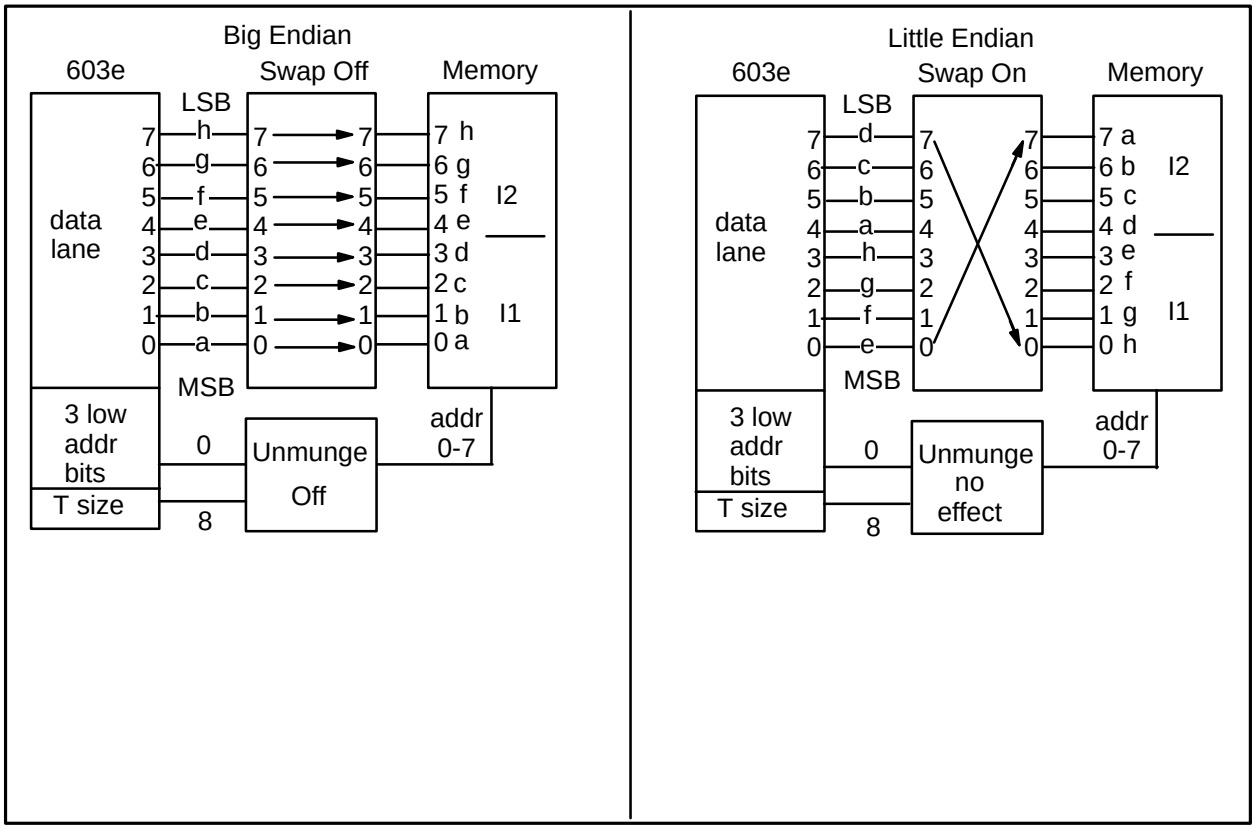


Figure 11-6. Instruction Alignment Example

It is possible to fetch instructions with 4 byte aligned transfers when the cache is turned off. In that case, the 603e does not munge the address in LE mode. The memory controller does not differentiate between instruction and data fetches, but the unmunger is ineffective because the memory is always read 8 byte wide, and data is presented on all 8 byte lanes. If the unmunger were used, the wrong instruction would be read. The net result is illustrated in Figure 11-7.

Example: 4 byte instruction fetch, I2=efgh at address xxxx xxx4

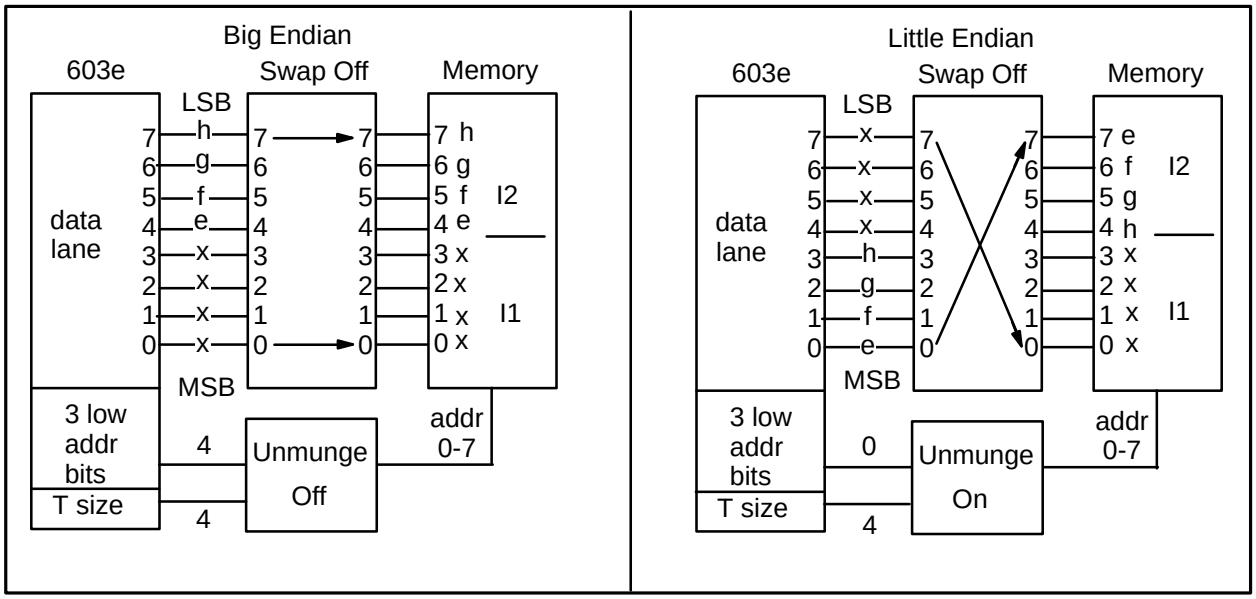


Figure 11-7. Wrong Instruction Read When Unmunger is used

11.11 Changing BE/LE Mode

There are two BE/LE mode controls. One is inside the 603e CPU and the other is a register bit on the motherboard. The 603e CPU interior mode is not visible to the motherboard hardware. The BE mode bit referred to in this document is the register bit on the motherboard. It is a bit in I/O space which is memory mapped just like other I/O registers. It defaults to BE mode.

The 603e CPU always powers up in the BE mode and begins fetching to fill its cache. Consequently, at least the first of the ROM code must be BE code. It is beyond the scope of this document to define how the system will know to switch to LE mode; however, great care must be made during the switch in order to synchronize the internal and external mode bits, to flush all caches, and to avoid executing extraneous code.

The following process switches the system from BE to LE mode when used in this system:

1. Disable L1 caching.
2. Disable L2 caching.
3. Flush all system caches.
4. Turn off interrupts immediately after a timer tick so no timer interrupts will occur during the next set of cycles.
5. Mask all interrupts.

6. Set the CPU state and the motherboard to LE (see Figure 11-8). Note that CPU is now in LE mode. All instructions must be in LE order.
7. Put interrupt handlers and CPU data structures in LE format.
8. Enable caches.
9. Enable Interrupts.
10. Start the LE operating system initialization.

Figure 11-8 shows the instruction stream to switch endian modes.

```
        x       mfspr   R2,1008        ;Load the HDO register
;Instructions to set the Little-Endian bit in R2
        0       sync
        4       sync
        8       sync
        C       mtspr   1008,R2        ;Moves to HID0 register
        10      sync
        14      sync
        18      sync
        1c      sync
        20      Store to external Endian control port (X8000 0092)
;The above instruction must be on a double word boundary
;So the following instruction is executed first (due to pipeline)
        24      eieio
; To this point all instructions are in Big Endian format
; The following instructions look the same in either Endian mode
        28      X38010138
        2C      X38010138
        ...     ;Enough of these instructions must be executed
        ...     ;to guarantee the above store has occurred.
;
;before any memory or I/O cycles are listed.
        xx      X38010138
```

Figure 11-8. Instruction Stream to Switch Endian Modes

11.12 Summary of Bi-Endian Operation and Notes

- When the 603e CPU is in BE mode, the memory is in BE mode, and data flowing on the PCI is in BE order so that it is recorded on the media in BE order. Byte 0 is the most significant byte.
- When the 603e CPU is in LE mode, the memory is in LE mode, and data flowing on the PCI is in LE order so that it is recorded on the media in LE order. Byte 0 is the least significant byte.
- The PCI bus is addressed in the same manner that memory is when the 603e CPU runs a cycle. The unmunging in LE mode changes the effective low-order address bits (the byte enables and A/D 2). On all but I/O cycles, the two low-order A/D lines are set to zero. On PCI I/O cycles, A/D 1,0 are also transformed by the unmunge operation
- No translations are made when PCI accesses memory so that the byte with address 0 on the PCI flows to byte 0 in memory — 1 to 1, 2 to 2, and so on. For example, if BE0# and BE1# are active and A/D 2 is a 0, then memory byte lanes 0 and 1 are addressed (cas 0 and cas 1 active on writes).
- Note that the LE devices which interpret data structures in the memory require that their control data be arranged in LE order even in BE mode. For example, SCSI scripts in memory must always be arranged in LE order because that is what the device expects.
- Devices such as video may require the bytes to be swapped unless these devices have byte swap capability.

Section 12
Electromechanical

12.1 MCM Electrical

The electrical characteristics of the MCM IOs are obtained by reference to the electrical characteristics of the individual devices to which a given IO is connected. The ratings of the MCM IOs, including the package effects, are better than or equal to the ratings of the individual devices. Please refer to the appropriate sections in the following documents:

- IBM27-82660 User's Manual
- PowerPC 603e (PID6–603e) Hardware Specifications
- TI 74LVT16245 Data Sheet
- IBM IBM041814 SRAM Data Sheet
- IDT IDT71216 TagRAM Data Sheet
- Motorola MPC970 Data Sheet.

The conditions under which the MCM operates must not exceed the Absolute Maximum Ratings of any of the individual devices.

The AC and DC Electrical Specifications, and Timing Specifications of each IO is affected by the combined characteristics of the devices that are attached to that net. For example, the characteristics of the PCI_AD lines are the sum of the characteristics of the 663 and the 664. The characteristics of the memory lines are those stated in the 660 User's Manual. The characteristics of the CPU bus lines are the sum of the characteristics of all of the attached devices.

For the specifications in the individual documents to apply, the MCM must be operated within the Recommended Operating Conditions of each of the individual devices.

12.1.1 AC Electrical Characteristics

The single-chip module (SCM) data sheets for the die packaged in the MCM are contained in Appendix G. As a first approximation, it is accurate to use AC characteristics (setup and hold) specifications from the SCM data sheets when doing a system timing analysis of the MCM. The MCM package contains both the semiconductiors and the wiring network interconnecting them.

Overall, a system timing budget is much improved by the density of the wiring; however, when referenced from the package I/O, MCM AC characteristics differ (in theory) from those listed in the SCM data sheets. Actually, such differences are less than 0.5 nanosec-

onds due to the short net lengths in the MCM ceramic substrate and to the extremely small paracitic capacitances of flip chip packaging.

In order to adjust SCM AC characteristics to values appropriate to the MCM package, subtract out the SCM package delay component of a timing parameter then add back in the corresponding delay for the MCM.

Example//??//

Package modeling information that allows such an analysis is found in Section 12.1.2. Note that the dynamic interfaces between the MCM and a system planar consist of the PCI port and the system memory port. Both interfaces are contained within the 660 bridge components. For this reason, only a paper model of the 660 SCM package is included here for use in package delay comparison.

12.1.2 SCM Package Delay Modeling

Figure 12-1 is a paper electrical model of the quad flatpack package that houses the 660 bridge die. Typical delays attributed to this package are //??// nanoseconds.

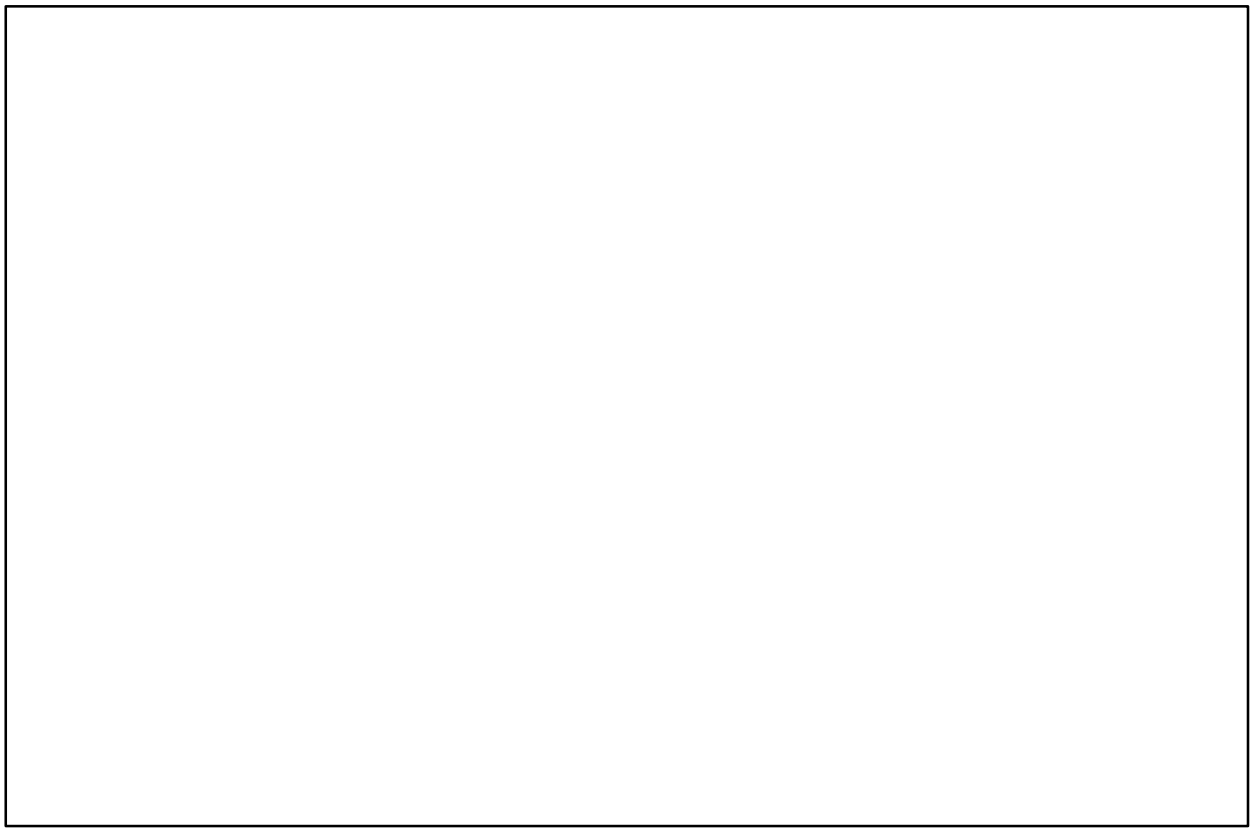


Figure 12-1. QFP Package

12.1.3 MCM Package Delay Modeling

To provide information needed to correctly model the MCM package, a paper model of the MCM package has been included in Figure 12-1. In this model, a transmission line is used to model the effects of a net trace in ceramic. The length of this transmission line is different for each signal that is considered. Refer to Table 12-1 for a listing of the net lengths for all MCM signals to which the user is expected to interface.

12.1.4 MCM Net Electrical Model

Figure 12-1 shows a typical electrical model of the major nets on the MCM. This model applies to the PCI, DRAM, and clock net groups. The segment labeled C shows the effects of the chip IO pad. The Redistribution segment shows the effects of the redistribution metalization that is added to the chips to produce flip-chips. The Ceramic Vias segments show the effects of vias in the MCM ceramic substrate. The Routed Signal segment shows the effects of trace lengths in the ceramic. The Solder Column segment shows the effects of the solder columns. For net lengths see Table 12-1.

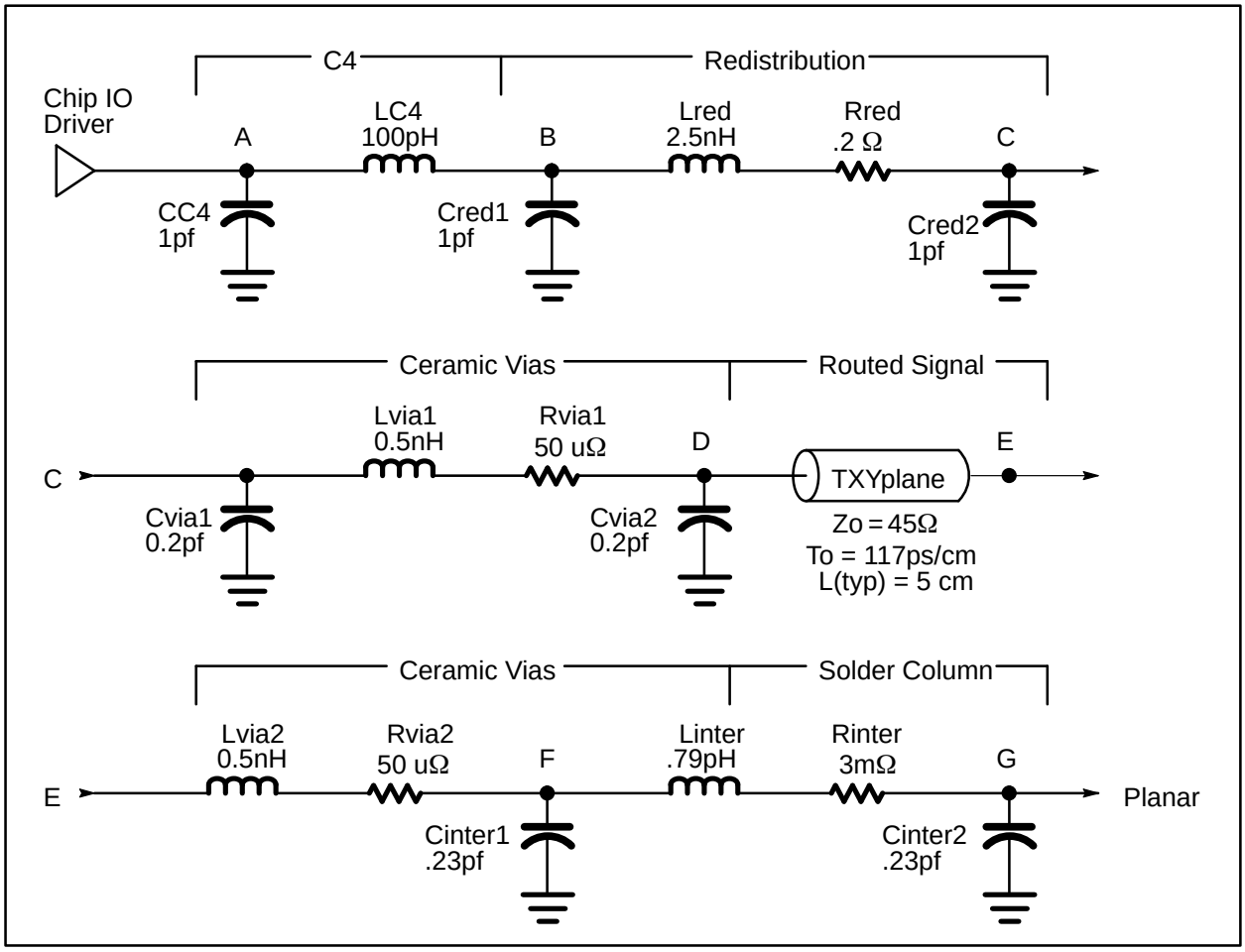


Figure 12-1. MCM Net Electrical Model

Table 12-1. MCM Primary I/O

Net Length	Type	Name	Net Topology			
63.0000	pci	PCI_AD0	J1.T01	U2.48	U3.236	
44.2500	pci	PCI_AD1	J1.R01	U2.59	U3.237	
50.6036	pci	PCI_AD2	J1.R02	U2.46	U3.238	
47.5000	pci	PCI_AD3	J1.P01	U2.43	U3.239	
42.1036	pci	PCI_AD4	J1.N02	U2.42	U3.10	
41.0000	pci	PCI_AD5	J1.M01	U2.41	U3.11	
42.3536	pci	PCI_AD6	J1.L01	U2.40	U3.12	
39.9571	pci	PCI_AD7	J1.L02	U2.39	U3.13	
40.1036	pci	PCI_AD8	J1.K01	U2.38	U3.14	
49.7500	pci	PCI_AD9	J1.J01	U2.37	U3.15	
58.8536	pci	PCI_PAR	J1.AH01	U2.7		
40.1036	pci	PCI_AD10	J1.H01	U2.36	U3.16	
46.9571	pci	PCI_AD11	J1.G01	U2.35	U3.17	
36.1036	pci	PCI_AD12	J1.G02	U2.34	U3.18	
55.0000	pci	PCI_AD13	J1.F01	U2.33	U3.19	
39.8536	pci	PCI_AD14	J1.E01	U2.32	U3.20	
46.9571	pci	PCI_AD15	J1.E02	U2.31	U3.21	
47.9571	pci	PCI_AD16	J1.D01	U2.30	U3.40	
48.6036	pci	PCI_AD17	J1.C02	U2.29	U3.41	
44.2500	pci	PCI_AD18	J1.B03	U2.28	U3.42	
47.5000	pci	PCI_AD19	J1.C04	U2.25	U3.43	
50.6036	pci	PCI_AD20	J1.A03	U2.24	U3.44	
49.4571	pci	PCI_AD21	J1.A04	U2.23	U3.45	
46.8536	pci	PCI_AD22	J1.B05	U2.22	U3.46	
51.0000	pci	PCI_AD23	J1.A05	U2.21	U3.47	
41.3536	pci	PCI_AD24	J1.A06	U2.20	U3.48	
44.2071	pci	PCI_AD25	J1.B07	U2.19	U3.49	
47.6036	pci	PCI_AD26	J1.A07	U2.18	U3.50	
47.6036	pci	PCI_AD27	J1.A08	U2.15	U3.51	
46.4571	pci	PCI_AD28	J1.B09	U2.14	U3.62	
43.7500	pci	PCI_AD29	J1.A09	U2.13	U3.63	
47.5000	pci	PCI_AD30	J1.A10	U2.12	U3.64	
42.2071	pci	PCI_AD31	J1.C10	U2.11	U3.65	
73.5000	pci	PCI_IRDY#	J1.AB01	U2.201	U3.167	
49.1036	pci	PCI_LOCK#	J1.AE02	U2.53		
72.6036	pci	PCI_PERR#	J1.U01	U2.10		
52.8536	pci	PCI_SERR#	J1.U02	U2.71		
56.1036	pci	PCI_STOP#	J1.AA01	U2.203		
64.9571	pci	PCI_TRDY#	J1.AC02	U2.202	U3.168	
44.3536	pci	PCI_AD_OE#	J1.N14	U2.195	U3.144	
46.1036	pci	PCI_C/BE0#	J1.V01	U2.6		
53.6036	pci	PCI_C/BE1#	J1.W02	U2.5		
48.2071	pci	PCI_C/BE2#	J1.W01	U2.4		
49.2500	pci	PCI_C/BE3#	J1.Y01	U2.3		
12.5000	pci	PCI_CLK_IN	J1.K29	U2.123		
54.3536	pci	PCI_DEVSEL#	J1.AA02	U2.204		

Table 12-1. MCM Primary I/O (Continued)

Net Length	Type	Name	Net Topology			
41.6036	pci	PCI_EXT_SEL	J1.U14	U2.67	U3.153	
62.2071	pci	PCI_OL_OPEN	J1.W14	U2.64	U3.165	
50.2071	pci	PCI_OUT_SEL	J1.R14	U2.68	U3.169	
52.8536	pci	664_PCI_REQ#	J1.AF01	U2.58		
55.5000	pci	PCI_FRAME_664#	J1.AC01	U2.200		
42.2500	memory	MA0	J1.AL20	U2.190		
20.2500	memory	MA1	J1.AL22	U2.189		
36.3536	memory	MA2	J1.AL24	U2.188		
36.0000	memory	MA3	J1.AL26	U2.187		
32.7500	memory	MA4	J1.AL28	U2.186		
32.3536	memory	MA5	J1.AL30	U2.185		
31.1036	memory	MA6	J1.AK31	U2.184		
28.6036	memory	MA7	J1.AH31	U2.181		
29.8536	memory	MA8	J1.AF31	U2.180		
25.6036	memory	MA9	J1.AD31	U2.179		
39.8536	memory	MD0	J1.AN03	U3.180		
36.5000	memory	MD1	J1.AM03	U3.182		
45.7500	memory	MD2	J1.AN04	U3.183		
50.0000	memory	MD3	J1.AN05	U3.184		
54.5000	memory	MD4	J1.AM05	U3.189		
53.3536	memory	MD5	J1.AN06	U3.190		
53.2500	memory	MD6	J1.AN07	U3.193		
45.1036	memory	MD7	J1.AM07	U3.194		
39.5000	memory	MD8	J1.AN09	U3.200		
38.7500	memory	MD9	J1.AM09	U3.201		
22.5000	memory	MA10	J1.AB31	U2.178		
20.2500	memory	MA11	J1.Y31	U2.177		
35.0000	memory	MDP0	J1.AN08	U3.141		
43.2500	memory	MDP1	J1.AN14	U3.122		
48.6036	memory	MDP2	J1.AN20	U3.103		
52.0000	memory	MDP3	J1.AN26	U3.82		
58.2500	memory	MDP4	J1.AL32	U3.37		
65.5000	memory	MDP5	J1.AE32	U3.234		
61.0000	memory	MDP6	J1.W32	U3.214		
53.8536	memory	MDP7	J1.N32	U3.195		
54.1036	memory	MD10	J1.AN10	U3.202		
45.3536	memory	MD11	J1.AN11	U3.203		
41.2500	memory	MD12	J1.AM11	U3.206		
42.7500	memory	MD13	J1.AN12	U3.211		
43.1036	memory	MD14	J1.AN13	U3.212		
44.3536	memory	MD15	J1.AM13	U3.213		
46.2500	memory	MD16	J1.AN15	U3.215		
65.8536	memory	MD17	J1.AM15	U3.222		
45.6036	memory	MD18	J1.AN16	U3.223		
47.2500	memory	MD19	J1.AN17	U3.224		
45.7500	memory	MD20	J1.AM17	U3.225		

Table 12-1. MCM Primary I/O (Continued)

Net Length	Type	Name	Net Topology			
51.3536	memory	MD21	J1.AN18	U3.231		
52.1036	memory	MD22	J1.AN19	U3.232		
70.2500	memory	MD23	J1.AM19	U3.233		
51.1036	memory	MD24	J1.AN21	U3.5		
48.5000	memory	MD25	J1.AM21	U3.6		
50.2500	memory	MD26	J1.AN22	U3.7		
50.5000	memory	MD27	J1.AN23	U3.28		
48.7500	memory	MD28	J1.AM23	U3.29		
52.6036	memory	MD29	J1.AN24	U3.30		
52.3536	memory	MD30	J1.AN25	U3.35		
50.0000	memory	MD31	J1.AM25	U3.36		
52.8536	memory	MD32	J1.AN27	U3.58		
50.1036	memory	MD33	J1.AM27	U3.59		
51.9571	memory	MD34	J1.AN28	U3.60		
54.2500	memory	MD35	J1.AN29	U3.73		
53.0000	memory	MD36	J1.AM29	U3.74		
56.8536	memory	MD37	J1.AN30	U3.75		
59.6036	memory	MD38	J1.AN31	U3.76		
56.2500	memory	MD39	J1.AM31	U3.81		
58.8536	memory	MD40	J1.AL33	U3.83		
57.7500	memory	MD41	J1.AK33	U3.90		
56.0000	memory	MD42	J1.AJ32	U3.91		
56.3536	memory	MD43	J1.AJ33	U3.92		
57.6036	memory	MD44	J1.AH33	U3.93		
54.2500	memory	MD45	J1.AG32	U3.100		
56.0000	memory	MD46	J1.AG33	U3.101		
54.3536	memory	MD47	J1.AF33	U3.102		
52.9571	memory	MD48	J1.AE33	U3.108		
54.5000	memory	MD49	J1.AD33	U3.109		
49.8536	memory	MD50	J1.AC32	U3.111		
51.1036	memory	MD51	J1.AC33	U3.112		
50.7500	memory	MD52	J1.AB33	U3.113		
50.2500	memory	MD53	J1.AA32	U3.118		
49.8536	memory	MD54	J1.AA33	U3.119		
54.5000	memory	MD55	J1.Y33	U3.121		
49.8536	memory	MD56	J1.W33	U3.123		
49.8536	memory	MD57	J1.V33	U3.130		
53.3536	memory	MD58	J1.U32	U3.131		
60.7500	memory	MD59	J1.U33	U3.132		
45.5000	memory	MD60	J1.T33	U3.133		
56.8536	memory	MD61	J1.R32	U3.138		
48.6036	memory	MD62	J1.R33	U3.139		
53.2500	memory	MD63	J1.P33	U3.140		
10.2500	memory	MCE0#	J1.L32	U2.174		
10.5000	memory	MCE1#	J1.L33	U2.173		
13.3536	memory	MCE2#	J1.K33	U2.172		

Table 12-1. MCM Primary I/O (Continued)

Net Length	Type	Name	Net Topology			
6.6036	memory	MCE3#	J1.J32	U2.171		
9.5000	memory	MCE4#	J1.J33	U2.170		
7.5000	memory	MCE5#	J1.H33	U2.169		
4.7500	memory	MCE6#	J1.G32	U2.168		
7.5000	memory	MCE7#	J1.G33	U2.165		
64.7500	memory	MRE0#	J1.AL04	U2.164		
67.3536	memory	MRE1#	J1.AL06	U2.163		
65.1036	memory	MRE2#	J1.AL08	U2.162		
60.6036	memory	MRE3#	J1.AL10	U2.161		
57.0000	memory	MRE4#	J1.AL12	U2.160		
54.2500	memory	MRE5#	J1.AL14	U2.159		
52.1036	memory	MRE6#	J1.AL16	U2.158		
53.8536	memory	MRE7#	J1.AL18	U2.157		
13.8536	memory	MWE0#	J1.M33	U2.176		
12.6036	memory	MWE1#	J1.N33	U2.175		
38.7950	funct.	TEA#	J1.N10	R1.D01	U1.154	U2.137
46.2857	funct.	DRTRY#	J1.AJ01	R1.H01	U1.156	
19.2500	funct.	ROM_OE#	J1.V31	U2.47		
17.5000	funct.	ROM_WE#	J1.T31	U2.60		
39.4691	funct.	INT_60X#	J1.U16	R1.D03	U1.188	
40.7150	funct.	MCP_60X#	J1.R16	R1.H02	U1.186	
81.0192	funct.	SRAM_OE#	J1.AA22	U7.50	U8.50	U9.50, U10.50
10.2020	funct.	PLL_CFG0	J1.D25	U1.213		
16.6744	funct.	PLL_CFG1	J1.B27	U1.211		
17.1617	funct.	PLL_CFG2	J1.A27	U1.210		
15.3550	funct.	PLL_CFG3	J1.C28	U1.208		
53.8536	funct.	ROM_LOAD	J1.U20	U2.70	U3.160	
32.4704	funct.	60X_AVDD	J1.B02	U1.209		
27.2459	funct.	QACK_60X#	J1.D17	R1.E03	U1.235	
4.2367	funct.	QREQ_60X#	J1.C18	U1.31		
4.2367	funct.	X_INT_60X#	J1.F31	U2.139		
4.2367	funct.	X_MCP_60X#	J1.H27	U2.138		
4.2367	funct.	X_SRAM_OE#	J1.G26	U2.117		
19.0000	funct.	INT_TO_664	J1.B31	U2.55		
8.5866	funct.	X_PCLK_60X	J1.D19	U1.212		
5.6036	funct.	X_TAG_BCLK	J1.P29	U5.69		
21.9935	funct.	SRESET_60X#	J1.D31	U1.189		
7.3536	funct.	TAG_ADDR_13	J1.L26	U5.33		
6.0000	funct.	TAG_DATA_11	J1.M27	U5.65		
52.1036	funct.	IGN_PCI_AD31	J1.AD01	U2.57		
4.6624	funct.	X_SRAM_BCLK0	J1.Y29	U7.51		
5.1624	funct.	X_SRAM_BCLK1	J1.AF29	U8.51		
9.1624	funct.	X_SRAM_BCLK2	J1.AF05	U9.51		
8.6624	funct.	X_SRAM_BCLK3	J1.Y05	U10.51		
1.7500	funct.	X_663_CPU_CLK	J1.F05	U3.157		

Table 12-1. MCM Primary I/O (Continued)

Net Length	Type	Name	Net Topology			
10.1036	funct.	X_664_CPU_CLK	J1.H29	U2.121		
8.6036	funct.	X_CPU_RDL_ OPEN	J1.G10	U3.148		
22.2500	funct.	NMI_FROM_ ISABRDG	J1.A31	U2.56		
7.2071	funct.	POWER_ GOOD/RESET#	J1.A28	U2.156		
7.2071	funct rw	HRESET#	J1.E32	R1.D04	U1.214	
35.6584	rw	TCK	J1.B25	R1.J04	U1.201	
41.2051	rw	TDI	J1.A25	R1.J05	U1.199	
14.6846	rw	TDO	J1.C26	U1.198		
34.9543	rw	TMS#	J1.A24	R1.J03	U1.200	
58.3719	rw	CKSTP_OUT#	J1.AJ02	R1.J01	U1.216	
58.3719	rw p_up	TRST#	J1.C24	R1.H04	U1.202	
29.4987	p_up	TT0	J1.A19	U1.191	U2.150	
27.6398	p_up	TT1	J1.B19	U1.190	U2.152	
15.3133	p_up	TT2	J1.A20	U1.185		
28.0788	p_up	TT3	J1.A21	U1.184	U2.126	
5.3536	p_up	SHD#	J1.D27	U2.141		
51.7310	p_up	SMI#	J1.AL02	R1.F05	U1.187	
11.6036	p_up	TAG_CS2	J1.H31	U5.76		
21.5000	p_up	TT2_664	J1.C14	U2.153		
35.9554	p_up	DBG_60X#	J1.G08	U1.26	U2.140	
70.1036	p_up	L2_CLAIM#	J1.AL01	U2.132		
21.8536	p_up	663_TEST#	J1.G14	U3.155		
49.6036	p_up	664_TEST#	J1.J04	U2.155		
30.8536	p_up	TAG_MATCH	J1.K31	U2.142	U5.50	
22.0000	p_up	BG_MASTER#	J1.N20	U2.135		
56.1497	p_up	SRAM_ADSP#	J1.AK03	U7.1	U8.1	U9.1, U10.1
5.0000	p_up	TAG_PWRDN#	J1.M31	U5.77		
48.1036	p_up	MWS_P2MRXS	J1.C30	U2.66	U3.152	
40.9072	p_up	664_STOP_ CLK_EN	J1.B29	R1.F04	U2.151	
30.7884	p_down	TT4	J1.A23	U1.180	U2.136	
24.5780	p_down	GBL#	J1.A12	U1.1	U2.120	
43.8399	p_down	SRAM_CS#	J1.AD29	U7.5	U8.5	U9.5, U10.5
19.5000	p_down	TAG_CS1#	J1.C33	U5.75		
72.1036	p_down	OE_245_B	J1.D29	U6A.25A	U6A.48A	U6B.25B, U6B.48B
32.2071	p_down	DIR_245_A	J1.G22	U6A.1A	U6A.24A	
43.6036	p_down	DIR_245_B	J1.L22	U6B.1B	U6B.24B	
20.7500	p_down	TAG_SFUNC	J1.C32	U5.22		
52.6036	p_down	CRS_C2PWXS	J1.A29	U2.65	U3.151	
39.1036	p_down	663_MIO_TEST	J1.A30	U3.156		
49.2500	p_down	664_MIO_TEST	J1.K05	U2.154		
13.7500	clk	XTAL1	J1.U30	U4.12		

Table 12-1. MCM Primary I/O (Continued)

Net Length	Type	Name	Net Topology			
11.7071	clk	XTAL2	J1.T29	U4.13		
32.5000	clk	FRZ_CLK	J1.B11	U4.3		
18.6036	clk	FRZ_DATA	J1.A22	U4.5		
8.5866	clk	PCLK_60X	J1.E18	U4.34		
66.3536	clk	TAG_BCLK	J1.N30	U4.36		
29.9142	clk	CLK_EXT_FB	J1.A17	U4.14		
27.0000	clk	CLK_FB_SEL	J1.A15	U4.9		
27.5000	clk	CLK_PLL_EN	J1.D33	U4.7		
67.5000	clk	SRAM_BCLK0	J1.W30	U4.44		
66.5000	clk	SRAM_BCLK1	J1.AE30	U4.46		
9.1624	clk	SRAM_BCLK2	J1.AG04	U4.48		
8.6624	clk	SRAM_BCLK3	J1.AA04	U4.50		
22.6036	clk	CLK_COM_FRZ	J1.A18	U4.6		
29.6036	clk	CLK_REF_SEL	J1.B13	U4.8		
24.4571	clk	CLK_TTL_CLK	J1.A26	U4.11		
28.0000	clk	CLK_VCO_SEL	J1.A13	U4.52		
1.7500	clk	663_CPU_CLK	J1.G04	U4.38		
63.6036	clk	664_CPU_CLK	J1.G30	U4.42		
24.1036	clk	CLK_BCLK_DIV0	J1.C16	U4.31		
22.2500	clk	CLK_BCLK_DIV1	J1.B17	U4.27		
26.3536	clk	CLK_FRZ_ STROBE	J1.F33	U4.4		
14.8536	clk	CLK_MPC601_ CLKS	J1.K27	U4.40		
25.3536	clk	CLK_ MR/TRISTATE	J1.A16	U4.2		
61.8536	clk	664_PCI_CLK	J1.J30	U4.18		
20.6036	clk	CLK_PCI_DIV0	J1.B21	U4.20		
20.0607	clk	CLK_PCI_DIV1	J1.C20	U4.26		
38.1036	clk	USER_PCICLK1	J1.AE01	U4.16		
29.5000	clk	USER_PCICLK2	J1.N01	U4.21		
42.8536	clk	USER_PCICLK3	J1.C01	U4.23		
31.1036	clk	USER_PCICLK4	J1.A11	U4.25		
25.4571	clk	USER_PCICLK5	J1.A14	U4.29		
29.7500	clk	USER_PCICLK6	J1.E33	U4.32		
3.0000	daisy	DAISY01	J1.D05			
3.0000	daisy	DAISY01	J1.E04			
2.5000	daisy	DAISY02	J1.D07			
2.5000	daisy	DAISY02	J1.E06			
2.5000	daisy	DAISY03	J1.D09			
2.5000	daisy	DAISY03	J1.E08			
2.5000	daisy	DAISY04	J1.D11			
2.5000	daisy	DAISY04	J1.E10			
2.5000	daisy	DAISY05	J1.D13			
2.5000	daisy	DAISY05	J1.E12			
2.5000	daisy	DAISY06	J1.E30			
2.5000	daisy	DAISY06	J1.F29			

Table 12-1. MCM Primary I/O (Continued)

Net Length	Type	Name	Net Topology			
3.0000	daisy	DAISY07	J1.E28			
3.0000	daisy	DAISY07	J1.F27			
2.5000	daisy	DAISY08	J1.E26			
2.5000	daisy	DAISY08	J1.F25			
3.2500	daisy	DAISY09	J1.E24			
3.2500	daisy	DAISY09	J1.F23			
4.3536	daisy	DAISY10	J1.E22			
4.3536	daisy	DAISY10	J1.F21			
3.0000	daisy	DAISY11	J1.AK05			
3.0000	daisy	DAISY11	J1.AK07			
3.0000	daisy	DAISY12	J1.AK09			
3.0000	daisy	DAISY12	J1.AK11			
3.2500	daisy	DAISY13	J1.AK13			
3.2500	daisy	DAISY13	J1.AK15			
4.0000	daisy	DAISY14	J1.AK17			
4.0000	daisy	DAISY14	J1.AK19			
3.0000	daisy	DAISY15	J1.AK21			
3.0000	daisy	DAISY15	J1.AK23			
3.0000	daisy	DAISY16	J1.AD03			
3.0000	daisy	DAISY16	J1.AF03			
3.0000	daisy	DAISY17	J1.Y03			
3.0000	daisy	DAISY17	J1.AB03			
3.0000	daisy	DAISY18	J1.T03			
3.0000	daisy	DAISY18	J1.V03			
3.0000	daisy	DAISY19	J1.M03			
3.0000	daisy	DAISY19	J1.P03			
3.0000	daisy	DAISY20	J1.H03			
3.0000	daisy	DAISY20	J1.K03			
3.0000	daisy	DAISY20	J1.K03			

12.2 MCM Thermal

Figure 12-2 contains a simplified drawing of the construction of the MCM. The individual devices are mounted to the ceramic substrate, which is mounted to the circuit board by solder columns. The cap covers the devices and the top of the substrate, and is filled with a thermal grease. The cap is non-hermetically sealed to the substrate.

As shown in Figure 12-3, the major heat flow path is from the devices to the thermal grease, to the aluminum cap, and to ambient. A heat sink can be attached to the cap if required.

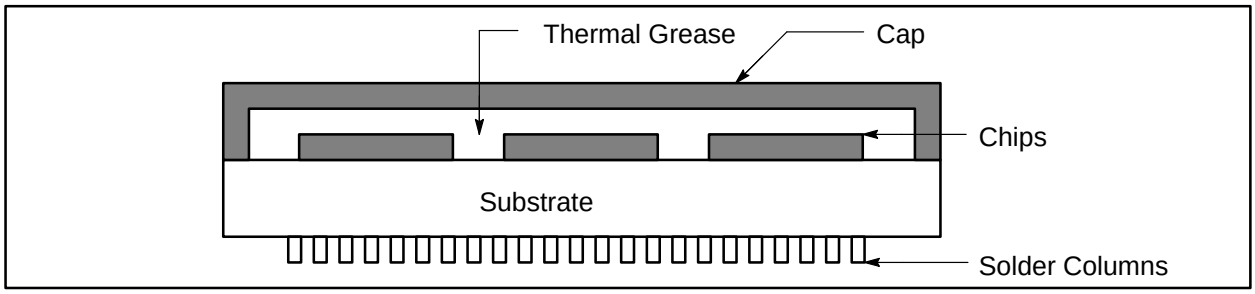


Figure 12-2. MCM Thermal Paths

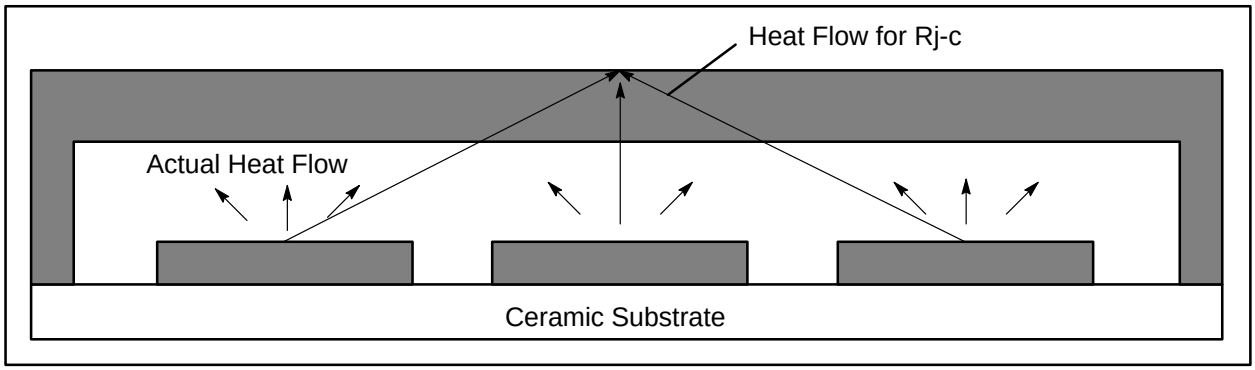


Figure 12-3. MCM Heat Flows

12.2.1 Chip Thermal Requirements

Table 12-2 shows many of the thermal specifications of the chips in the MCM. The values given for non-IBM products are superceded by any values found in the manufacturers' data sheets.

At a typical power dissipation, the difference between the temperature of the center of the chip and the temperature of the center of the MCM cap was measured (see Figure 12-3). The thermal resistance values from chip junction to MCM cap (Rj-c) shown in Table 12-2 were derived from these measurements:

$$R(j\text{-}c) = [T(chip) - T(cap)] / Power(chip),$$

and do not change significantly over the operating range of the MCM. Rj-c describes the thermal resistance along the path from the center of the device to the center of the MCM cap. This specification can be used with the MCM to simplify thermal calculations. Table 12-3 shows the same data as Table 12-2, but for the expected maximum power dissipation of the MCM. All references to the MCM cap in Table 12-2 and Table 12-3 refer to the center of the MCM cap.

Table 12-2. Thermal Specifications for MCM Chips, Typical Pd

Parameter	603e	663	664	'245	MPC 970	SRAM	Tag RAM
Maximum Junction Temperature (°C)	105	85	85	85	70	70	70
Thermal Resistance, Junction to MCM Cap (° C/W)	1.22	.1	.88	11.6	4.24	.55	.53
Power Dissipation, Typical (W)	3.2	.5	1	.2	.7	.7	.5
Junction Temp Rise Above Cap (°C)	3.9	.05	.88	2.3	3.0	.39	.27
Maximum Allowed Cap Temperature at Typical Pd (°C)	101	85	84.1	82.7	67	69.6	69.7
Typical MCM Total Pd = 9 W							

Table 12-3. Thermal Specifications for MCM Chips, Maximum Pd

Parameter	603e	663	664	'245	MPC 970	SRAM	Tag RAM
Maximum Junction Temperature (°C)	105	85	85	85	70	70	70
Thermal Resistance, Junction to MCM Cap (° C/W)	1.22	.1	.88	11.6	4.24	.55	.53
Power Dissipation, Maximum (W)	4	.6	1.3	.2	.83	1	1
Junction Temp Rise Above Cap (°C)	4.9	.06	1.1	2.3	3.5	.55	.53
Maximum Allowed Cap Temperature at Max Pd (°C)	100	85	83.9	82.7	66.5	69.5	69.5
Maximum MCM Total Pd = 12 W							

12.2.2 MCM Cooling Requirements

Table 12-2 and Table 12-3 show the typical and maximum total power dissipation of the MCM. Additionally, the tables show that the MCM cap must be maintained below 67°C at typical Pd and 66°C at maximum Pd to ensure that all of the chips are adequately cooled.

There are other factors, such as heat spreading and alternate heat conduction pathways, that tend to reduce the cooling requirements. Treatment of these complex MCM topics is beyond the scope of this document.

The total power entering the MCM cap from the chips is the sum of the power dissipation of each individual chip. This power must be removed from the MCM while ensuring that the temperature of the center of the MCM cap does not exceed the value derived in Section 12.2.1 for various operating conditions. Table 12-4 shows the required total thermal resistance from the cap of the MCM to ambient at various ambient temperatures.

Table 12-4. Required Maximum Thermal Resistance, Cap to Ambient

Ambient Temperature	At Typical Pd	At Maximum Pd
65 °C	0.10	0.08
60 °C	0.67	0.50
55 °C	1.20	0.92
50 °C	1.80	1.33
45 °C	2.33	1.75
40 °C	2.88	2.17
35 °C	3.44	2.60
30 °C	4.00	3.00
25 °C	4.55	3.40

12.2.3 Cooling Recommendations

Table 12-5 shows the thermal resistance from the MCM cap to ambient at various airflows.

Table 12-5. Thermal Resistance From Cap to Ambient – No Heat Sink

Air Flow (Linear Feet Per Minute)	Θc–a (Thermal Resistance, Cap to Ambient)
50	15.45 ° C/W
100	12.7 ° C/W
200	9.7 ° C/W
300	8.5 ° C/W
400	7.7 ° C/W
500	7.4 ° C/W
600	7.1 ° C/W
700	6.9 ° C/W

Comparison of Table 12-4 with Table 12-5 shows that a heat sink will be required to meet the MCM maximum cooling requirement.

12.2.3.1 Thermal Resistance From Cap to Heat Sink

Table 12-6 shows the thermal resistance from the MCM cap to a heat sink for two different cases. Each case assumes a 45mm x 45mm heat sink (the same size as the MCM cap), with 100% coverage of a thermally conductive adhesive between the cap and the heatsink.

Table 12-6. Thermal Resistance From Cap to Heat Sink

Parameter	Chromerics T405 Reworkable Adhesive	Al Epoxy 7655 Non-Reworkable
Thermal Conductivity (K, Watts per meter–deg Kelvin)	.4 W/mk	1.06 W/mK
Thickness	0.14 mm	0.1 mm
Θc–hs (Thermal Resistance, Cap to Heat Sink)	0.18 ° C/W	0.05 ° C/W

12.2.3.2 Thermal Resistance From Heat Sink to Ambient

Table 12-7 shows the required heat sink performance at various ambient temperatures, using a value of .1°C/W for the thermal resistance from the MCM cap to the heat sink, 100% coverage, and 45mm x 45mm heat sink base.

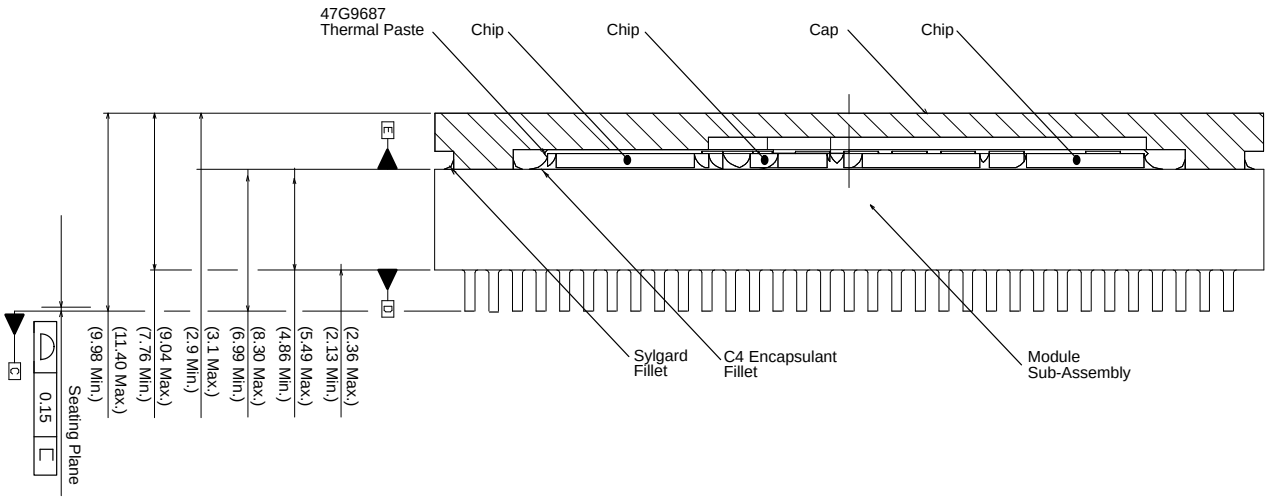
Table 12-7. Required Maximum Thermal Resistance, Cap to Ambient

Ambient Temperature	At Typical Pd	At Maximum Pd
65 °C	n/a	n/a
60 °C	0.57	0.40
55 °C	1.10	0.82
50 °C	1.70	1.23
45 °C	2.23	1.65
40 °C	2.78	2.07
35 °C	3.34	2.50
30 °C	3.90	2.90
25 °C	4.45	3.30

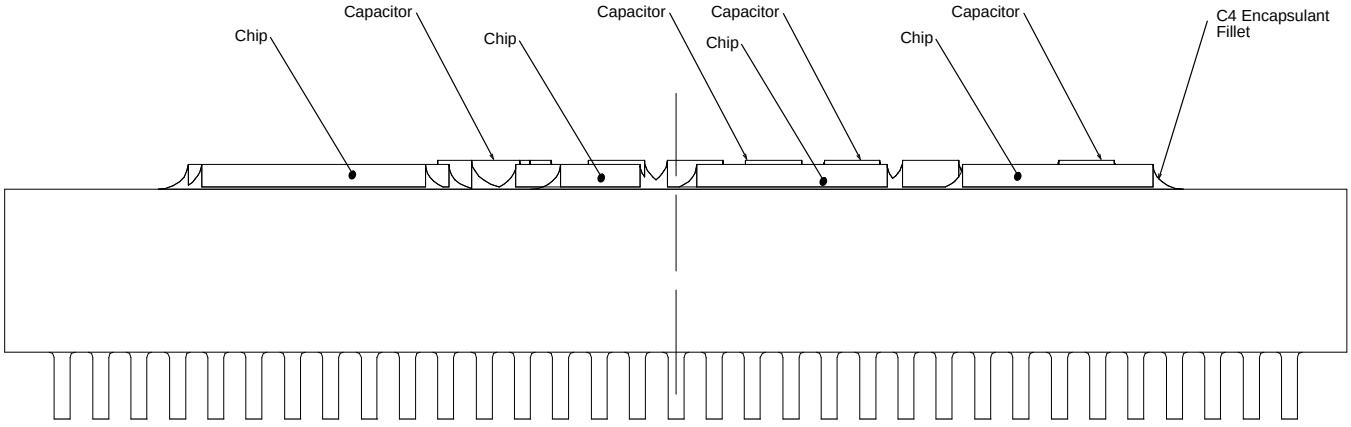
High performance passive heatsinks with good forced air flow and typical fansinks are able to perform to these specifications.

12.3 MCM Mechanical Drawings

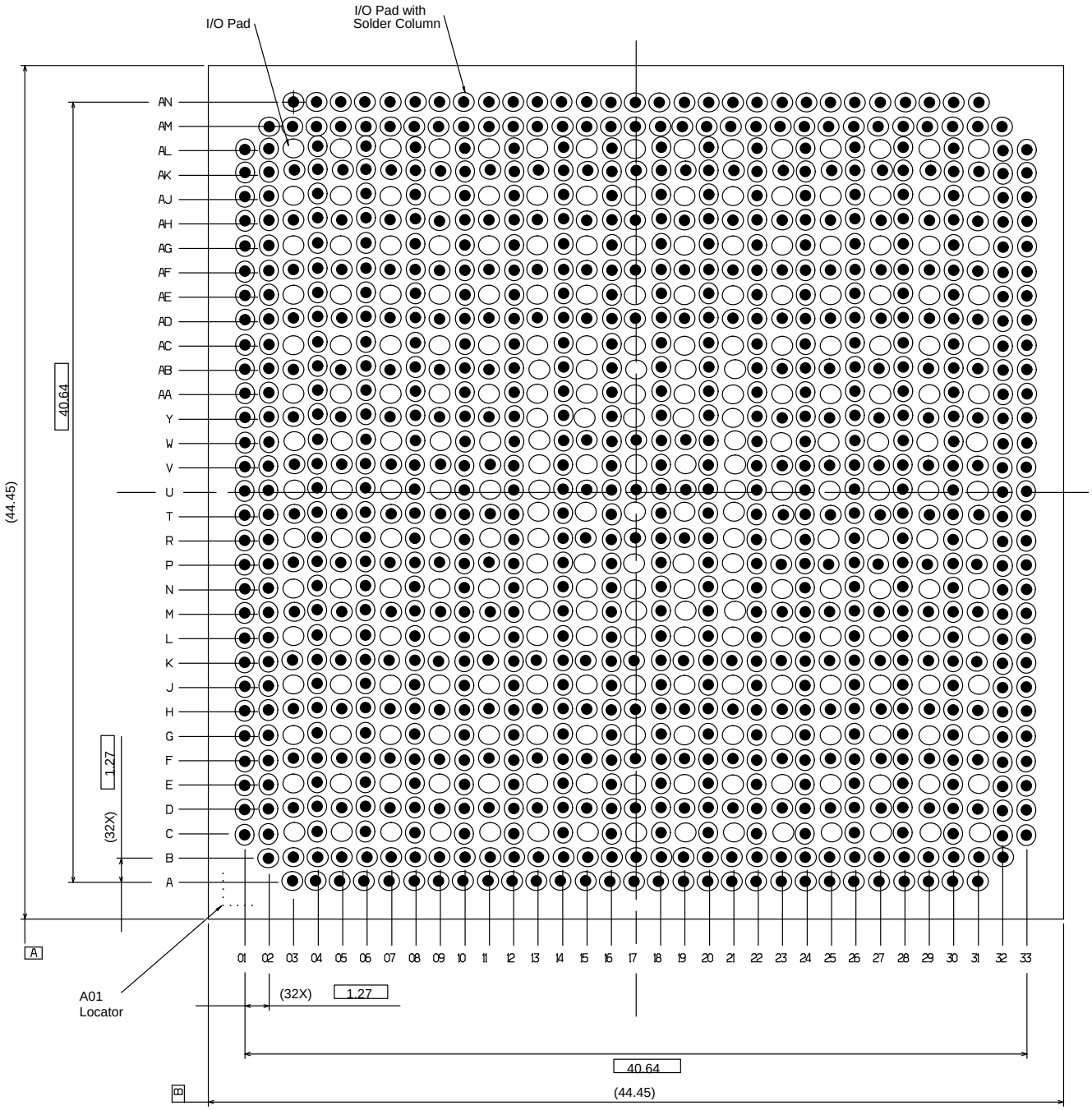
12.3.1 MCM with Cap



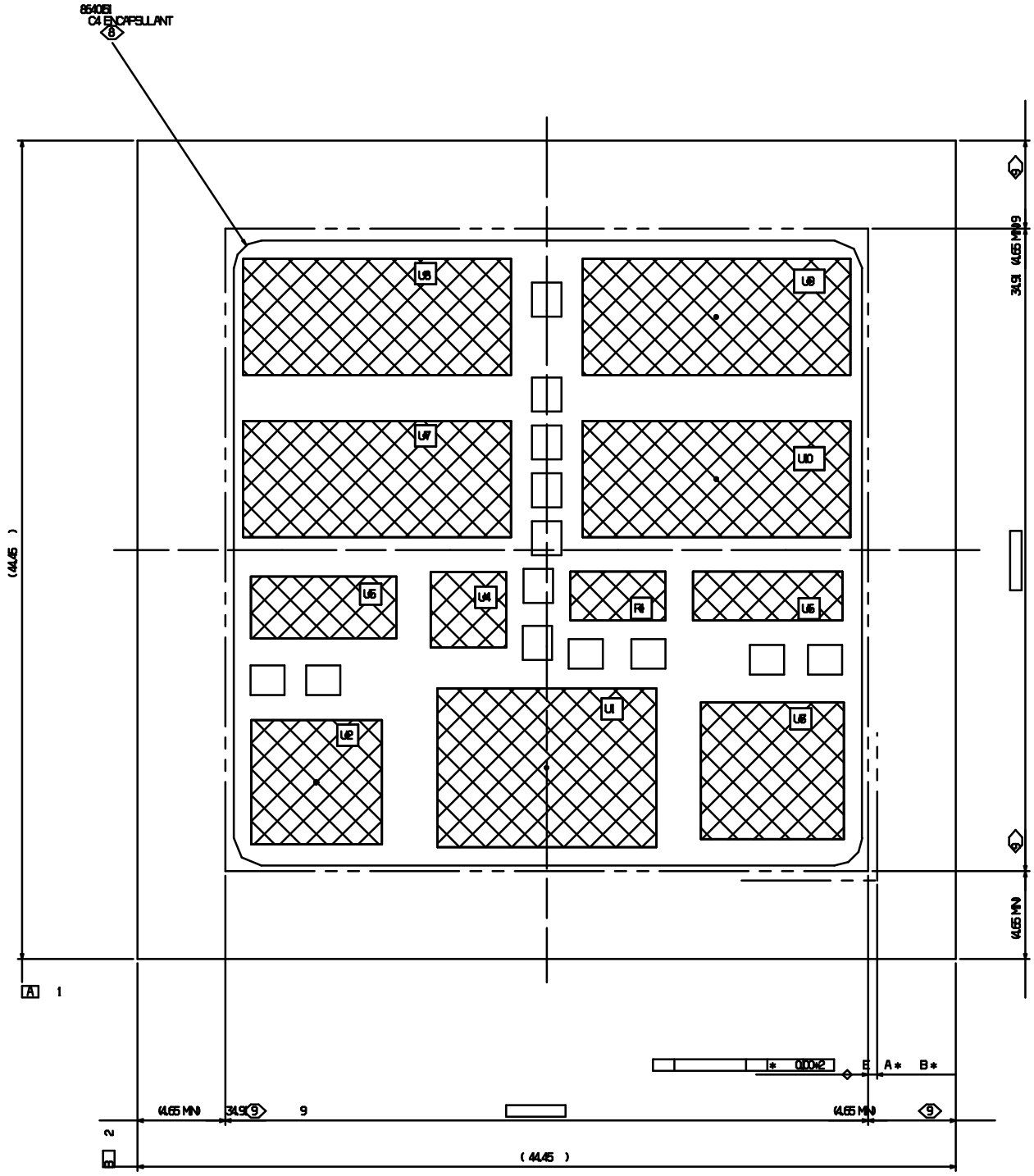
12.3.2 MCM Without Cap



12.3.3 MCM Solder Columns



12.3.4 MCM Component Placement



Part Number 91H5797
Sheet 1 of 5

Section 13
MCM Schematics

This section contains the schematics of the Odyssey MCM. For schematics of the planar, see Appendix F.

The schematics are numbered separately from the rest of the MCM technical specification.

13.1 Component Placement

MCM

MCM TEST AND STRESS ENGINEERING

POUGHKEEPSIE, NY

ODYSSEY

MULTI-CHIP MODULE (MCM)

VERSION 5.1

MAY 2, 1996

CONTACT:
JED EASTMAN
TOM BARDSLEY

PPC603
512K SYNCH CACHE
663, 664 PCI, MEMORY CNTL

IBM

SIZE	REVISION	DRWNG PART NUMBER
B		

DRAWING J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 13:41:27 1996

CONTENTS

DESIGN DESCRIPTION:

THE ODYSSEY MCM IS A POWERPC 603E-BASED MULTI-CHIP MODULE WITH 256K OF L2 CACHE. THE LANAI AND KAUAI CHIPS (660 CHIPSET) PROVIDE THE MEMORY CONTROL AND PCI BRIDGE FUNCTION. 256K OF L2 CACHE IS PROVIDED BY 4 64KX18 SRAM DIE. A DISCRETE TAG RAM IS USED IN THE DESIGN. A MOTOROLA MPC970 IS ALSO JOINED TO THE MCM. A CUSTOM RESISTOR CHIP IS USED ALSO.

ODYSSEY IS BASED ON 660 CHIPSET REFERENCE PLATFORM

ALL CHIPS OPERATE AT 3.3 V EXCEPT TAG RAM, WHICH OPERATES AT 5.0 V (VDD3).

CHANGE HISTORY:

07/20/95 DRAFT VERSION 1.0 DISTRIBUTED FOR REVIEW

08/03/95 VERSION 1.1 RELEASED
- PINS ASSIGNED
- RESISTOR, CAPACITORS FIXED

08/08/95 VERSION 2.0 RELEASED
- BROUGHT 664 PCI CLOCK OFF AND BACK ON
- BROKE OUT PAL I/O FOR CONTINGENCY
- CHANGED QACK_60X AND QREG_60X TO USER I/O
- MAJOR POWER PIN ASSIGNMENT CHANGES
- ADDED PCI_PAR AND TAG_DTY MODULE I/O
- SRAM_ADSP* USER INTERFACE FROM TEST INTF

08/22/95 VERSION 2.1 RELEASED
- REASSIGNMENT OF DECOUPLING CAPACITORS, VDD1=7, VDD2=2, VDD3=1
- REASSIGN 60X_AVDD FROM H17 TO C14
- REASSIGNMENT OF PINS FOR FOLLOWING NETS:
  - TT4: J08 - E20 CI60X*: R10 - C08
  - TT3: H07 - E18 WT60X*: P09 - D15
  - TT2: G08 - E14 TEA*: K09 - A12
  - TT1: F07 - E15 CSE0_60X: H15
  - TT0: F09 - F15
- CHANGED 603E MODEL: PIN 140 TO DBG* FROM RESRVD
- ADDITION OF CAPACITORS C11 THRU C15

08/23/95 VERSION 2.2 RELEASED
- BROUGHT 60X BUS CLOCKS OFF AND BACK ONTO MODULE FOR TUNING
- REMOVED CAPACITORS C14 AND C15 - 13 TOTAL, NOT 15

08/29/95 VERSION 2.3 RELEASED
- NUMEROUS PIN SWAPS TO EASE BOARD WIRING
- NAMED DAISY CHAIN NETS
- ABUF12 SPARE PIN AC04, CHANGED TAG_OE NET
- MODIFIED LANAI MODEL, PINS 111 AND 231 NOW GND AND VDD
- REASSIGNMENT OF NUMEROUS RESISTORS
- CHANGED SEVERAL PULLUPS TO 5.0 V FROM 3.3 V
- TIED TAG 'A' (U6A) INPUTS TO MODULE PIN
- MCM PIN M22 SWITCHED FROM VDD1 TO VDD3

09/18/95 VERSION 3.0 RELEASED
- CHANGED TO MPC970 FROM MPC972 - MAJOR CHANGES TO MODULE I/O
- BROUGHT TAG<11> AND A13 OF TAG OFF MODULE FOR 256K OPTION
- CHANGED K SECTION OF RESISTOR CHIP TO PULL-UP
- BROKE CPU_RDL_OPEN NET FOR 100 OHM SERIES RESISTOR FOR LANAI 2.0

10/12/95 VERSION 3.1 RELEASED
- WIRING OF 16245 CHANGED
- SRAM CLOCKS SWAPPED (X_SRAM_BCLKX)

10/26/95 VERSION 3.2 RELEASED
- VERSION 2.0 663 PINOUT UPDATED
- ECS BASED ON EMULATOR BOARD DEBUG

11/3/95 VERSION 3.3 RELEASED
- 664_TEST* AND 663_TEST* PULL UP RESISTORS MOVED TO TAG RAM
- 663 PARITY BUSSES CORRECTED

12/1/95 VERSION 4.0 RELEASED
- MAJOR CHANGES TO PINOUT OF MODULE TO EASE BOARD WIRING

1/15/96 VERSION 5.0 RELEASED
- PINOUT MODIFIED - 3 NETS AFFECTED. MPC_60X*, INT_60X*, AND SRAM_OE*

INITIAL PARTS BUILT ACCORDING TO VERSION 5.0

5/2/96 VERSION 5.1 RELEASED
- WIRING OF CAPS C11, C12, AND C13
- ALL CHIPS WIRED ACCORDING TO C4 NOMENCLATURE, NOT PACKAGED PINS
- ADDRESS BUS CONNECTED TO 16245 INPUTS AS PULLUP FOR RELIABILITY
- ABILITY TO USE EITHER REDISTRIBUTED OR NON-REDISTRIBUTED KODIAK

TOTAL PINS: 1077 (PADS)

SIGNAL:	446	VDD1:	64
DAISY CHAIN:	40	VDD2:	64
NC:	326	VDD3:	9
		VSS:	128

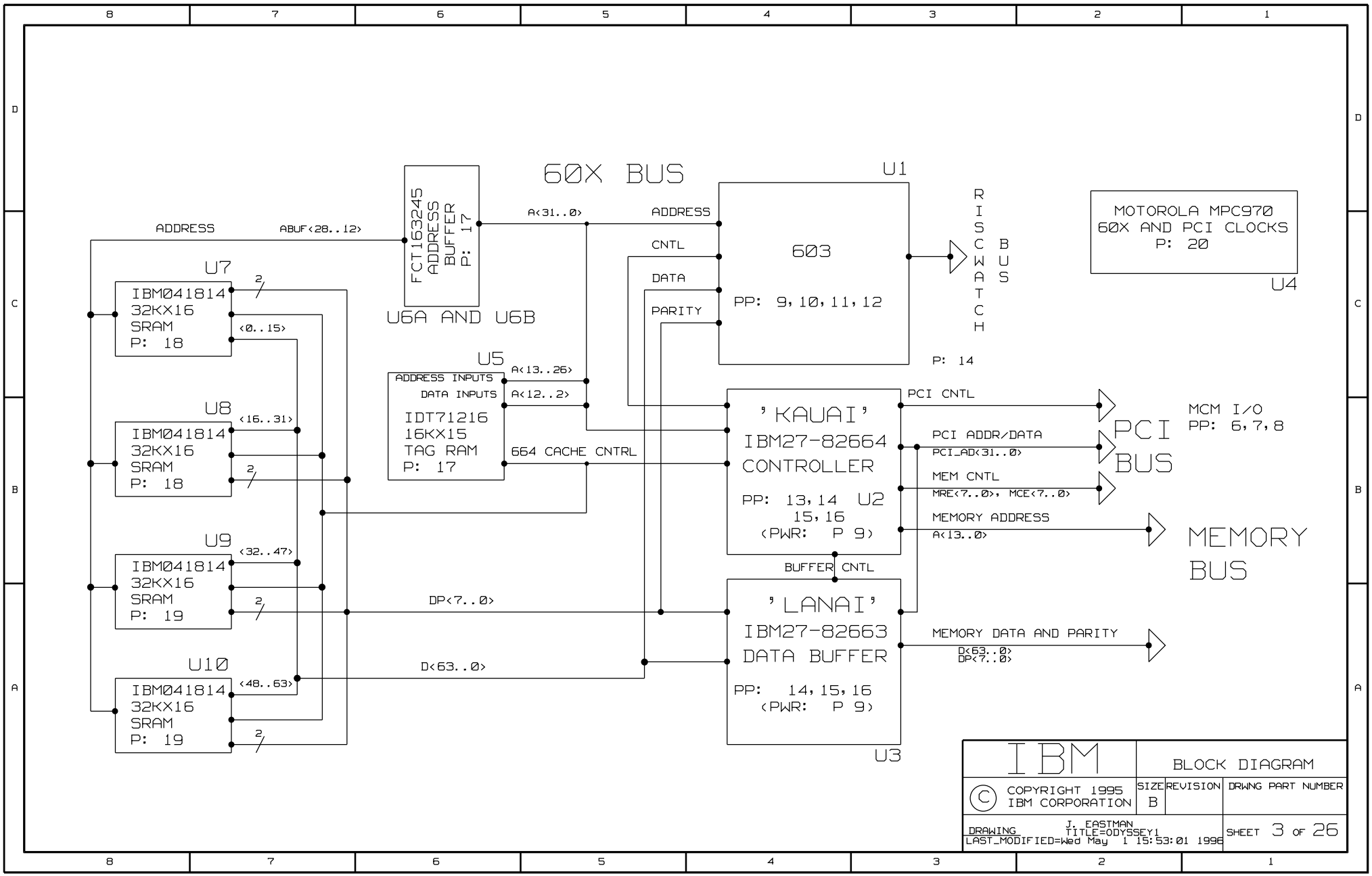
UNUSED PINS CAN BE DEPOPULATED OF COLUMNS.

IBM

CONTENTS AND DESCRIPTION

SIZE	REVISION	DRWG PART NUMBER
B		

DRAWING J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Wed May 1 15:52:40 1996


60X BUS
U1
603
PP: 9, 10, 11, 12
P: 14
RISC WATCH BUS
MOTOROLA MPC970
60X AND PCI CLOCKS
P: 20
U4
FCT163245
ADDRESS
BUFFER
P: 17
U6A AND U6B
ADDRESS
ABUF<28..12>
A<31..0>
CNTL
DATA
PARITY
U7
IBM041814
32KX16
SRAM
P: 18
<0..15>
U8
<16..31>
U9
P: 19
<32..47>
U10
<48..63>
U5
ADDRESS INPUTS
DATA INPUTS
A<13..26>
A<12..2>
IDT71216
16KX15
TAG RAM
P: 17
664 CACHE CNTRL
'KAUAI'
IBM27-82664
CONTROLLER
PP: 13, 14 U2
15, 16
<PWR: P 9>
PCI CNTL
PCI ADDR/DATA
PCI_AD<31..0>
MEM CNTL
MRE<7..0>, MCE<7..0>
MEMORY ADDRESS
A<13..0>
PCI BUS
MCM I/O
PP: 6, 7, 8
MEMORY BUS
BUFFER CNTL
DP<7..0>
D<63..0>
'LANAI'
IBM27-82663
DATA BUFFER
PP: 14, 15, 16
<PWR: P 9>
MEMORY DATA AND PARITY
D<63..0>
DP<7..0>
U3
IBM
BLOCK DIAGRAM
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Wed May 1 15:53:01 1996
SHEET 3 OF 26

PHYSICAL DESIGN NOTES:

THE FOLLOWING NETS MUST BE SHIELDED AND KEPT AS SHORT AS POSSIBLE:

XTAL1	U4.AB12	J1.U30
XTAL2	U4.AD12	J1.T29
60X_AVDD	J1.B02	U1.209
USER_PCICLK1	J1.AE01	U4.16
USER_PCICLK2	J1.N01	U4.21
USER_PCICLK3	J1.C01	U4.23
USER_PCICLK4	J1.A11	U4.25
USER_PCICLK5	J1.A14	U4.29
USER_PCICLK6	J1.E33	U4.32
60X_CLK_OUT	J1.J18	U1.221
CLK_EXT_FB	J1.A17	U4.14
CLK_TTL_CLK	J1.A26	U4.11

EACH OF THE FOLLOWING PAIRS OF NETS MUST ADD UP TO 3.0 +- 0.25 INCHES IN LENGTH AND BE SHIELDED FROM OTHER NETS:

663_CPU_CLK	J1.G04	U4.38
X_663_CPU_CLK	J1.F05	U3.157
X_664_CPU_CLK	J1.H29	U2.121
664_CPU_CLK	J1.G30	U4.42
664_PCI_CLK	J1.J30	U4.18
PCI_CLK_IN	J1.K29	U2.123
PCLK_60X	J1.E18	U4.34
X_PCLK_60X	J1.D19	U1.212
SRAM_BCLK0	J1.W30	U4.44
X_SRAM_BCLK0	J1.Y29	U7.51
SRAM_BCLK1	J1.AE30	U4.46
X_SRAM_BCLK1	J1.AF29	U8.51
SRAM_BCLK2	J1.AG04	U4.48
X_SRAM_BCLK2	J1.AF05	U9.51
SRAM_BCLK3	J1.AA04	U4.50
X_SRAM_BCLK3	J1.Y05	U10.51
TAG_BCLK	J1.N30	U4.36
X_TAG_BCLK	J1.P29	U5.69

STAR NETS ARE NOT ALLOWED.

NETS MUST BE WIRED ACCORDING TO SCHEDULE.

IBM

PHYSICAL DESIGN INFORMATION

SIZE B | REVISION | DRWG PART NUMBER

DRAWING J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 13:50:17 1996

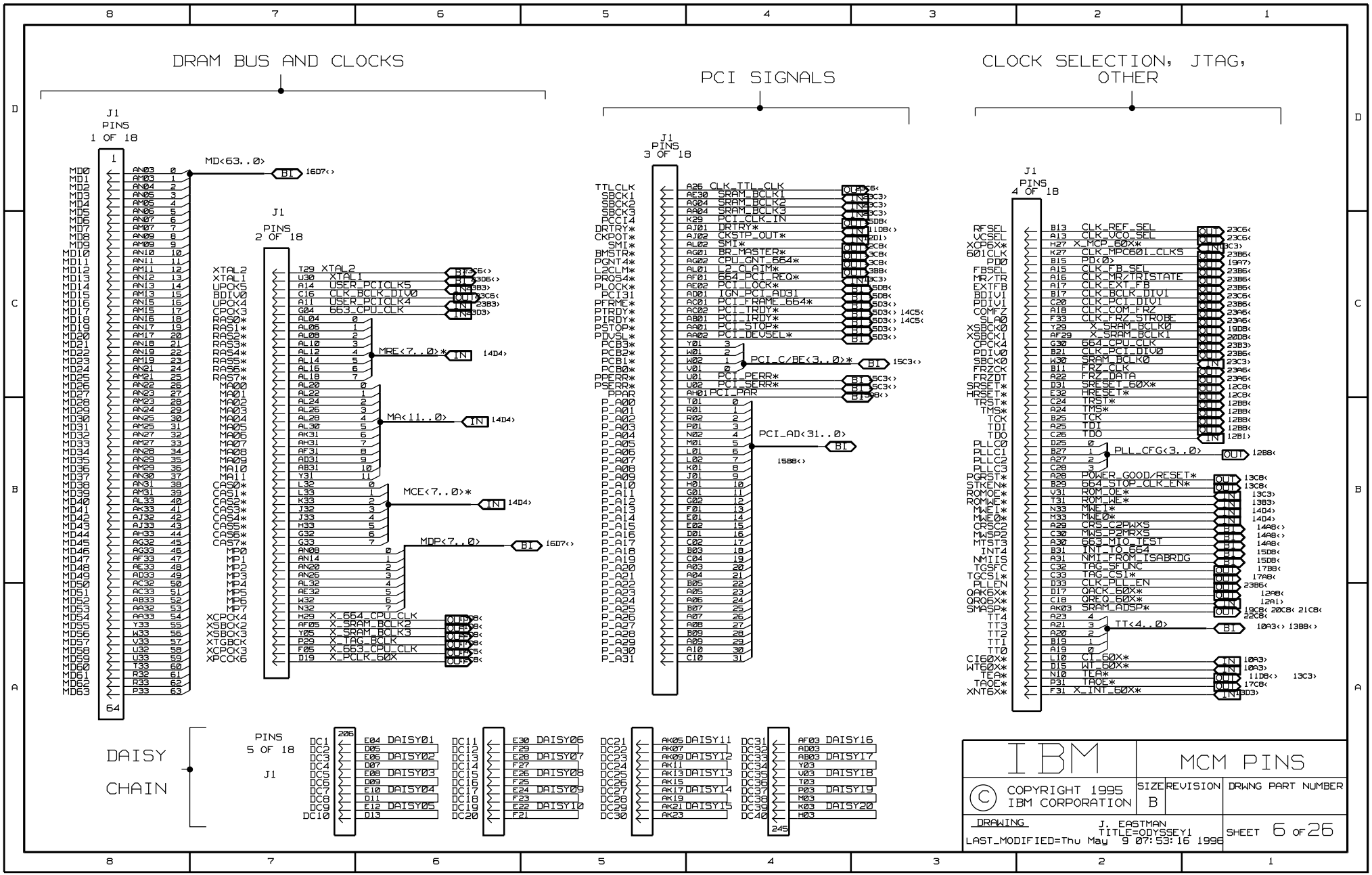
SHEET 4 OF 26

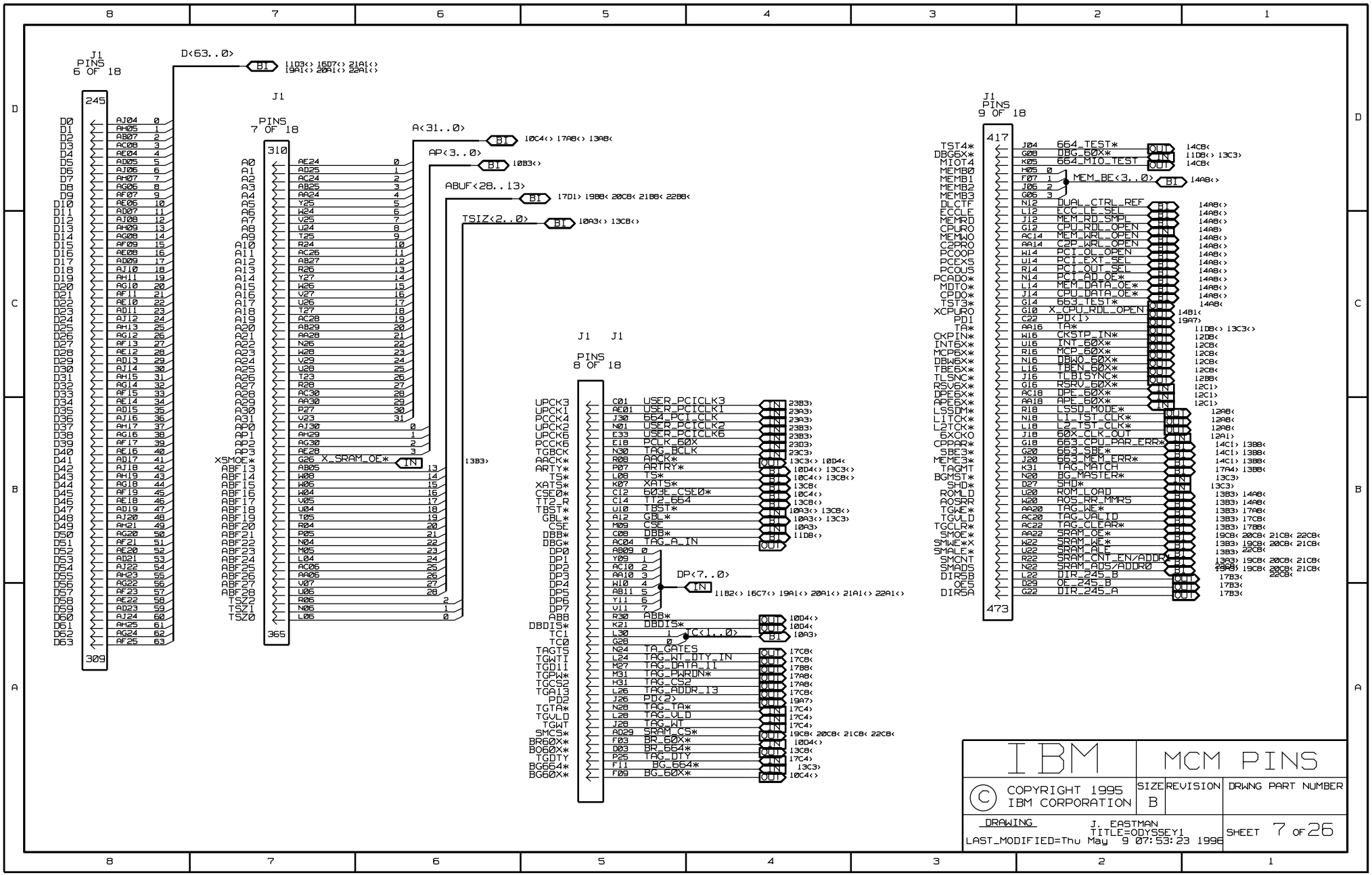
CONFIGURATION NOTES:

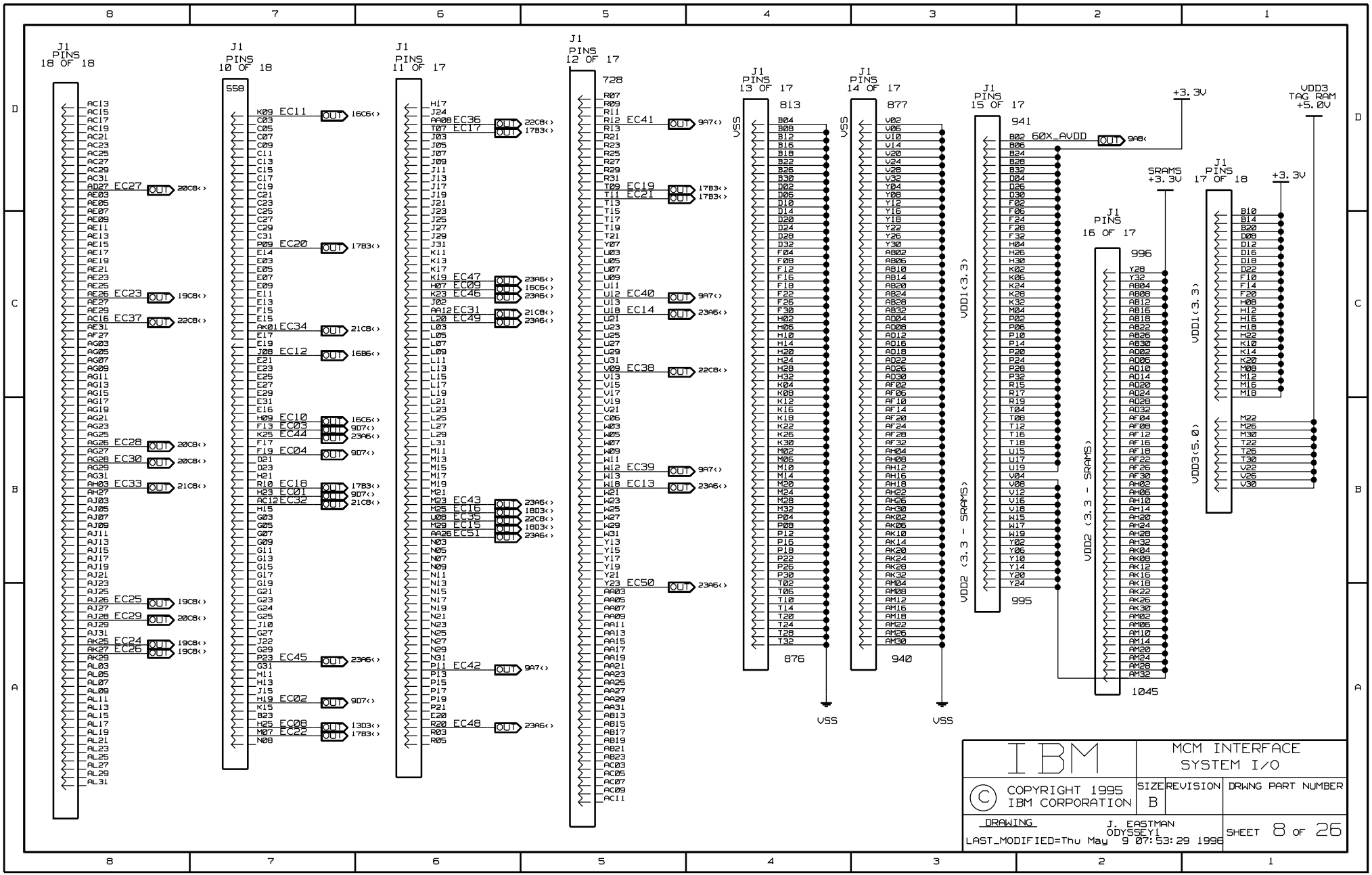
1. REQUIRED CHIP VERSIONS:
   PPC603E, 100 MHZ
   LANAI (663): 2.0
   KAUAI (664): 1.2 ONLY
2. CLOCK SPEEDS:
   60X BUS: 66 MHZ
   PCI: 33 MHZ
   603E PLL: 200 MHZ
   603E CPU: 100 MHZ

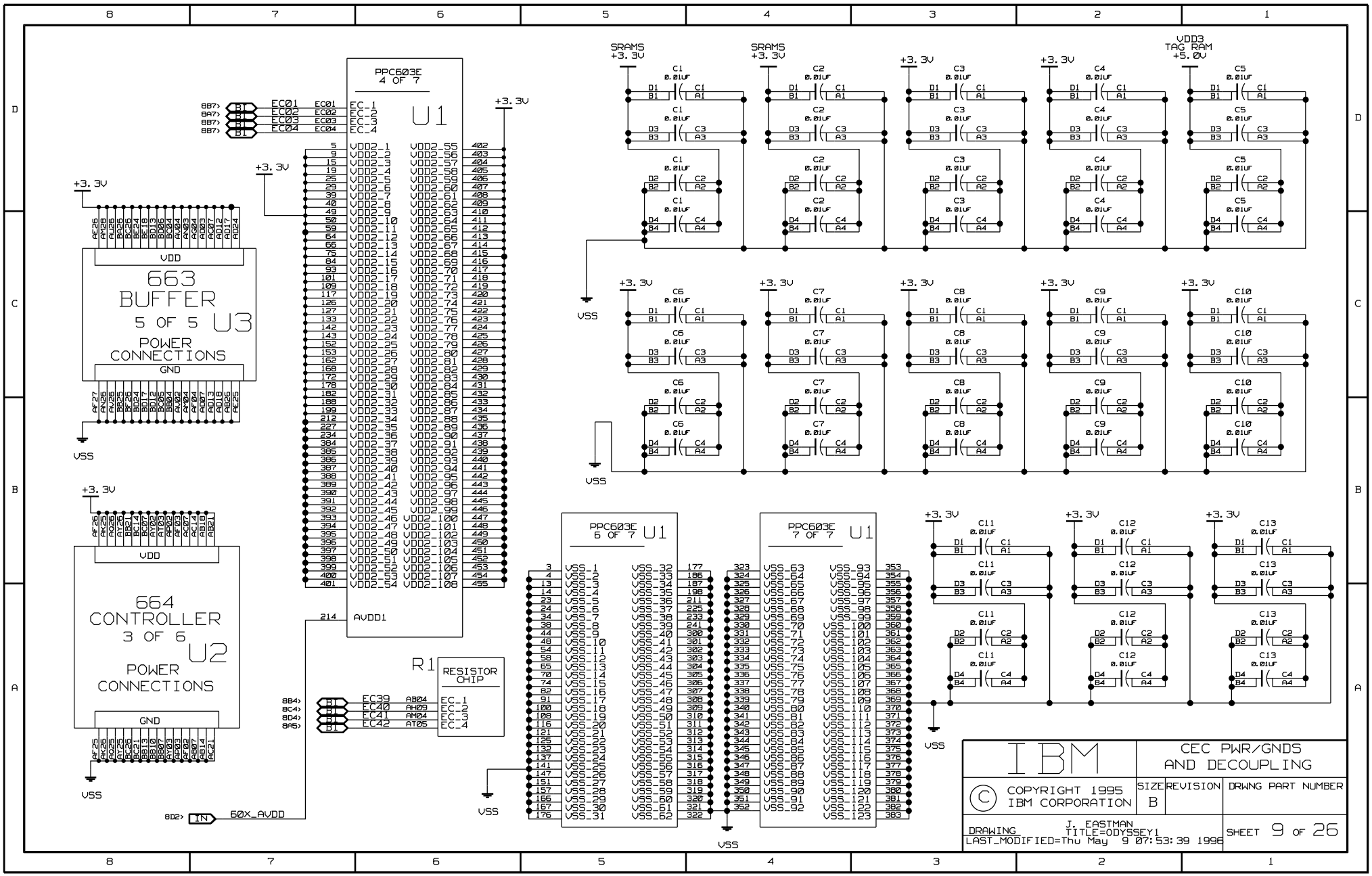
3. 603E PLL CONFIGURATION BITS
   PLL_CFG<0:3> OF 603E SHOULD BE SET TO 00 11 WITH
   APPROPRIATE RESISTORS ON MOTHERBOARD:
4. XTAL1 AND XTAL2 MUST BE CONNECTED TO 66 MHZ CRYSTAL
   OR TCLK MUST BE CONNECTED TO 66 MHZ CLOCK INPUT.

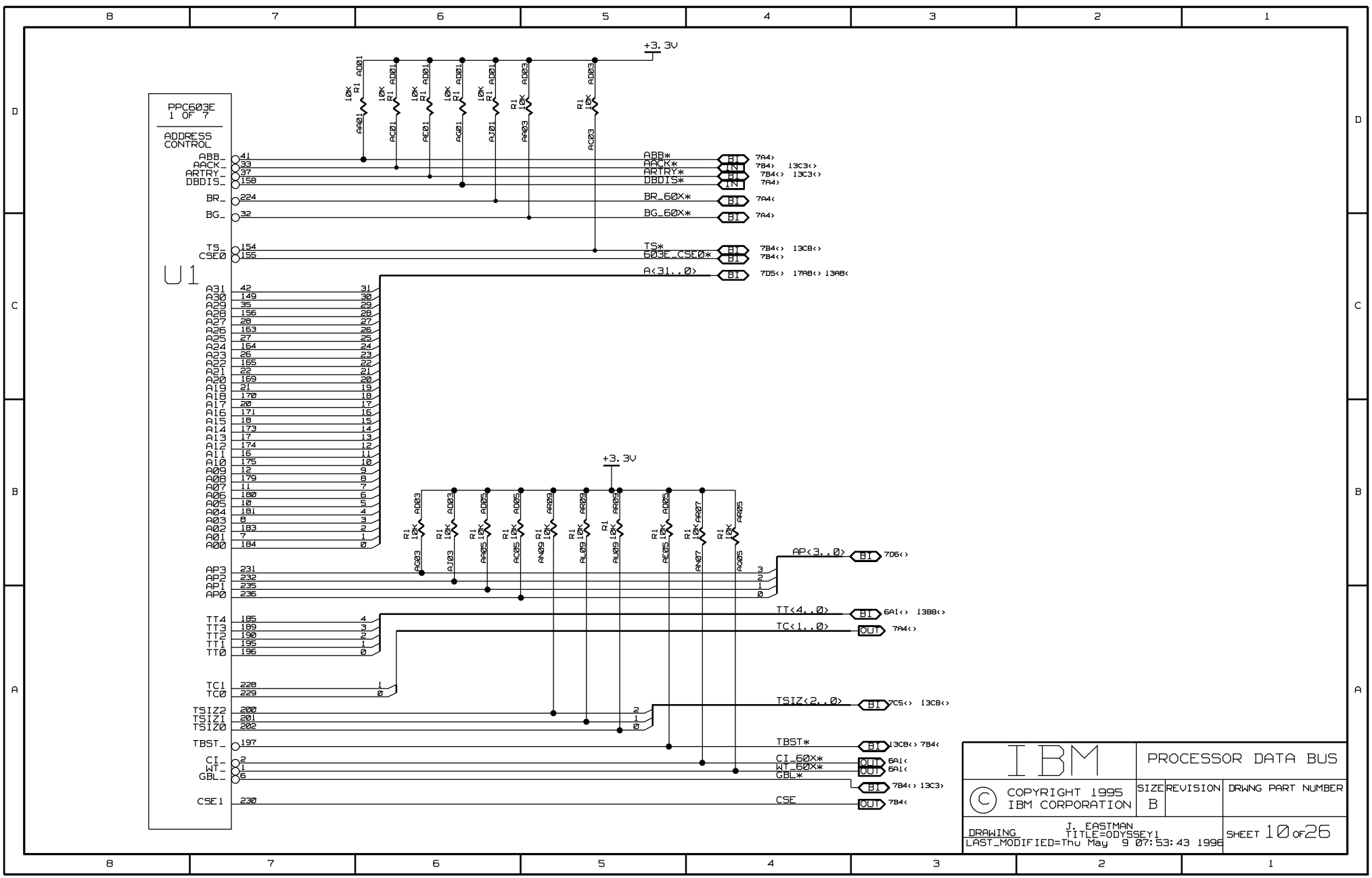
IBM	CONFIGURATION NOTES		
	SIZE B	REVISION	DRWNG PART NUMBER
DRAWING LAST_MODIFIED=Wed May 1 15:53:30 1996	J. EASTMAN TITLE=ODYSSEY1		SHEET 5 OF 26

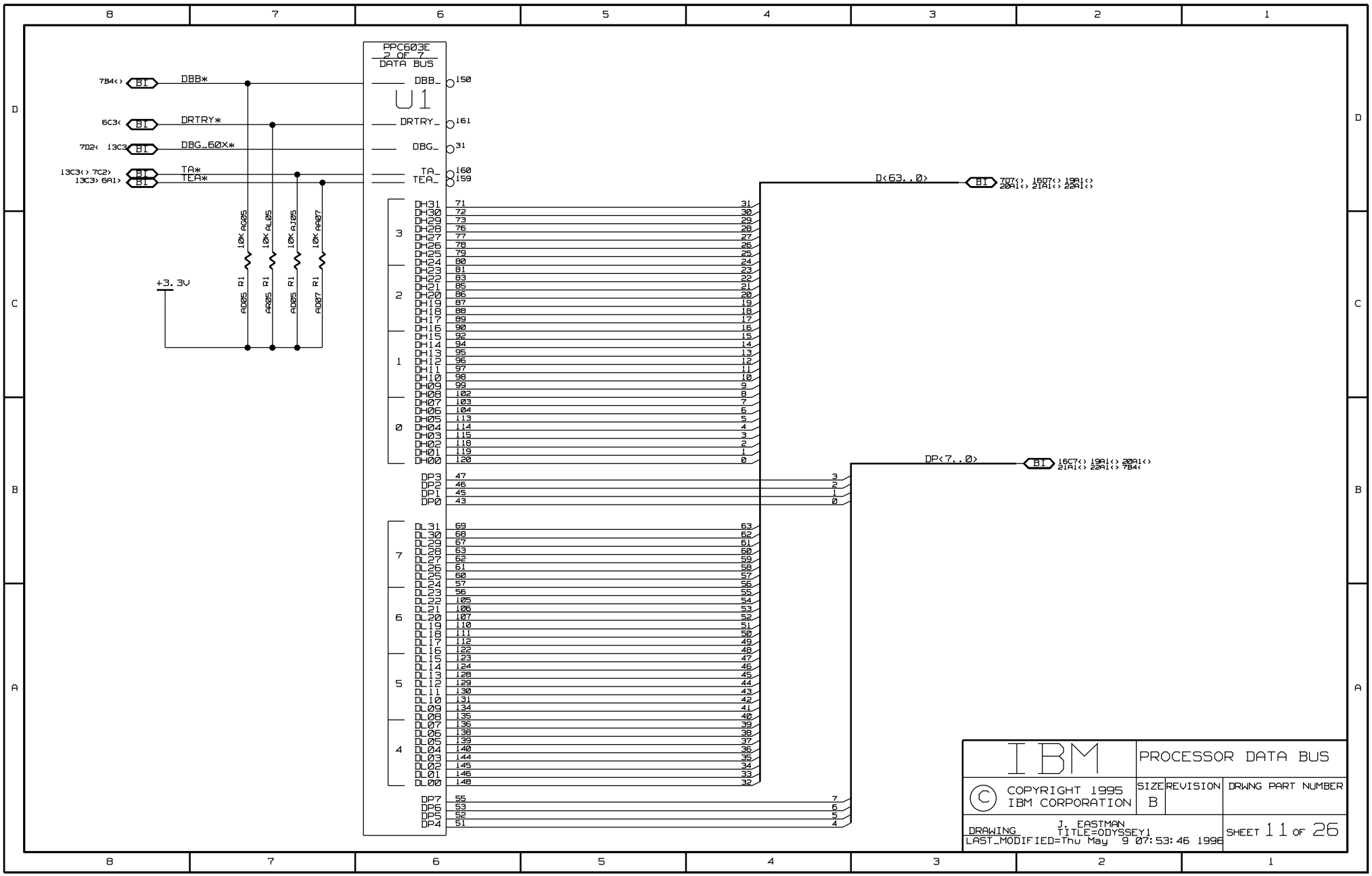

DRAM BUS AND CLOCKS
PCI SIGNALS
CLOCK SELECTION, JTAG, OTHER
J1 PINS 1 OF 18
J1 PINS 2 OF 18
J1 PINS 3 OF 18
J1 PINS 4 OF 18
J1 PINS 5 OF 18
MD<63..0>
MRE<7..0>*
MA<11..0>
MCE<7..0>*
MDP<7..0>
PCI_C/BE<3..0>*
PCI_AD<31..0>
PLL_CFG<3..0>
TT<4..0>
DAISY CHAIN
IBM
MCM PINS
COPYRIGHT 1995 IBM CORPORATION
SIZE B
REVISION
DRWG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:53:16 1996
SHEET 6 OF 26

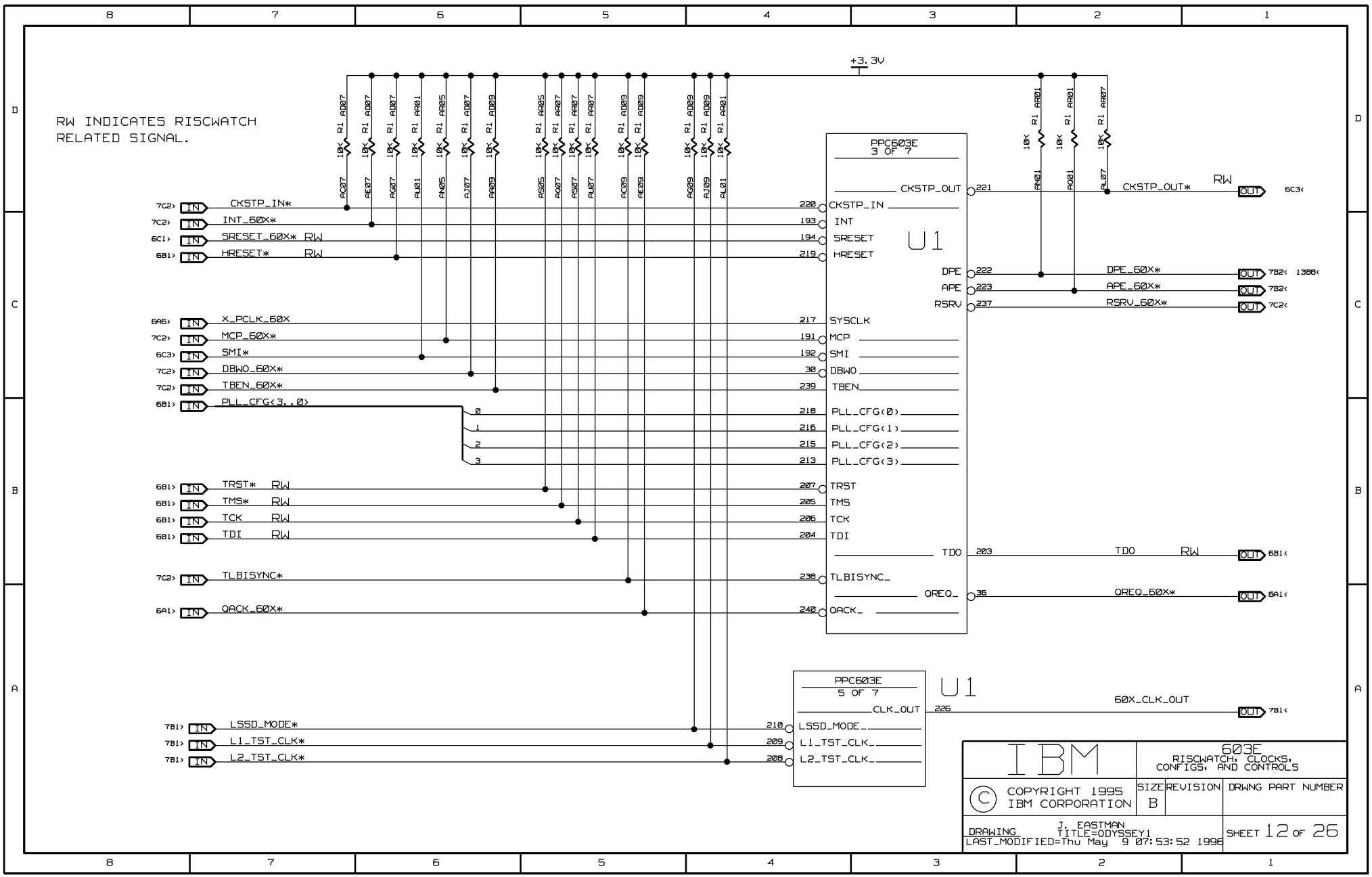

J1
PINS
6 OF 18
D<63..0>
J1
PINS
7 OF 18
A<31..0>
AP<3..0>
ABUF<28..13>
TSIZ<2..0>
J1
PINS
8 OF 18
DP<7..0>
TC<1..0>
J1
PINS
9 OF 18
MEM_BE<3..0>
IBM
MCM PINS
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:53:23 1996
SHEET 7 OF 26

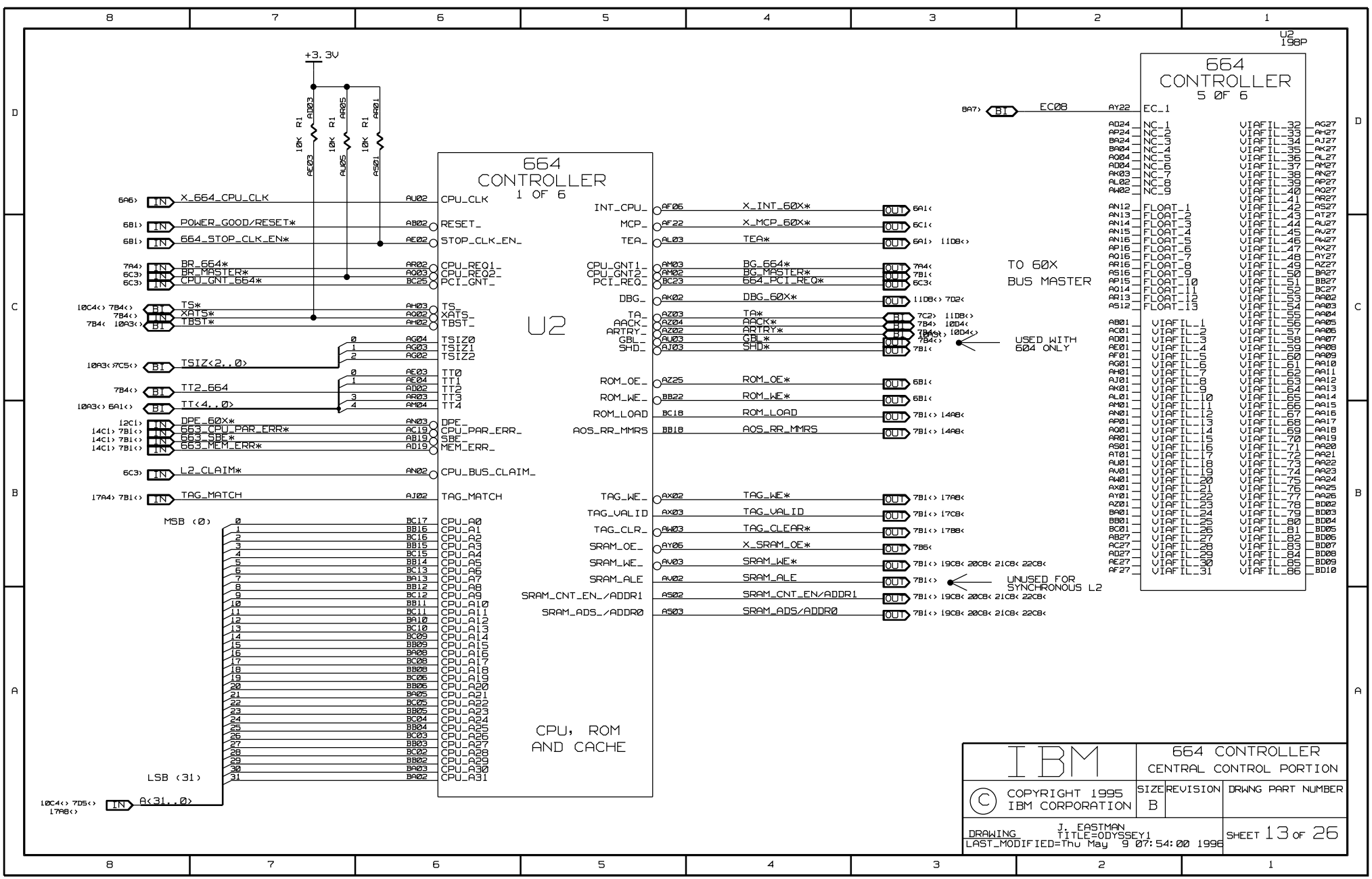

J1 PINS 18 OF 18
J1 PINS 10 OF 18
J1 PINS 11 OF 17
J1 PINS 12 OF 17
J1 PINS 13 OF 17
J1 PINS 14 OF 17
J1 PINS 15 OF 17
J1 PINS 16 OF 17
J1 PINS 17 OF 18
728
813
877
941
996
995
1045
876
940
VSS
VDD1(3.3)
VDD2 (3.3 - SRAMS)
VDD3(5.0)
+3.3V
SRAMS +3.3V
VDD3 TAG RAM +5.0V
60X_AVDD
IBM
MCM INTERFACE SYSTEM I/O
COPYRIGHT 1995 IBM CORPORATION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN ODYSSEY1
LAST_MODIFIED=Thu May 9 07:53:29 1996
SHEET 8 OF 26

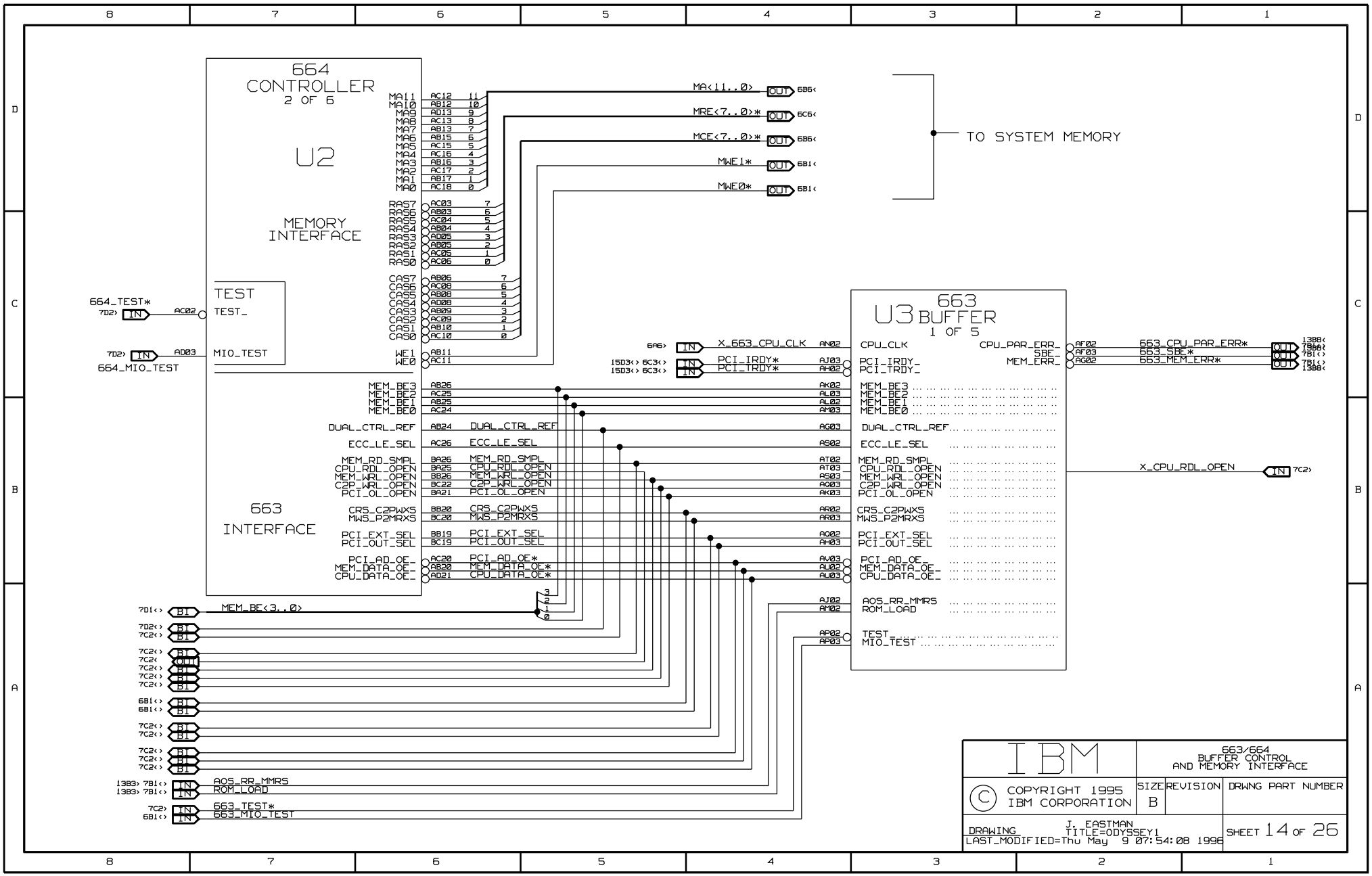

PPC603E
4 OF 7
U1
663
BUFFER
5 OF 5 U3
POWER
CONNECTIONS
664
CONTROLLER
3 OF 6
U2
POWER
CONNECTIONS
PPC603E
6 OF 7 U1
PPC603E
7 OF 7 U1
R1
RESISTOR
CHIP
SRAMS
+3.3V
VDD3
TAG RAM
+5.0V
+3.3V
VSS
VDD
GND
AVDD1
60X_AVDD
IBM
CEC PWR/GNDS
AND DECOUPLING
COPYRIGHT 1995
IBM CORPORATION
SIZE
B
REVISION
DRWG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:53:39 1996
SHEET 9 OF 26

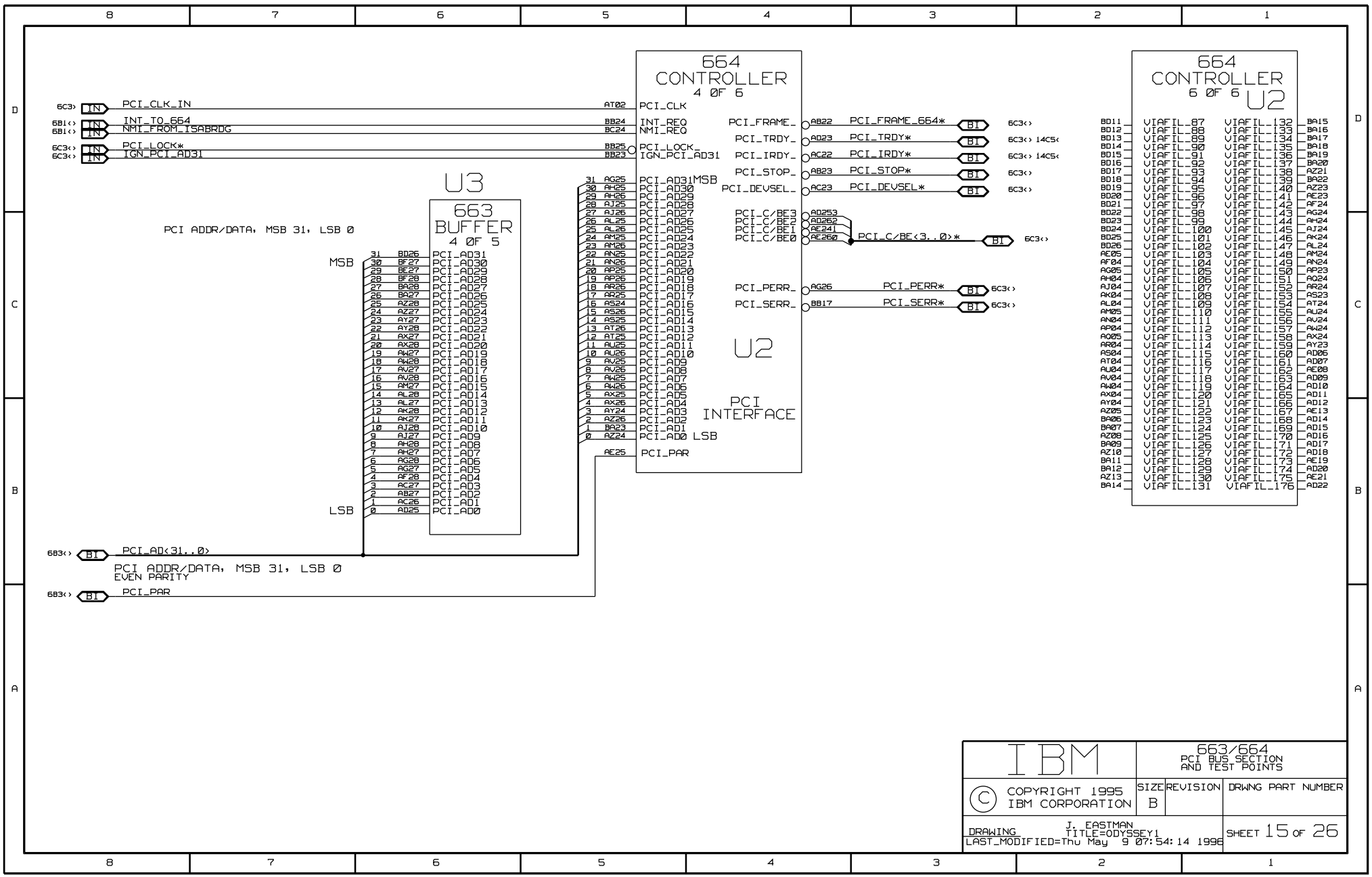

PPC603E
1 OF 7
ADDRESS
CONTROL
U1
+3.3V
ABB_
AACK_
ARTRY_
DBDIS_
BR_
BG_
TS_
CSE0
CSE1
A31
A30
A29
A28
A27
A26
A25
A24
A23
A22
A21
A20
A19
A18
A17
A16
A15
A14
A13
A12
A11
A10
A09
A08
A07
A06
A05
A04
A03
A02
A01
A00
AP3
AP2
AP1
AP0
TT4
TT3
TT2
TT1
TT0
TC1
TC0
TSIZ2
TSIZ1
TSIZ0
TBST_
CI_
WT_
GBL_
ABB*
AACK*
ARTRY*
DBDIS*
BR_60X*
BG_60X*
TS*
603E_CSE0*
A<31..0>
AP<3..0>
TT<4..0>
TC<1..0>
TSIZ<2..0>
TBST*
CI_60X*
WT_60X*
GBL*
CSE
BI
IN
OUT
7A4>
7B4<> 13C3<>
7B4<> 13C8<>
7B4<>
7D5<> 17A8<> 13A8<
7D6<>
6A1<> 13B8<>
7A4<>
7C5<> 13C8<>
13C8<> 7B4<
6A1<
7B4<> 13C3>
7B4<
IBM
PROCESSOR DATA BUS
COPYRIGHT 1995
IBM CORPORATION
SIZE
B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:53:43 1996
SHEET 10 OF 26


PPC603E
2 OF 7
DATA BUS
U1
DBB_
DRTRY_
DBG_
TA_
TEA_
DBB*
DRTRY*
DBG_60X*
TA*
TEA*
10K
+3.3V
D<63..0>
DP<7..0>
PROCESSOR DATA BUS
IBM
COPYRIGHT 1995
IBM CORPORATION
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:53:46 1996
SHEET 11 OF 26

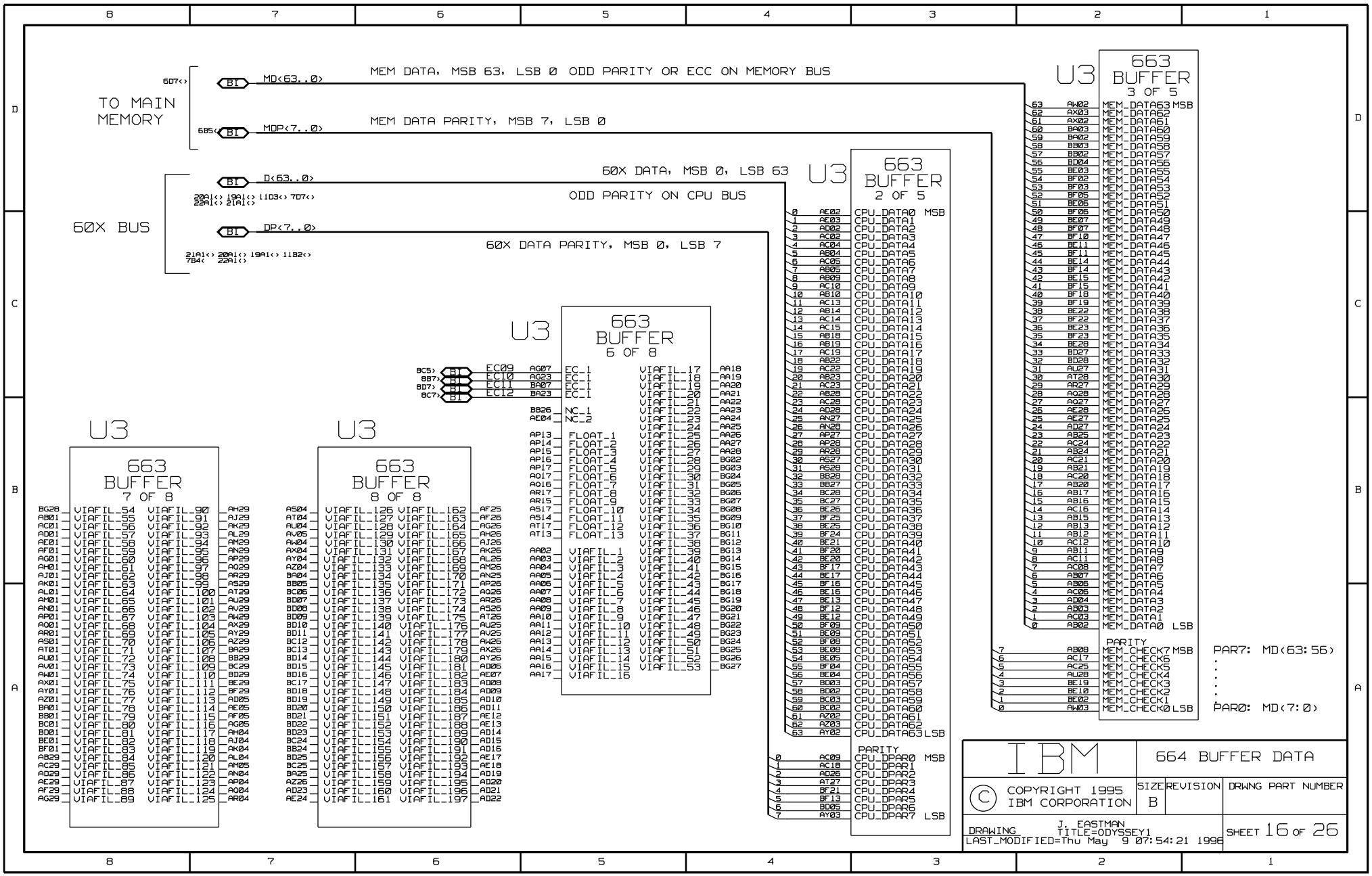

RW INDICATES RISCWATCH
RELATED SIGNAL.
+3.3V
PPC603E
3 OF 7
U1
CKSTP_OUT
CKSTP_IN
INT
SRESET
HRESET
DPE
APE
RSRV
SYSCLK
MCP
SMI
DBWO
TBEN
PLL_CFG(0)
PLL_CFG(1)
PLL_CFG(2)
PLL_CFG(3)
TRST
TMS
TCK
TDI
TDO
TLBISYNC_
QREQ_
QACK_
CKSTP_IN*
INT_60X*
SRESET_60X*
HRESET*
X_PCLK_60X
MCP_60X*
SMI*
DBWO_60X*
TBEN_60X*
PLL_CFG(3..0)
TRST*
TMS*
TCK
TDI
TLBISYNC*
QACK_60X*
CKSTP_OUT*
DPE_60X*
APE_60X*
RSRV_60X*
TDO
QREQ_60X*
PPC603E
5 OF 7
U1
CLK_OUT
60X_CLK_OUT
LSSD_MODE*
L1_TST_CLK*
L2_TST_CLK*
LSSD_MODE_
L1_TST_CLK_
L2_TST_CLK_
IBM
603E
RISCWATCH, CLOCKS,
CONFIGS, AND CONTROLS
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:53:52 1996
SHEET 12 OF 26

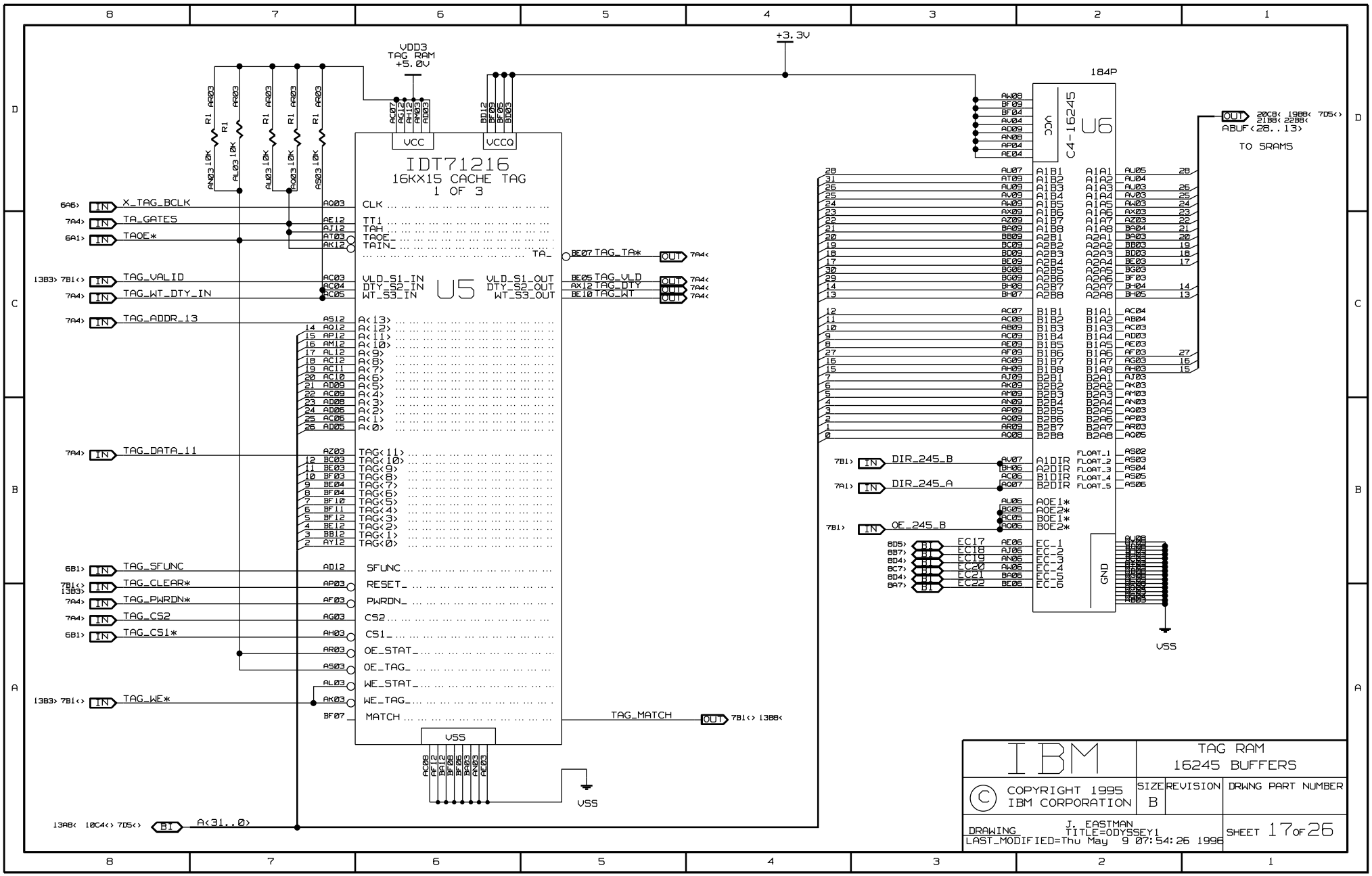

664 CONTROLLER 1 OF 6
U2
CPU, ROM AND CACHE
664 CONTROLLER 5 OF 6
U2 198P
+3.3V
10K R1
TO 60X BUS MASTER
USED WITH 604 ONLY
UNUSED FOR SYNCHRONOUS L2
MSB (0)
LSB (31)
IBM
664 CONTROLLER
CENTRAL CONTROL PORTION
COPYRIGHT 1995 IBM CORPORATION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:54:00 1996
SHEET 13 OF 26

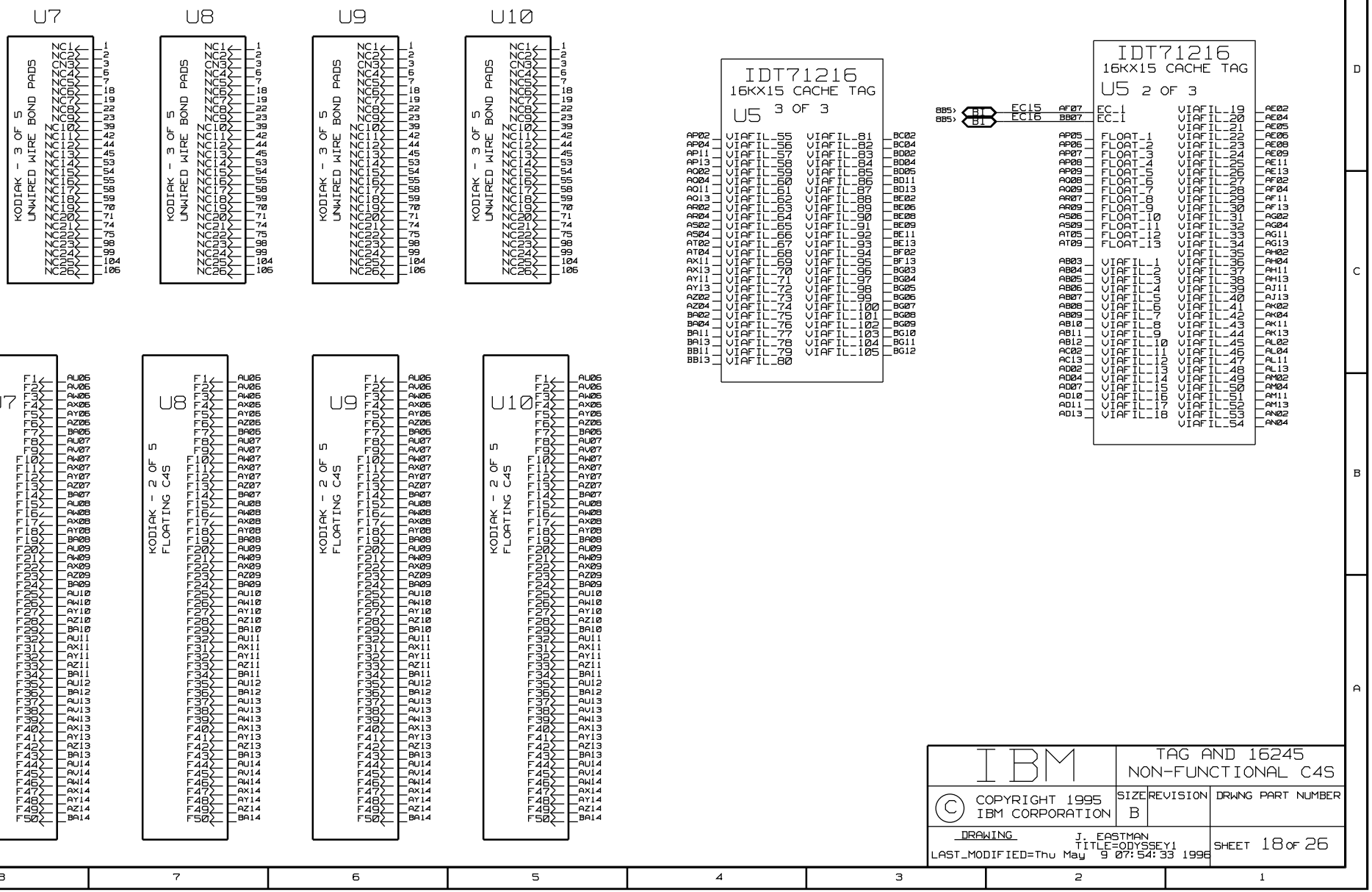

664
CONTROLLER
2 OF 6
U2
MEMORY
INTERFACE
TEST
TEST_
MIO_TEST
663
INTERFACE
664_TEST*
664_MIO_TEST
MA<11..0>
MRE<7..0>*
MCE<7..0>*
MWE1*
MWE0*
TO SYSTEM MEMORY
663
U3 BUFFER
1 OF 5
X_663_CPU_CLK
PCI_IRDY*
PCI_TRDY*
CPU_CLK
PCI_IRDY_
PCI_TRDY_
CPU_PAR_ERR_
SBE_
MEM_ERR_
663_CPU_PAR_ERR*
663_SBE*
663_MEM_ERR*
MEM_BE3
MEM_BE2
MEM_BE1
MEM_BE0
DUAL_CTRL_REF
ECC_LE_SEL
MEM_RD_SMPL
CPU_RDL_OPEN
MEM_WRL_OPEN
C2P_WRL_OPEN
PCI_OL_OPEN
CRS_C2PWXS
MWS_P2MRXS
PCI_EXT_SEL
PCI_OUT_SEL
PCI_AD_OE_
MEM_DATA_OE_
CPU_DATA_OE_
X_CPU_RDL_OPEN
MEM_BE<3..0>
AOS_RR_MMRS
ROM_LOAD
TEST
MIO_TEST
663_TEST*
663_MIO_TEST
IBM
663/664
BUFFER CONTROL
AND MEMORY INTERFACE
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:54:08 1996
SHEET 14 OF 26

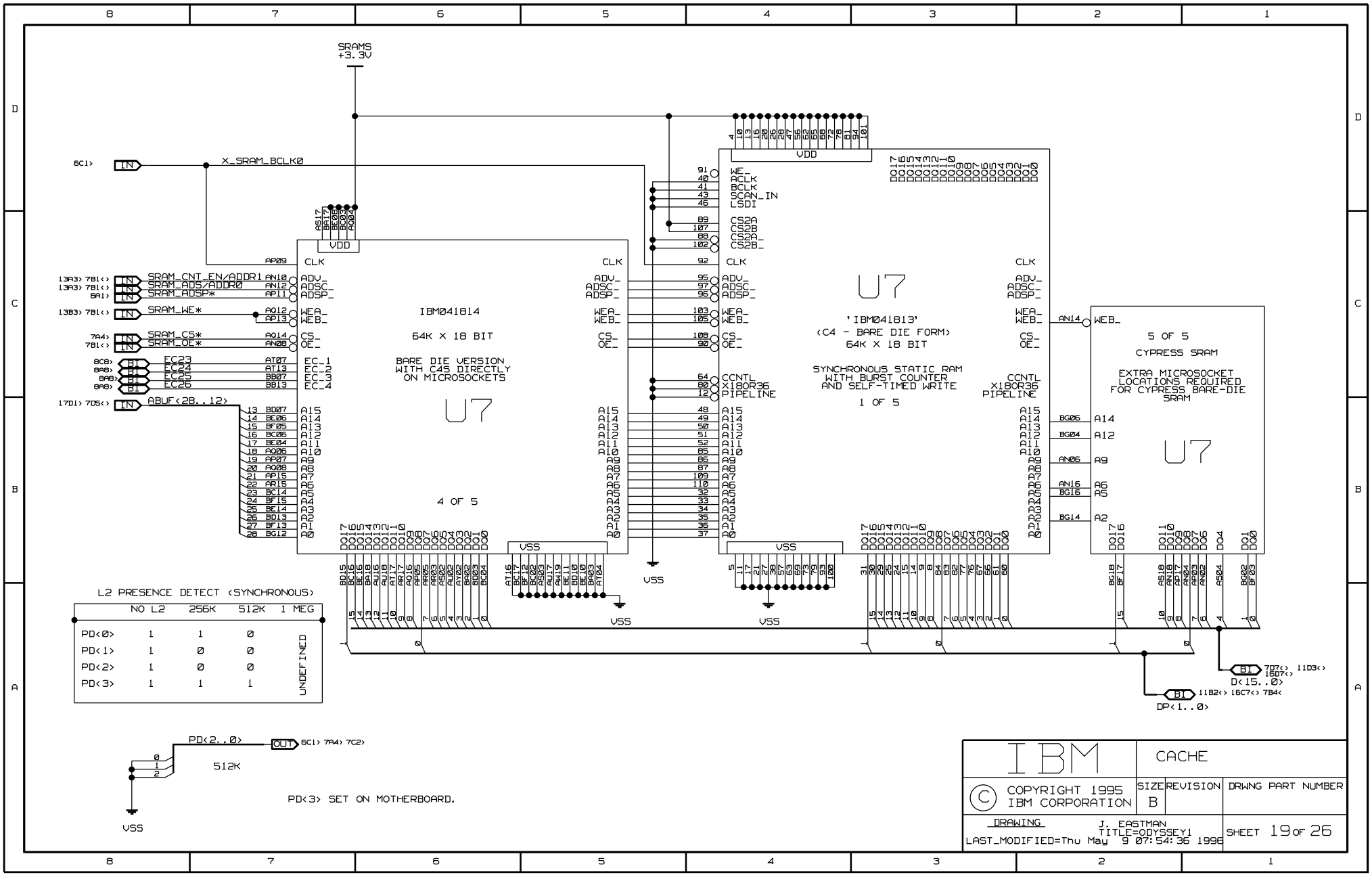


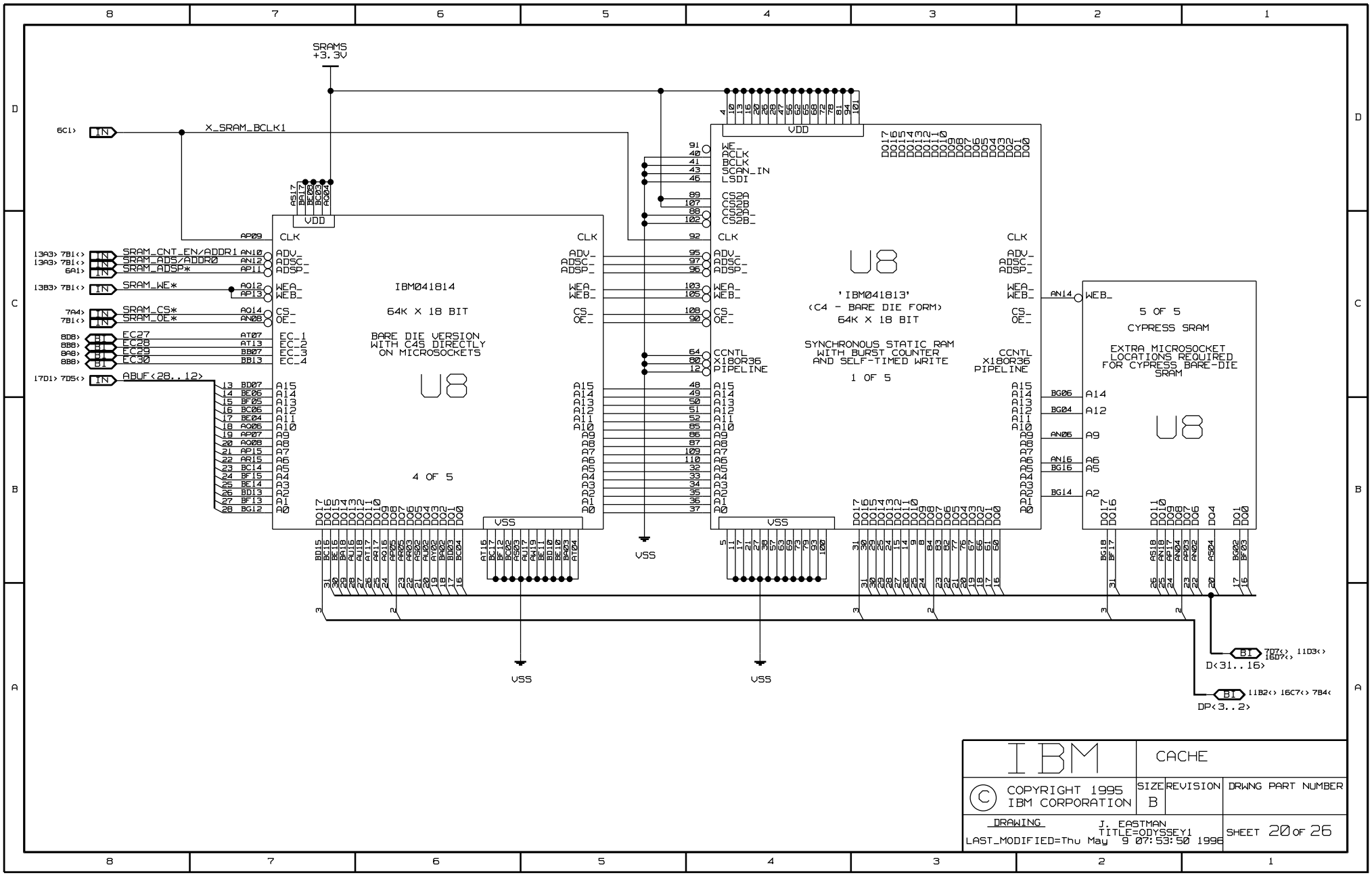
IBM	663/664 PCI BUS SECTION AND TEST POINTS	
© COPYRIGHT 1995 IBM CORPORATION	SIZE B	REVISION / DRWG PART NUMBER
DRAWING J. EASTMAN TITLE=ODYSSEY1 LAST_MODIFIED=Thu May 9 07:54:14 1996	SHEET 15 OF 26	

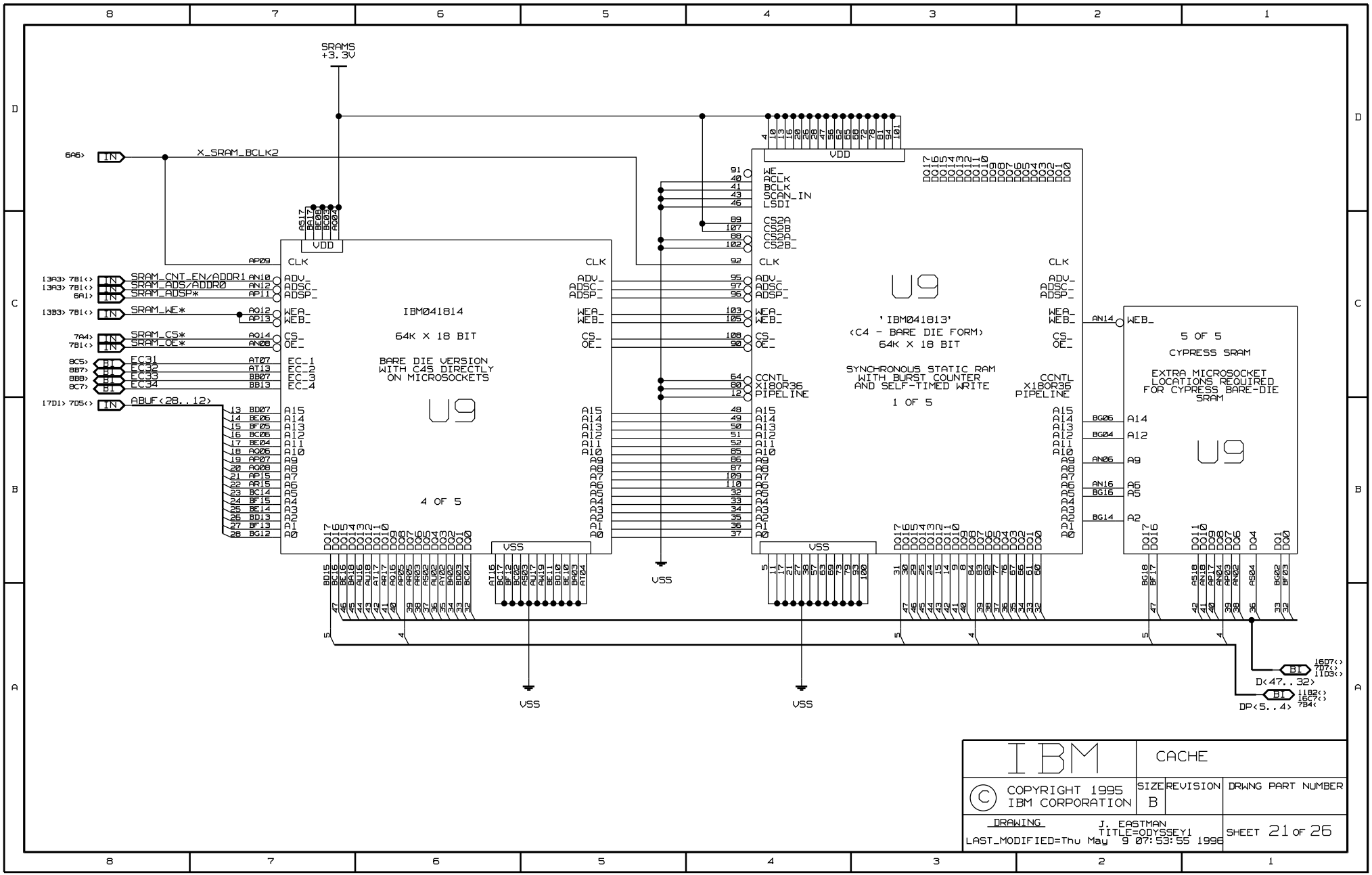

TO MAIN MEMORY
MD<63..0>
MEM DATA, MSB 63, LSB 0 ODD PARITY OR ECC ON MEMORY BUS
MDP<7..0>
MEM DATA PARITY, MSB 7, LSB 0
60X BUS
D<63..0>
60X DATA, MSB 0, LSB 63
ODD PARITY ON CPU BUS
DP<7..0>
60X DATA PARITY, MSB 0, LSB 7
U3
663 BUFFER
2 OF 5
CPU_DATA0 MSB
CPU_DATA63 LSB
PARITY
CPU_DPAR0 MSB
CPU_DPAR7 LSB
U3
663 BUFFER
3 OF 5
MEM_DATA63 MSB
MEM_DATA0 LSB
PARITY
MEM_CHECK7 MSB
MEM_CHECK0 LSB
PAR7: MD<63:56>
PAR0: MD<7:0>
U3
663 BUFFER
6 OF 8
U3
663 BUFFER
7 OF 8
U3
663 BUFFER
8 OF 8
IBM
664 BUFFER DATA
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:54:21 1996
SHEET 16 OF 26

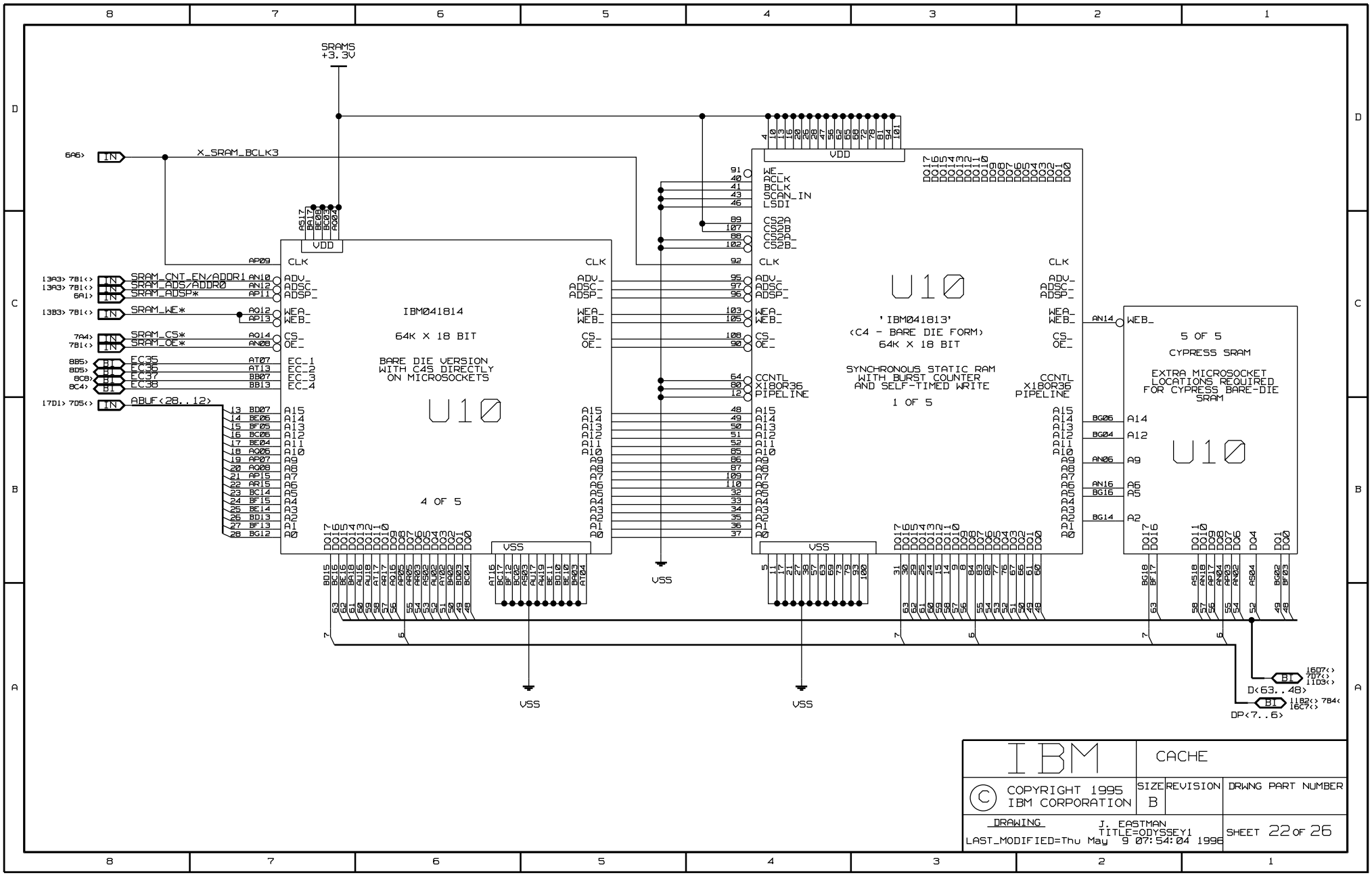

VDD3
TAG RAM
+5.0V
+3.3V
VCC
VCCQ
IDT71216
16KX15 CACHE TAG
1 OF 3
U5
X_TAG_BCLK
TA_GATES
TAOE*
TAG_VALID
TAG_WT_DTY_IN
TAG_ADDR_13
TAG_DATA_11
TAG_SFUNC
TAG_CLEAR*
TAG_PWRDN*
TAG_CS2
TAG_CS1*
TAG_WE*
A<31..0>
CLK
TAH
TAOE_
TAIN_
TA_
VLD_S1_IN
DTY_S2_IN
WT_S3_IN
VLD_S1_OUT
DTY_S2_OUT
WT_S3_OUT
TAG_TA*
TAG_VLD
TAG_DTY
TAG_WT
SFUNC
RESET_
PWRDN_
CS2
CS1_
OE_STAT_
OE_TAG_
WE_STAT_
WE_TAG_
MATCH
TAG_MATCH
VSS
184P
U6
C4-16245
DIR_245_B
DIR_245_A
OE_245_B
ABUF<28..13>
TO SRAMS
IBM
TAG RAM
16245 BUFFERS
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:54:26 1996
SHEET 17 OF 26

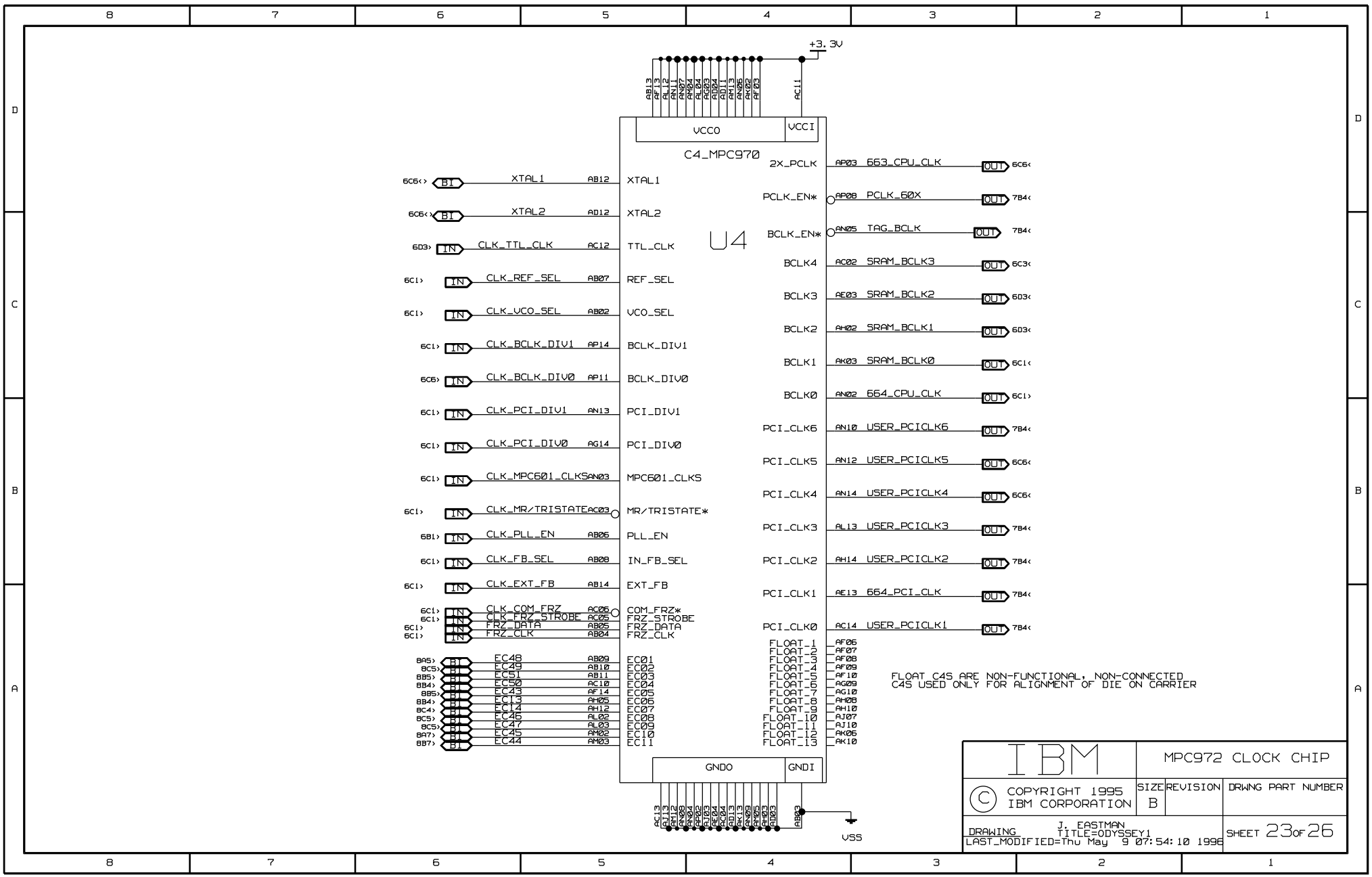

U7
U8
U9
U10
KODIAK - 3 OF 5
UNWIRED WIRE BOND PADS
KODIAK - 2 OF 5
FLOATING C4S
IDT71216
16KX15 CACHE TAG
U5 3 OF 3
U5 2 OF 3
VIAFIL_55
VIAFIL_56
VIAFIL_57
VIAFIL_58
VIAFIL_59
VIAFIL_60
VIAFIL_61
VIAFIL_62
VIAFIL_63
VIAFIL_64
VIAFIL_65
VIAFIL_66
VIAFIL_67
VIAFIL_68
VIAFIL_69
VIAFIL_70
VIAFIL_71
VIAFIL_72
VIAFIL_73
VIAFIL_74
VIAFIL_75
VIAFIL_76
VIAFIL_77
VIAFIL_78
VIAFIL_79
VIAFIL_80
VIAFIL_81
VIAFIL_82
VIAFIL_83
VIAFIL_84
VIAFIL_85
VIAFIL_86
VIAFIL_87
VIAFIL_88
VIAFIL_89
VIAFIL_90
VIAFIL_91
VIAFIL_92
VIAFIL_93
VIAFIL_94
VIAFIL_95
VIAFIL_96
VIAFIL_97
VIAFIL_98
VIAFIL_99
VIAFIL_100
VIAFIL_101
VIAFIL_102
VIAFIL_103
VIAFIL_104
VIAFIL_105
EC15
EC16
FLOAT_1
FLOAT_2
FLOAT_3
FLOAT_4
FLOAT_5
FLOAT_6
FLOAT_7
FLOAT_8
FLOAT_9
FLOAT_10
FLOAT_11
FLOAT_12
FLOAT_13
VIAFIL_1
VIAFIL_2
VIAFIL_3
VIAFIL_4
VIAFIL_5
VIAFIL_6
VIAFIL_7
VIAFIL_8
VIAFIL_9
VIAFIL_10
VIAFIL_11
VIAFIL_12
VIAFIL_13
VIAFIL_14
VIAFIL_15
VIAFIL_16
VIAFIL_17
VIAFIL_18
VIAFIL_19
VIAFIL_20
VIAFIL_21
VIAFIL_22
VIAFIL_23
VIAFIL_24
VIAFIL_25
VIAFIL_26
VIAFIL_27
VIAFIL_28
VIAFIL_29
VIAFIL_30
VIAFIL_31
VIAFIL_32
VIAFIL_33
VIAFIL_34
VIAFIL_35
VIAFIL_36
VIAFIL_37
VIAFIL_38
VIAFIL_39
VIAFIL_40
VIAFIL_41
VIAFIL_42
VIAFIL_43
VIAFIL_44
VIAFIL_45
VIAFIL_46
VIAFIL_47
VIAFIL_48
VIAFIL_49
VIAFIL_50
VIAFIL_51
VIAFIL_52
VIAFIL_53
VIAFIL_54
IBM
TAG AND 16245
NON-FUNCTIONAL C4S
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:54:33 1996
SHEET 18 OF 26


SRAMS
+3.3V
X_SRAM_BCLK0
SRAM_CNT_EN/ADDR1
SRAM_ADS/ADDR0
SRAM_ADSP*
SRAM_WE*
SRAM_CS*
SRAM_OE*
EC23
EC24
EC25
EC26
ABUF<28..12>
IBM041814
64K X 18 BIT
BARE DIE VERSION
WITH C4S DIRECTLY
ON MICROSOCKETS
4 OF 5
'IBM041813'
<C4 - BARE DIE FORM>
64K X 18 BIT
SYNCHRONOUS STATIC RAM
WITH BURST COUNTER
AND SELF-TIMED WRITE
1 OF 5
5 OF 5
CYPRESS SRAM
EXTRA MICROSOCKET
LOCATIONS REQUIRED
FOR CYPRESS BARE-DIE
SRAM
L2 PRESENCE DETECT (SYNCHRONOUS)
NO L2 256K 512K 1 MEG
PD<0> 1 1 0
PD<1> 1 0 0
PD<2> 1 0 0
PD<3> 1 1 1
UNDEFINED
PD<2..0>
512K
PD<3> SET ON MOTHERBOARD.
D<15..0>
DP<1..0>
VDD
VSS
CLK
CS_
WEA_
WEB_
ADV_
ADSC_
ADSP_
OE_
CCNTL
X18OR36
PIPELINE
IBM
CACHE
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:54:36 1996
SHEET 19 OF 26


SRAMS
+3.3V
X_SRAM_BCLK1
SRAM_CNT_EN/ADDR1
SRAM_ADS/ADDR0
SRAM_ADSP*
SRAM_WE*
SRAM_CS*
SRAM_OE*
ABUF<28..12>
IBM041814
64K X 18 BIT
BARE DIE VERSION
WITH C4S DIRECTLY
ON MICROSOCKETS
4 OF 5
'IBM041813'
<C4 - BARE DIE FORM>
64K X 18 BIT
SYNCHRONOUS STATIC RAM
WITH BURST COUNTER
AND SELF-TIMED WRITE
1 OF 5
5 OF 5
CYPRESS SRAM
EXTRA MICROSOCKET
LOCATIONS REQUIRED
FOR CYPRESS BARE-DIE
SRAM
VDD
VSS
D<31..16>
DP<3..2>
IBM
CACHE
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:53:50 1996
SHEET 20 OF 26


SRAM5
+3.3V
X_SRAM_BCLK2
SRAM_CNT_EN/ADDR1
SRAM_ADS/ADDR0
SRAM_ADSP*
SRAM_WE*
SRAM_CS*
SRAM_OE*
ABUF<28..12>
IBM041814
64K X 18 BIT
BARE DIE VERSION
WITH C4S DIRECTLY
ON MICROSOCKETS
4 OF 5
'IBM041813'
<C4 - BARE DIE FORM>
64K X 18 BIT
SYNCHRONOUS STATIC RAM
WITH BURST COUNTER
AND SELF-TIMED WRITE
1 OF 5
5 OF 5
CYPRESS SRAM
EXTRA MICROSOCKET
LOCATIONS REQUIRED
FOR CYPRESS BARE-DIE
SRAM
VDD
VSS
D<47..32>
DP<5..4>
IBM
CACHE
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:53:55 1996
SHEET 21 OF 26


SRAMS
+3.3V
X_SRAM_BCLK3
SRAM_CNT_EN/ADDR1
SRAM_ADS/ADDR0
SRAM_ADSP*
SRAM_WE*
SRAM_CS*
SRAM_OE*
ABUF<28..12>
IBM041814
64K X 18 BIT
BARE DIE VERSION
WITH C4S DIRECTLY
ON MICROSOCKETS
4 OF 5
'IBM041813'
<C4 - BARE DIE FORM>
64K X 18 BIT
SYNCHRONOUS STATIC RAM
WITH BURST COUNTER
AND SELF-TIMED WRITE
1 OF 5
5 OF 5
CYPRESS SRAM
EXTRA MICROSOCKET
LOCATIONS REQUIRED
FOR CYPRESS BARE-DIE
SRAM
VDD
VSS
D<63..48>
DP<7..6>
IBM
CACHE
COPYRIGHT 1995
IBM CORPORATION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:54:04 1996
SHEET 22 OF 26


+3.3V
VCC0
VCCI
C4_MPC970
U4
XTAL1
XTAL2
CLK_TTL_CLK
CLK_REF_SEL
CLK_VCO_SEL
CLK_BCLK_DIV1
CLK_BCLK_DIV0
CLK_PCI_DIV1
CLK_PCI_DIV0
CLK_MPC601_CLKS
CLK_MR/TRISTATE
CLK_PLL_EN
CLK_FB_SEL
CLK_EXT_FB
CLK_COM_FRZ
CLK_FRZ_STROBE
FRZ_DATA
FRZ_CLK
EC48
EC49
EC51
EC50
EC43
EC13
EC14
EC46
EC47
EC45
EC44
2X_PCLK
PCLK_EN*
BCLK_EN*
BCLK4
BCLK3
BCLK2
BCLK1
BCLK0
PCI_CLK6
PCI_CLK5
PCI_CLK4
PCI_CLK3
PCI_CLK2
PCI_CLK1
PCI_CLK0
663_CPU_CLK
PCLK_60X
TAG_BCLK
SRAM_BCLK3
SRAM_BCLK2
SRAM_BCLK1
SRAM_BCLK0
664_CPU_CLK
USER_PCICLK6
USER_PCICLK5
USER_PCICLK4
USER_PCICLK3
USER_PCICLK2
664_PCI_CLK
USER_PCICLK1
FLOAT_1
FLOAT_2
FLOAT_3
FLOAT_4
FLOAT_5
FLOAT_6
FLOAT_7
FLOAT_8
FLOAT_9
FLOAT_10
FLOAT_11
FLOAT_12
FLOAT_13
FLOAT C4S ARE NON-FUNCTIONAL, NON-CONNECTED
C4S USED ONLY FOR ALIGNMENT OF DIE ON CARRIER
GNDO
GNDI
VSS
IBM
MPC972 CLOCK CHIP
COPYRIGHT 1995
IBM CORPORATION
SIZE
B
REVISION
DRWNG PART NUMBER
DRAWING
J. EASTMAN
TITLE=ODYSSEY1
LAST_MODIFIED=Thu May 9 07:54:10 1996
SHEET 23 OF 26

*** Signal Cross-Reference ***
--- for the entire design --

```
60X_AVDD          8D2> 9AB<
60X_CLK_OUT       12A1> 7B1<
603E_CSE0 *       7B4<> 10C4<>
663_CPU_CLK       23D3> 6C6<
663_CPU_PAR_ERR * 7B1<> 14C1> 13B8<
663_MEM_ERR *     7B1<> 14C1> 13B8<
663_MIO_TEST      6B1<> 14AB<
663_SBE *         7B1<> 14C1> 13B8<
663_TEST *        7C2> 14AB<
664_CPU_CLK       6C1> 23B3>
664_MIO_TEST      7D2> 14CB<
664_PCI_CLK       23A3> 7B4<
664_PCI_REQ *     13C3> 6C3<
664_STOP_CLK_EN * 6B1> 13CB<
664_TEST *        7D2> 14CB<
A <31..0>         7D5<> 10C4<> 17AB<> 13AB<
AACK *            7B4> 13C3> 10D4<
ABB *             7A4> 10D4<>
ABUF <28..12>     7D5<> 17D1> 19B8< 20C8< 21B8< 22B8<

AOS_RR_MMRS       7B1<> 13B3> 14AB<
AP <3..0>         7D6<> 10B3<>
APE_60X *         12C1> 7B2<
ARTRY *           7B4<> 10D4<> 13C3<>
BG_60X *          7A4> 10C4<>
BG_664 *          13C3> 7A4<
BG_MASTER *       13C3> 7B1<
BR_60X *          10D4<> 7A4<
BR_664 *          7A4> 13CB<
BR_MASTER *       6C3> 13C8<
C2P_WRL_OPEN      7C2<> 14AB<>
CI_60X *          10A3> 6A1<
CKSTP_IN *        7C2> 12D8<
CKSTP_OUT *       12D1> 6C3<
CLK_BCLK_DIV0     6C6> 23C6<
CLK_BCLK_DIV1     6C1> 23C6<
CLK_COM_FRZ       6C1> 23A6<
CLK_EXT_FB        6C1> 23B6<
CLK_FB_SEL        6C1> 23B6<
CLK_FRZ_STROBE    6C1> 23A6<
CLK_MPC601_CLK5   6C1> 23B6<
CLK_MR/TRISTATE   6C1> 23B6<
CLK_PCI_DIV0      6C1> 23B6<
CLK_PCI_DIV1      6C1> 23B6<
CLK_PLL_EN        6B1> 23B6<
CLK_REF_SEL       6C1> 23C6<
CLK_TTL_CLK       6D3> 23C6<
CLK_VCO_SEL       6C1> 23C6<
CPU_DATA_OE *     7C2<> 14AB<>
CPU_GNT_664 *     6C3> 13CB<
CPU_RDL_OPEN      14AB> 7C2<
CRS_C2PWX5        6B1<> 14AB<>
CSE               10A3> 7B4<
D <63..0>         7D7<> 11D3<> 16D7<> 19A1<> 20A1<>
                  21A1<> 22A1<>
DAISY01           6A6
DAISY02           6A6
DAISY03           6A6
DAISY04           6A6
DAISY05           6A6
DAISY06           6A5
DAISY07           6A5
DAISY08           6A5

DAISY09           6A5
DAISY10           6A5
DAISY11           6A5
DAISY12           6A5
DAISY13           6A5
DAISY14           6A5
DAISY15           6A5
DAISY16           6A4
DAISY17           6A4
DAISY18           6A4
DAISY19           6A4
DAISY20           6A4
DBB *             7B4<> 11D8<>
DBDIS *           7A4> 10D4<
DBG_60X *         11D8<> 13C3> 7D2<
DBWO_60X *        7C2> 12C8<
DIR_245_A         7A1> 17B3<
DIR_245_B         7B1> 17B3<
DP <7..0>         11B2<> 16C7<> 19A1<> 20A1<> 21A1<>
                  22A1<> 7B4<
DPE_60X *         12C1> 7B2< 13B8<
DRTRY *           11D8<> 6C3<
DUAL_CTRL_REF     7D2<> 14AB<>
EC01              8B7> 9D7<>
EC02              8A7> 9D7<>
EC03              8B7> 9D7<>
EC04              8B7> 9D7<>
EC08              8A7> 13D3<>
EC09              8C5> 16C6<>
EC10              8B7> 16C6<>
EC11              8D7> 16C6<>
EC12              8C7> 16B6<>
EC13              8B4> 23A6<>
EC14              8C4> 23A6<>
EC15              8B5> 18D3<>
EC16              8B5> 18D3<>
EC17              8D5> 17B3<>
EC18              8B7> 17B3<>
EC19              8D4> 17B3<>
EC20              8C7> 17B3<>
EC21              8D4> 17B3<>
EC22              8A7> 17B3<>
EC23              8C8> 19C8<>
EC24              8A8> 19C8<>
EC25              8A8> 19C8<>
EC26              8A8> 19C8<>
EC27              8D8> 20C8<>
EC28              8B8> 20C8<>
EC29              8A8> 20C8<>
EC30              8B8> 20C8<>
EC31              8C5> 21C8<>
EC32              8B7> 21C8<>
EC33              8B8> 21C8<>
EC34              8C7> 21C8<>
EC35              8B5> 22C8<>
EC36              8D5> 22C8<>
EC37              8C8> 22C8<>
EC38              8C4> 22C8<>
EC39              8B4> 9A7<>
EC40              8C4> 9A7<>
EC41              8D4> 9A7<>
EC42              8A5> 9A7<>
EC43              8B5> 23A6<>
EC44              8B7> 23A6<>
EC45              8A7> 23A6<>
EC46              8C5> 23A6<>

EC47              8C5> 23A6<>
EC48              8A5> 23A6<>
EC49              8C5> 23A6<>
EC50              8B4> 23A6<>
EC51              8B5> 23A6<>
ECC_LE_SEL        7C2<> 14AB<>
FRZ_CLK           6C1> 23A6<
FRZ_DATA          6C1> 23A6<
GBL *             7B4<> 10A3<> 13C3>
HRESET *          6B1> 12C8<
IGN_PCI_AD31      6C3<> 15D8<
INT_60X *         7C2> 12C8<
INT_TO_664        6B1<> 15D8<
L1_TST_CLK *      7B1> 12A8<
L2_CLAIM *        6C3> 13B8<
L2_TST_CLK *      7B1> 12A8<
LSSD_MODE *       7B1> 12A8<
MA <11..0>        14D4> 6B6<
MCE <7..0> *      14D4> 6B6<
MCP_60X *         7C2> 12C8<
MD <63..0>        6D7<> 16D7<>
MDP <7..0>        6B5<> 16D7<>
MEM_BE <3..0>     7D1<> 14AB<>
MEM_DATA_OE *     7C2<> 14AB<>
MEM_RD_SMPL       7C2<> 14AB<>
MEM_WRL_OPEN      7C2<> 14AB<>
MRE <7..0> *      14D4> 6C6<
MWE0 *            14D4> 6B1<
MWE1 *            14D4> 6B1<
MWS_P2MRXS        6B1<> 14AB<>
NMI_FROM_ISABRDG  6B1<> 15DB<
OE_245_B          7B1> 17B3<
PCI_AD <31..0>    6B3<> 15B8<>
PCI_AD_OE *       7C2<> 14AB<>
PCI_C/BE <3..0> * 6C3<> 15C3<>
PCI_CLK_IN        6C3> 15D8<
PCI_DEVSEL *      6C3<> 15D3<>
PCI_EXT_SEL       7C2<> 14AB<>
PCI_FRAME_664 *   6C3<> 15D3<>
PCI_IRDY *        6C3<> 15D3<> 14C5<
PCI_LOCK *        6C3<> 15D8<
PCI_OL_OPEN       7C2<> 14AB<>
PCI_OUT_SEL       7C2<> 14AB<>
PCI_PAR           6B3<> 15AB<>
PCI_PERR *        6C3<> 15C3<>
PCI_SERR *        6C3<> 15C3<>
PCI_STOP *        6C3<> 15D3<>
PCI_TRDY *        6C3<> 15D3<> 14C5<
PCLK_60X          23D3> 7B4<
PD <2..0>         6C1> 7A4> 7C2> 19A7>
PLL_CFG <3..0>    6B1> 12B8<
POWER_GOOD/RESET * 6B1> 13CB<
QACK_60X *        6A1> 12A8<
QREQ_60X *        12A1> 6A1<
ROM_LOAD          7B1<> 13B3> 14A8<
ROM_OE *          13C3> 6B1<
ROM_WE *          13B3> 6B1<
RSRV_60X *        12C1> 7C2<
SHD *             13C3> 7B1<
SMI *             6C3> 12C8<
SRAM_ADS/ADDR0    7B1<> 13A3> 19C8< 20C8< 21C8< 22C8<

SRAM_ADSP *       6A1> 19C8< 20C8< 21C8< 22C8<
SRAM_ALE          7B1<> 13B3>
SRAM_BCLK0        23C3> 6C1<
SRAM_BCLK1        23C3> 6D3<

SRAM_BCLK2        23C3> 6D3<
SRAM_BCLK3        23C3> 6C3<
SRAM_CNT_EN/ADDR1 7B1<> 13A3> 19C8< 20C8< 21C8<
                  22C8<
SRAM_CS *         7A4> 19C8< 20C8< 21C8< 22C8<
SRAM_OE *         7B1<> 19C8< 20C8< 21C8< 22C8<
SRAM_WE *         7B1<> 13B3> 19C8< 20C8< 21C8< 22C8<

SRESET_60X *      6C1> 12C8<
TA *              7C2> 11D8<> 13C3<>
TAG_ADDR_13       7A4> 17C8<
TAG_A_IN          7B4>
TAG_BCLK          23C3> 7B4<
TAG_CLEAR *       7B1<> 13B3> 17B8<
TAG_CS1 *         6B1> 17A8<
TAG_CS2           7A4> 17A8<
TAG_DATA_11       7A4> 17B8<
TAG_DTY           17C4> 7A4<
TAG_MATCH         7B1<> 17A4> 13B8<
TAG_PWRDN *       7A4> 17A8<
TAG_SFUNC         6B1> 17B8<
TAG_TA *          17C4> 7A4<
TAG_VALID         7B1<> 13B3> 17C8<
TAG_VLD           17C4> 7A4<
TAG_WE *          7B1<> 13B3> 17A8<
TAG_WT            17C4> 7A4<
TAG_WT_DTY_IN     7A4> 17C8<
TAOE *            6A1> 17C8<
TA_GATES          7A4> 17C8<
TBEN_60X *        7C2> 12C8<
TBST *            10A3<> 13C8<> 7B4<
TC <1..0>         7A4<> 10A3>
TCK               6B1> 12B8<
TDI               6B1> 12B8<
TDO               12B1> 6B1<
TEA *             6A1> 11D8<> 13C3>
TLBISYNC *        7C2> 12B8<
TMS *             6B1> 12B8<
TRST *            6B1> 12B8<
TS *              7B4<> 10C4<> 13C8<>
TSIZ <2..0>       7C5<> 10A3<> 13C8<>
TT <4..0>         6A1<> 10A3<> 13B8<>
TT2_664           7B4<> 13C8<>
USER_PCICLK1      23A3> 7B4<
USER_PCICLK2      23B3> 7B4<
USER_PCICLK3      23B3> 7B4<
USER_PCICLK4      23B3> 6C6<
USER_PCICLK5      23B3> 6C6<
USER_PCICLK6      23B3> 7B4<
WT_60X *          10A3> 6A1<
XATS *            7B4<> 13C8<
XTAL1             6C6<> 23D6<>
XTAL2             6C6<> 23C6<>
X_663_CPU_CLK     6A6> 14C5<
X_664_CPU_CLK     6A6> 13D8<
X_CPU_RDL_OPEN    7C2> 14B1<
X_INT_60X *       13D3> 6A1<
X_MCP_60X *       13C3> 6C1<
X_PCLK_60X        6A6> 12C8<
X_SRAM_BCLK0      6C1> 19D8<
X_SRAM_BCLK1      6C1> 20D8<
X_SRAM_BCLK2      6A6> 21D8<
X_SRAM_BCLK3      6A6> 22D8<
X_SRAM_OE *       13B3> 7B6<
X_TAG_BCLK        6A6> 17D8<
```

IBM

SIZE	REVISION	DRWNG PART NUMBER
B		

*** Unit Cross-Reference ***
--- for the entire design --

C1	CAP_C4	9D5 9D5 9D5 9D5
C2	CAP_C4	9D4 9D4 9D4 9D4
C3	CAP_C4	9D3 9D3 9D3 9D3
C4	CAP_C4	9D2 9D2 9D2 9D2
C5	CAP_C4	9D1 9D1 9D1 9D1
C6	CAP_C4	9B5 9C5 9C5 9C5
C7	CAP_C4	9B4 9C4 9C4 9C4
C8	CAP_C4	9B3 9C3 9C3 9C3
C9	CAP_C4	9B2 9C2 9C2 9C2
C10	CAP_C4	9B1 9C1 9C1 9C1
C11	CAP_C4	9A3 9A3 9B3 9B3
C12	CAP_C4	9A2 9A2 9B2 9B2
C13	CAP_C4	9A1 9A1 9B1 9B1
J1	PIN1089	6A7 6C7 6D2 6D5 6D8 7C5 7C5 7D3 7D7 7D8 8C2 8D1 8D3 8D3 8D4 8D5 8D6 8D7 8D8
R1	RESCHIP	9A6 10B4 10B4 10B5 10B5 10B5 10B5 10B6 10B6 10B6 10B6 10B6 10D5 10D6 10D6 10D6 10D6 10D6 10D6 10D6 11C7 11C7 11C7 11C7 12D2 12D2 12D2 12D4 12D4 12D4 12D5 12D5 12D5 12D5 12D5 12D5 12D6 12D6 12D6 12D6 12D6 12D6 12D7 13D6 13D7 13D7 17D7 17D7 17D7 17D7 17D7 17D7
U1	C4_PPC603	9B4 9B5 9D6 10C8 11D6 12A3 12C3
U2	C4_KAUAI	9A8 13C5 13D1 14D7 15C4 15D1
U3	C4_LANAI	9C7 14C3 15D6 16B7 16B8 16C6 16D2 16D4
U4	C4_MPC970	23C4
U5	C4_TAG	17C6 18D2 18D4
U6	C4_LVT16245	17D2
U7	C4_KODIAK	18B8 18D8 19B2 19B6 19C3
U8	C4_KODIAK	18B7 18D7 20B2 20B6 20C4
U9	C4_KODIAK	18B6 18D6 21B1 21B6 21C3
U10	C4_KODIAK	18B5 18D5 22B2 22B6 22C3

IBM

SIZE	REVISION	DRWNG PART NUMBER
B		

SHEET 25 OF 26

THIS PAGE INTENTIONALLY LEFT BLANK

IBM

DRAWING

SIZE	REVISION	DRWNG PART NUMBER
B		

LAST_MODIFIED=Tue Apr 30 22:47:11 1996
ODYSSEY1
ODYSSEY1