NEC

VRC5074 System Controller

Data Sheet **June 1998**

1.0 Introduction

1.1 Overview

The VRC5074 System Controller is a software-configurable chip that directly connects the VR5000 CPU to SDRAM memory, a PCI Bus, and a Local Bus, without external logic or buffering. From the CPU's viewpoint, the controller acts as a memory controller, DMA controller, PCI-Bus host bridge, and Local-Bus host bridge. From the viewpoint of PCI agents, the controller acts as master and target on the PCI Bus. The controller also has one serial port and four timers.

1.2 Features

❑ **CPU Interface**

- Connects directly to a 250 MHz VR5000 CPU.
- 100 MHz CPU bus.
- Peak block-transfer throughput of 800 Mbytes/sec, maximum sustained throughput of 640 Mbytes/sec.
- 16 x 8-byte (128-byte) CPU-to-controller FIFO.
- Little-endian or big-endian byte order on CPU interface.
- Supports secondary cache.
- 15 interrupt sources, individually enabled and assigned to one of the CPU's seven interrupt inputs.
- Supports all CPU bus-cycle types (but the only write type is pipelined write). Parity generation and checking on CPU data cycles.
- Mode data at reset provided by a serial EEPROM or by the controller.
- 3.3V I/O.

❑ **Memory Interface**

- 100 MHz memory bus.
- Maximum sustained throughput of 800 Mbytes/sec.
- Supports three physical loads per data bit: two SDRAM physical banks and one other (e.g., EPROM, Flash, or buffers bridging to a secondary memory bus).
- Supports four types of SDRAM with two to four on-chip virtual banks: 256Mb four-bank, 64Mb four-bank, 64Mb two-bank, 16Mb two-bank.
- On-chip bank-interleaving buffers.
- Programmable address ranges for each memory bank.
- Memory may maintain multiple open SDRAM pages.
- Parity or ECC generation and checking of memory data cycles with 64+8 bits of SDRAM and no performance degradation.

- Read/write buffers:
  - 8-dword (64-byte) CPU Write FIFO.
  - 8-dword (64-byte) PCI Write FIFO.
- On-chip refresh generation.
- 3.3V I/O.

❑ **PCI Bus**

- Full compliance with *PCI Local Bus Specification, Revision 2.1*.
- Four possible configurations:
  - 66 MHz, 64-bit bus (maximum sustained bandwidth 533 Mbytes/sec)
  - 66 MHz, 32-bit bus (maximum sustained bandwidth 267 Mbytes/sec)
  - 33 MHz, 64-bit bus (maximum sustained bandwidth 267 Mbytes/sec)
  - 33 MHz, 32-bit bus (maximum sustained bandwidth 133 Mbytes/sec)
- PCI-Master support, allowing the CPU, DMA, and Local-Bus masters to access targets on the PCI Bus via two programmable PCI Address Windows.
- PCI-Target support, allowing PCI-Bus masters to access to all controller resources.
  - Eleven programmable Base Address Register (BAR) windows.
  - All reads are delayed transactions.
  - Up to four simultaneous delayed transactions.
- Master and target read/write bursts up to 2 Mbytes in length.
- Master and target read/write buffers:
  - 32-entry x 8-byte (256-byte) PCI Output FIFO.
  - 32-entry x 8-byte (256-byte) PCI Input FIFO.
  - 4-entry x 8-byte (32-byte) CPU Delayed Read Completion (DRC) Buffer.
  - 4-entry x 8-byte (32-byte) DMA Delayed Read Completion (DRC) Buffer.
- Optional PCI Central Resource functions:
  - Buffered PCI clock to 5 other PCI devices.
  - PCI clock can be external or derived from CPU clock.
  - Arbitration for the controller and 5 other PCI devices.
  - CPU interrupt control for 5 PCI devices.
- Full PCI Configuration Space.
- 64-bit addressing support for master and target using Dual Address Cycle (DAC).
- Locked cycle (exclusive access) support as master and target.
- Parity generation and checking on address and data cycles.
- Compliant with both 3.3V and 5V PCI signaling.

❑ **Local Bus**

- 25 MHz or 50 MHz bus (0.25 or 0.50 of system clock).
- Programmable chip-selects for 7 devices plus Boot ROM.
  - Each chip-select supports up to 4GB address space.
  - Devices may alternatively be located on the memory bus.
  - Chip-select signals may alternatively be used for DMA or UART control, or as general-purpose I/O signals.
- Support for burst cycles on the Local Bus.

- Support for Local-Bus master control of the Local Bus, using 68000 or Intel arbitration protocols.
- Programmable control-signal relationships and timing:
  - Timing can be fixed or use external Ready signal.
  - 12-bit timer for external Ready signal.
- 3.3V outputs, 5V-tolerant inputs

❑ **DMA**

- Two DMA channels.
- Block transfers to or from any physical address.
- Transfers initiated by the CPU, a PCI-Bus master, or a Local-Bus master.
- Peak block-transfer throughput of 800 Mbytes/sec, maximum sustained throughput of 640 Mbytes/sec.
- 32 x 8-byte (256-byte) DMA FIFO.
- Two sets of DMA control registers. One set can be programmed while the other performs a transfer.
- Chained transfers—when one transfer completes, another programmed transfer automatically begins.
- Supports bidirectional, unaligned transfers.
- Optional hardware handshake signals (REQ#, ACK#, EOT#) if certain chip-selects are not used.

❑ **Serial Port (UART)**

- Compatible with National Semiconductor’s PC16550D UART.
- Receiver and transmitter each have a 16-byte FIFO.
- 5, 6, 7, or 8 bits per character.
- Even, odd, or no parity-bit generation and detection.
- 1, 1.5, or 2 stop-bit generation.
- Baud-rate generator division of input clock by 1 to (2^{16} -1).
- Prioritized interrupt controls.
- DSR and DTR control signals.
- Optional hardware controls (CTS#, RTS#, DCD#, XIN#) if certain chip-selects are not used.

❑ **Timers**

- 16-bit SDRAM refresh timer.
- 24-bit CPU-bus read timer.
- 32-bit general-purpose timer.
- 32-bit watchdog timer.
- All timers are cascadable.

❑ **Multi-Controller Support**

Contents

2.0 Internal and System Architecture

2.1 Internal Architecture

There are three masters internal to the controller that can generate accesses:

- CPU
- PCI Bus
- DMA (which generates accesses on behalf of the CPU, PCI-Bus masters, or Local-Bus masters)

There are four targets internal to the controller that can respond to an access:

- Memory (SDRAM and other devices on the memory bus)
- PCI Bus
- Local Bus
- Controller’s Internal Registers (Table 8 on page 31)

There are independent, point-to-point buses, 64-bits wide in each direction, that connect all possible master-target pairs (except loop-back pairs):

- CPU-to-Memory
- CPU-to-PCI Bus
- CPU-to-Local Bus
- CPU-to-Controller’s Internal Registers
- DMA-to-Memory
- DMA-to-PCI Bus
- DMA-to-Local Bus
- DMA-to-Controller’s Internal Registers
- PCI Bus-to-Memory
- PCI Bus-to-Local Bus
- PCI Bus-to-Controller’s Internal Registers

Figure 1 shows these internal buses. If only one master accesses a given target, no resource contention occurs, so that accesses by all masters can proceed simultaneously to their separate targets. When multiple masters attempt to access a given target, the controller arbitrates as follows:

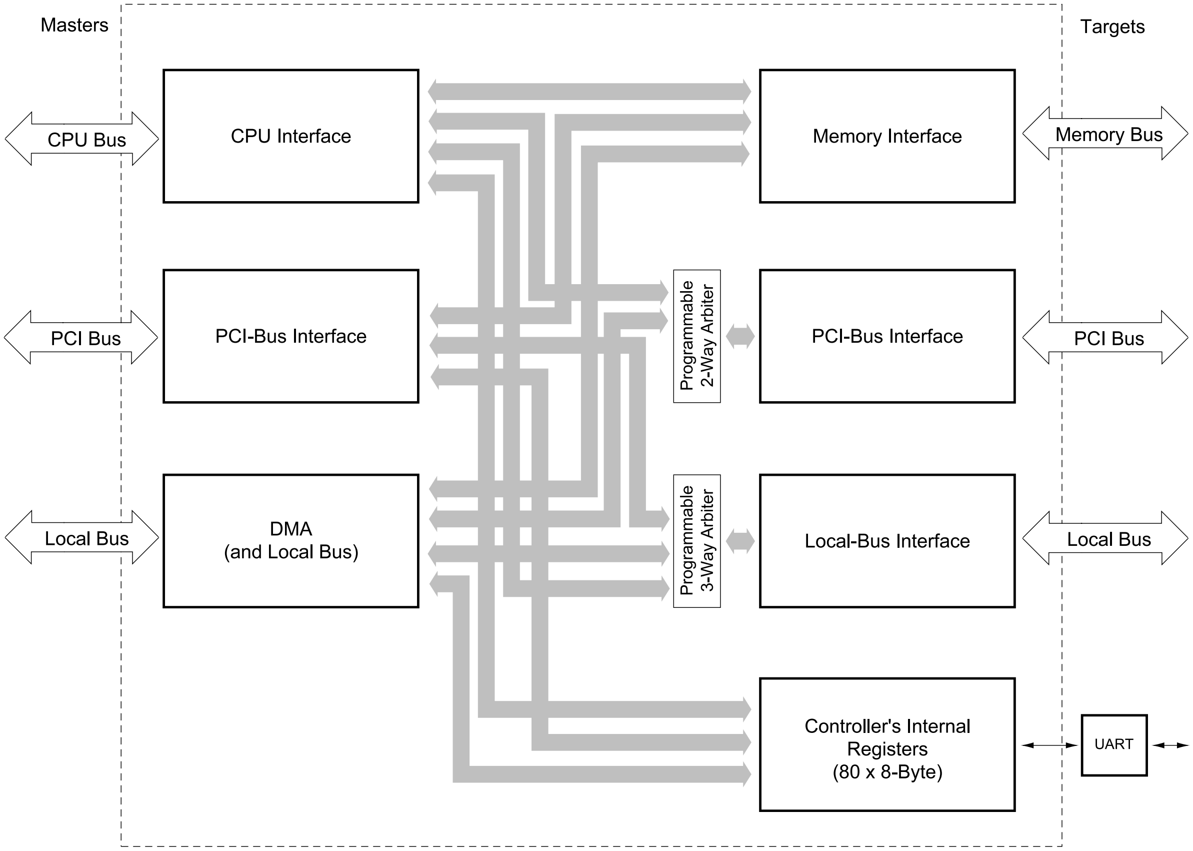
When the Controller’s Internal Registers are targeted by multiple masters simultaneously, the arbitration is very fast, because the registers run so quickly. The longest delay any master is likely to see is only a few clocks.

The Memory target also responds very fast when targeted by multiple masters simultaneously. It attempts to service all requests in the most efficient manner, for example by giving priority to requests for a page that is currently open. SDRAM has such high bandwidth that it is unlikely for any one master to be held off for more than a few clocks.

The PCI-Bus target has an arbiter for responding to simultaneous accesses by the CPU and DMA. The arbiter is controlled by programmable fields that govern the duration of consecutive accesses by these masters.

The Local-Bus target, like the PCI-Bus target, has a programmable arbiter that governs the duration of consecutive accesses by the CPU, DMA and PCI masters.

Figure 1: VRC5074 Internal Architecture



5074-066.eps

2.2 System-Design Options

Several signals are sampled at reset (Section 12.0) to determine the properties of the controller's operation in a system, including:

- *Endian Mode:* The CPU interface can operate in either little-endian or big-endian mode. However, the memory, PCI-Bus, and Local-Bus interfaces always operate in little-endian mode.
- *PCI-Bus and Local-Bus Width:* The controller can support either a 64-bit PCI Bus and no Local Bus, or a 32-bit PCI Bus and a 32-bit Local Bus.
- *PCI Central Resource Functions:* The controller can operate either as the PCI Central Resource or it can operate in a PCI Stand-Alone Mode (i.e., not the Central Resource).
- *Multi-Controller Configurations:* When multiple controllers are used in a system, each has its own ID and address space, and one controller is the Main Controller.

Figure 2 through Figure 7 show examples of how the controller can be used in system designs.

Figure 2: Single-Controller, 32-Bit PCI-Bus Configuration

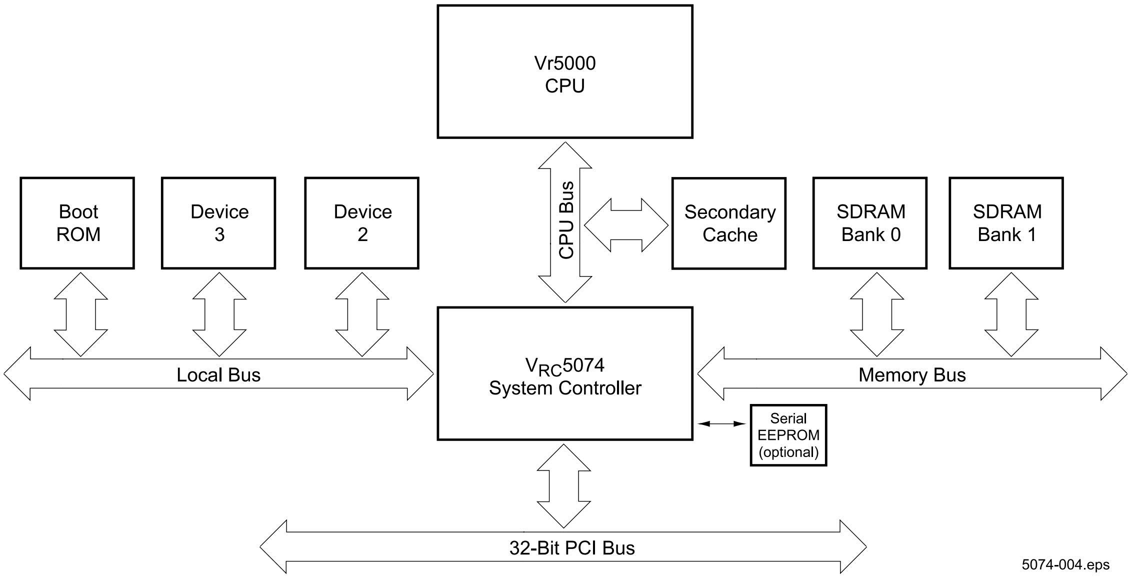


Figure 2 shows a system in which the controller supports two physical banks of SDRAM memory, a 32-bit Local Bus with Boot ROM and two other devices, and a 32-bit PCI Bus. If the CPU and controller shown here are the main CPU and the main PCI controller in the system, the controller can perform all (or any) of the PCI Central Resource functions for other PCI devices, and the CPU can run the PCI Configuration Space cycles for all PCI devices in the system.

If the VR5000 CPU has a secondary cache, the controller monitors cache hits. An optional Serial EEPROM provides mode data to the CPU at reset. If the EEPROM is not used, the controller itself can configure the CPU with a default mode sequence.

Figure 3: Single-Controller, 64-Bit PCI-Bus Configuration With Memory-Bus Buffer

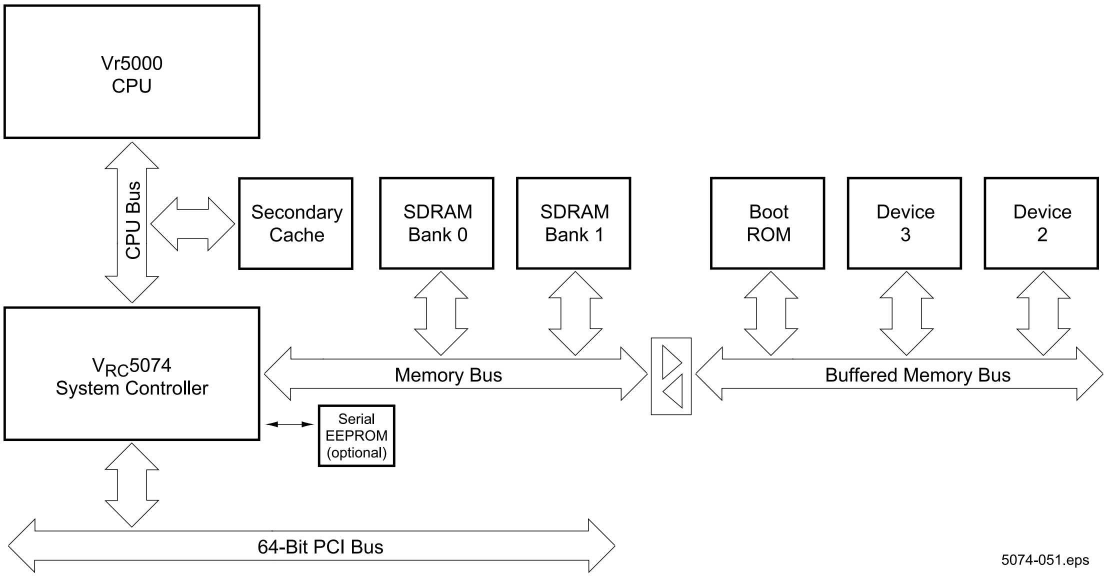


Figure 3 shows a system in which the controller supports the maximum of three physical loads on the memory bus—two physical banks of SDRAM memory plus one row of transceivers, which in turn support additional devices. Signals that were used in Figure 2 for a the 32-bit Local Bus are configured here to be the high address and data bits for a 64-bit PCI Bus.

Only the address and data signals to non-SDRAM loads on the memory bus need to be buffered. The chip-selects for these devices need not be buffered, because each of these bits supports only a single load.

Figure 4: Single-Controller, 64-Bit PCI-Bus Configuration

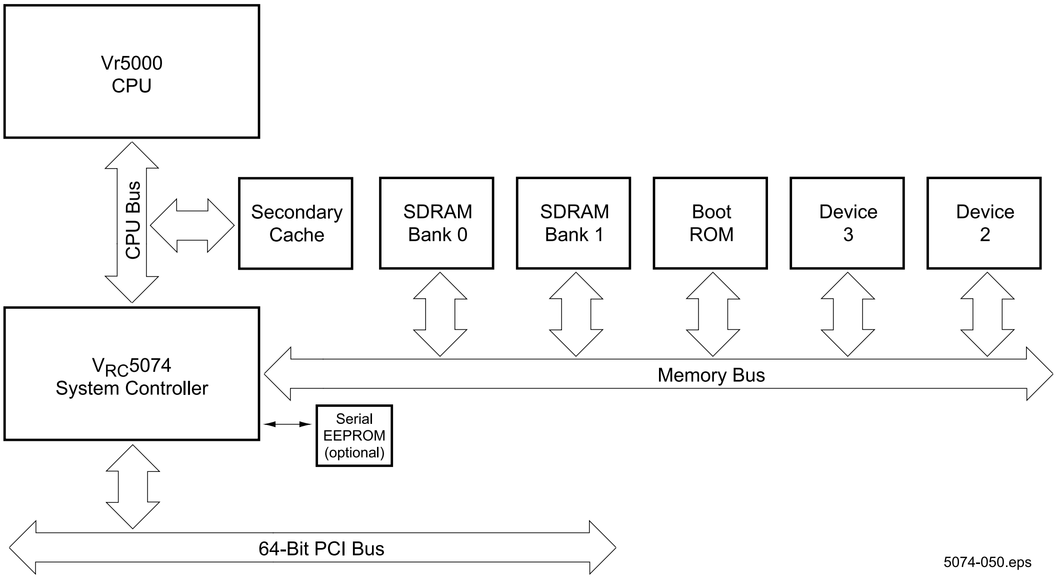
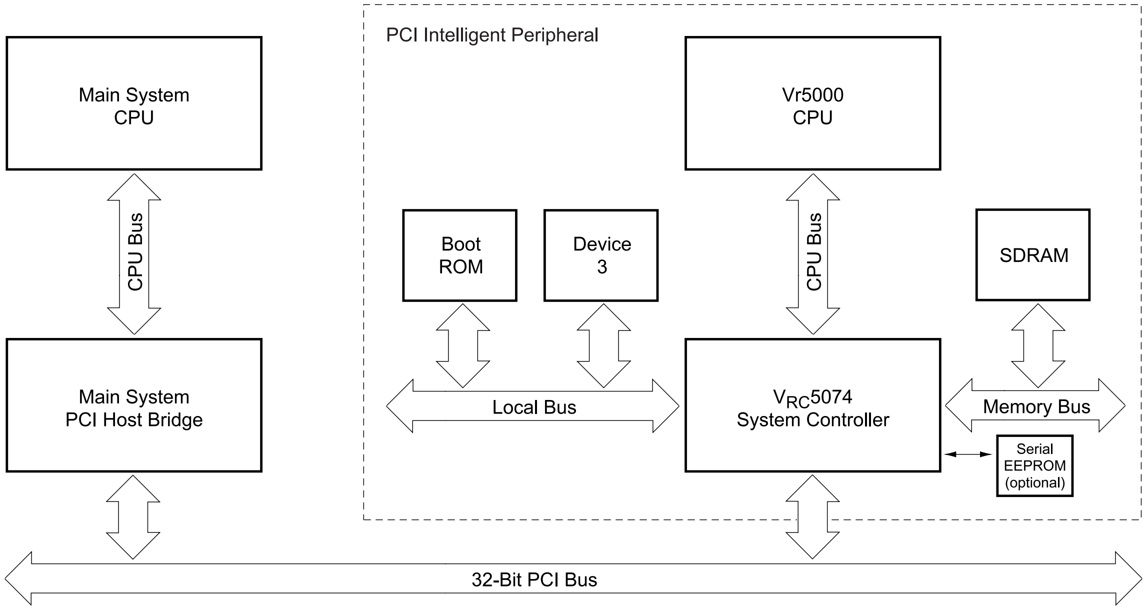


Figure 4 is similar to Figure 3, but shows a system in which the controller supports more than the maximum of three physical loads on the memory bus. If more than three loads are placed on the memory bus, the bus will slow down. Such configurations require either a CPU SysClock slower than 100 MHz or buffering on the memory bus, as is done in Figure 3.

Figure 5: Intelligent PCI Peripheral Configuration (Stand-Alone Mode)



5074-052.eps

Figure 5 shows a system in which a VRC5074 controller is an intelligent peripheral to a Main CPU and its associated PCI host bridge. The VRC5074 controller is on a PCI board with direct connection to its own VR5000 CPU, supporting one or two physical banks of SDRAM on the memory bus plus up to eight other devices on the Local Bus. The daughter board connects to the main system controller over a 32-bit PCI Bus. Accesses via the Main Controller to its resources can proceed simultaneously with accesses via the VRC5074 controller to its resources, except when two PCI Bus masters attempt to access the same resource simultaneously via the shared PCI Bus.

In such a system, the main system controller would typically act as the PCI Central Resource, and the main system CPU would run the PCI Configuration Space cycles for all PCI devices in the system. This is called a *stand-alone* configuration because the VRC5074 controller does not perform the PCI Central Resource functions.

Figure 6: PCI Peripheral Configuration With No CPU (Stand-Alone Mode)

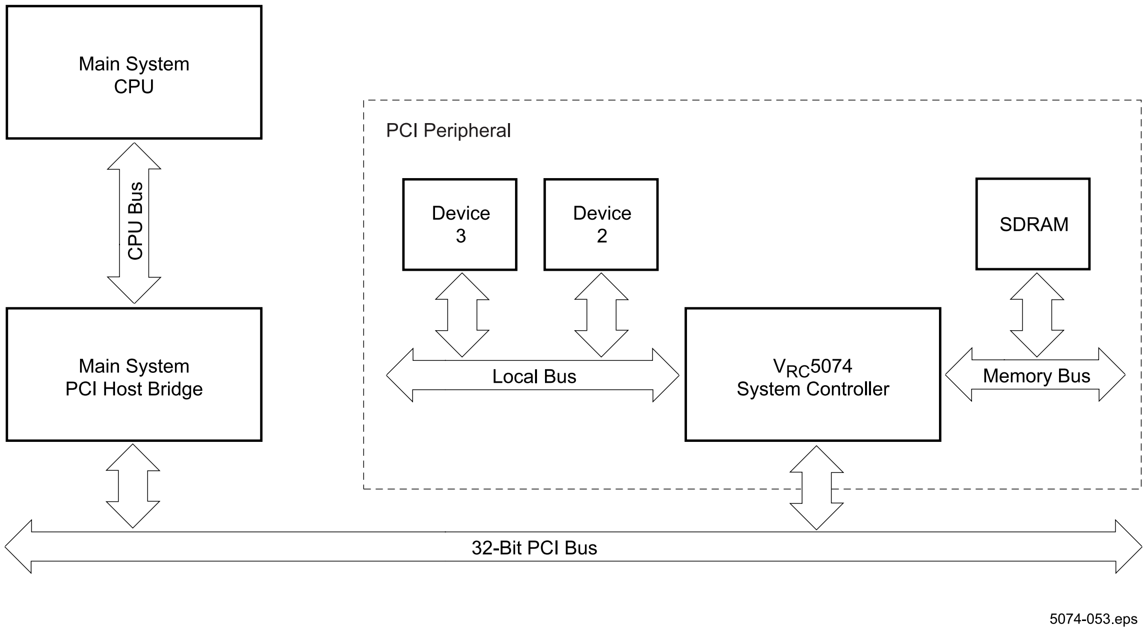


Figure 6 shows a system in which the controller is placed on a PCI daughter board without its own CPU. As in Figure 5, the controller supports one or two physical banks of SDRAM on the memory bus plus up to eight other devices on the Local Bus. The daughter board connects to the main system controller over a 32-bit PCI Bus.

Here again, the main system controller acts as the PCI Central Resource, and the main system CPU runs the PCI Configuration Space cycles for all PCI devices in the system. This is also called a *stand-alone* configuration, because the V_{RC}5074 controller does not provide the PCI Central Resource functions.

Figure 7: Multi-Controller Configuration With Dual PCI Buses

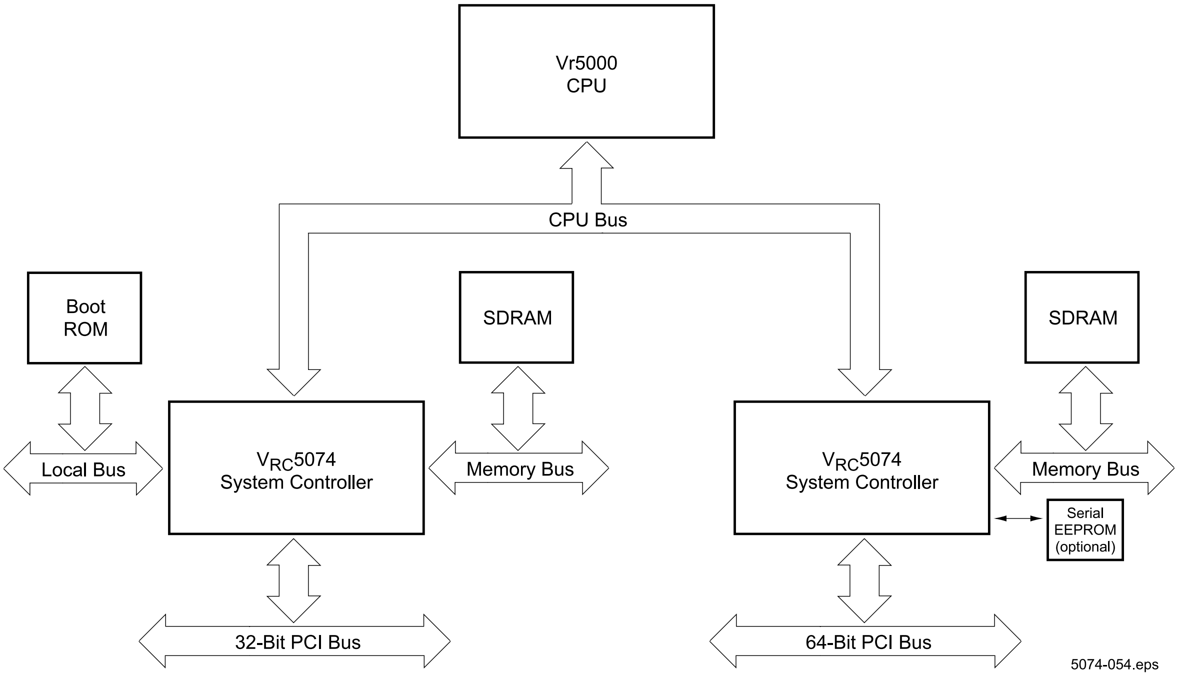


Figure 7 shows a system in which one VR5000 CPU is attached directly to two VRC5074 controllers. In this example, one controller is configured to support a 32-bit PCI Bus while the other controller supports a second, separate, 64-bit PCI Bus. Either controller can support one or two physical banks of SDRAM, and the controller supporting the 32-bit PCI Bus can have up to seven other devices on its Local Bus. A similar multi-configuration could be used to attach one or more VRC5074 controllers and one or more ASICs to a single CPU.

If the VR5000 CPU is the main system CPU, it would run the PCI Configuration Space cycles for all PCI devices in the system, and each of the two VRC5074 controllers would provide PCI Central Resource functions for its associated PCI Bus.

2.3 Terminology

- # as a suffix on a signal name means active-Low. Signals without this suffix are active-High.
- *0x* means a hexadecimal number.
- *assert* means to drive a signal to its active state (active-Low or active-High).
- *b* means bit, or a binary number.
- *B* means byte.
- *controller* means the VRC5074 System Controller.
- *dword* or *doubleword* means 8 bytes. This definition is MIPS-compatible and differs from the *PCI Local Bus Specification*, where a dword is 4 bytes.
- *external agent* means any logic device directly connected to the CPU that supports CPU requests.
- *external device* means any logic device, other than the CPU, that is connected to the controller.
- *flushed* is not used, because it is an ambiguous term (it means either write-back or discard).
- *h* means a hexadecimal nibble.
- *Local Bus* means the controller's Local Bus, not the PCI Local Bus.
- *Main Controller* means the controller directly connected to the main CPU in a system. Only the Main Controller should run PCI Configuration Space cycles.
- *Mb* means megabit.
- *MB* means megabyte.
- *memory* (unless otherwise modified) means memory attached to the controller.
- *module* means a set of chips, as in a SIMM or DIMM.
- *n* means an integer.
- *negate* means to drive a signal to its inactive state. See *assert*, above.
- *PCI Stand-Alone Mode* means the controller's operating mode when it is not providing the PCI Central Resource functions for the system.
- *qword* or *quadword* means 16 bytes. This definition is MIPS-compatible and differs from the *PCI Local Bus Specification*, where a qword is 8 bytes.
- *SDRAM* means synchronous DRAM.
- *word* means 4 bytes. This definition is MIPS-compatible and differs from the *PCI Local Bus Specification*, where a *word* is 2 bytes.

2.4 Reference Documents

The following documents form a part of this data sheet.

- *Vr5000 Microprocessor User's Manual, Revision 1.1* (NEC Electronics, Inc., Document No. U11323EU1V0UM00).
- *Vr5000 Bus Interface User's Manual, Revision 1.1* (NEC Electronics, Inc., Document No. U11322EU1V0UM00).
- *CB-C9 Multiplying Asynchronous PLL (APLL) Data Sheet, Preliminary, November 1996* (NEC Electronics, Inc.).

- *CB-C9 ASIC Family 0.35 Micron Standard Cell Specification Version 1.1a Analog PLL for Clock Skew Control - AAPLNIL, Preliminary* (NEC Electronics, Inc.).
- *CB-C8VX/VM ASIC Family 0.5 micron Standard Cell User's Manual, Mega Function NY16550L UART, Preliminary, 4 October 1996* (NEC Electronics, Inc.).
- *DDB-V$_{RC}$5074 Single Board Computer Specification* (NEC Electronics, Inc.).
- *PCI Local Bus Specification, Revision 2.1* (Peripheral Component Interconnect Special Interest Group).

3.0 Signal Summary

The controller has 350 signals, 124 power or ground pins, and 26 no-connects or blank pins, for a total of 500 pins. Table 1 through Table 6 summarize signal functions. An “#” suffix on a signal name means active-Low. The pinouts are shown in Section 17.0 on page 197.

Table 1: CPU-Bus Signals

Signal	I/O	5V Tolerant	Reset Value	Pullup/ Pulldown	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
BigEndian	I/O	No	HiZ		0	25	3	**Endian Mode**. This signal is normally an input, just as it is to the CPU. It specifies the endian mode of t he CPU interface (big-endian = High, little-endian = Low). The input from this signal is ORed with the Endian Bit (EB) of the Serial Mode EEPROM sequence to specify the CPU’s endian mode (Section 12.4.2). As an output, this signal is the chip-select for the Serial Mode EEPROM (Section 12.4.1). The signal is also an output during the wiggle-mode test (Section 15.0).
CntrValid#	I/O	No	HiZ	external pullup	100	50	6	**Controller Output Valid**. Output from the controller indicating valid information on SysAD bus, except that it is an input in multi-controller configurations (Section 5.3.4). The signal connects to the ValidIn# signal on the CPU.
CntrVccOk	O	No	Low		0	50	6	**Controller Vcc OK**. Output from the controller initialization logic, indicating that the CPU can read the initialization (mode) bits. CntrVccOk is held low by VccOk until the controller initialization logic has read the Serial Mode EEPROM (Section 12.4.2).
ColdReset#	O	No	Low		0	50	6	**Cold Reset**. Asserted when VccOk is negated or on a software cold reset (Section 5.5.1). Negated synchronously with SysClock, 64K clocks after CntrVccOk is asserted.
CPUValid#	I	No	HiZ					**CPU Output Valid**. Input from the CPU indicating valid information on SysAD bus. This signal connects to the ValidOut# signal on the CPU.
Int#[5:0]	O	No	HiZ	external pullup	100	50	6	**Maskable Interrupts**. Controller interrupts to CPU.
MCWrRdy#	O	No	High		50	50	6	**Multi-Controller Write Ready**. Output from controller indicating when it can accept a CPU write. The signal is used only in multi-controller configurations (Section 5.3.4).
ModeClock	I	Yes						**Mode Clock**. SysClock divided by 256. Provided by the CPU (Section 12.4.2).
ModeOut	O	Yes	High				6	**Mode Data**. Serial boot-mode data for CPU initialization. The data is generated by the controller, or it is generated from a Serial Mode EEPROM and monitored and corrected by the controller (Section 12.4). This signal connects to the ModeIn signal on the CPU.

Table 1: CPU-Bus Signals (continued)

Signal	**I/O**	**5V Tolerant**	**Reset Value**	**Pullup/ Pulldown**	**Toggle Rate (MHz)**	**AC Load (pF)**	**DC Drive (mA)**	**Description**
NMI#	O	No	HiZ	external pullup	100	50	6	**Non-Maskable Interrupt**. Controller non-maskable interrupt to CPU.
PROM_CLK	O	Yes	Low		5	50	6	**PROM Clock**. Output clock to the Serial Mode EEPROM (Section 12.4.2).
PROM_SD	I/O	Yes	HiZ	external pullup	5	50	6	**PROM Serial Data**. The controller drives address and commands out and receives CPU serial boot-mode data in on this signal, which is connected to the Serial Mode EEPROM (Section 12.4.2). The signal must be pulled up if the Serial Mode EEPROM is not implemented.
Reset#	O	No	Low		0	50	6	**Reset**. Asserted when VccOk is negated or on programmed warm reset. Negated either 64 clocks after ColdReset# is negated for power-up and cold resets, or 64 clocks after being asserted due to a warm reset (Section 5.5.1).
ScDOE#	O	No	HiZ	external pulldown	50	50	6	**Secondary-Cache Data Output Enable**. The controller negates ScDOE# during cache misses, when the controller is providing data to the CPU, and asserts ScDOE# to indicate that it will supply the last dword of a read response in the next clock.
ScMatch	I	No						**Secondary-Cache Match**. Hit/miss indication from secondary cache for current request. Valid two clocks after the address is driven.
ScWord[1:0]	I/O	No			50	50	6	**Secondary-Cache Word**. Doubleword offset within the secondary cache line.
SysAD[63:0]	I/O	No	HiZ		100	50	6	**System Address and Data**. System multiplexed address/data bus. The controller uses the SysAD[63:0] bits and the SysCmd[2:0] bits to internally generate byte enables, per Table 4.14 of the *V_R5000 Bus Interface User's Manual*.
SysADC[7:0]	I/O	No	HiZ		100	50	6	**System Address and Data Check**. System address/data check bus (one even-parity bit per SysAD byte).
SysClock	I	No						**System Clock**. The controller has an internal phase-locked loop (PLL) attached to SysClock.
SysCmd[8:0]	I/O	No	HiZ		100	50	6	**System Command**. The command and or data-type for the current bus cycle.
VccOk	I	Yes						**Vcc OK**. Input from external analog circuit indicating that power to the CPU and controller has been above 3.135 volts for more than 100 milliseconds. The assertion of this signal begins the initialization sequence.
WrRdy#	I/O	No	High		50	50	6	**Write Ready**. Output from the controller indicating when it can accept a CPU write, except that it is an input in multi-controller configurations (Section 5.3.4).

Table 2: Memory-Bus Signals

Signal	I/O	5V Tolerant	Reset Value	Pullup/ down	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
BootCS#	O	Yes	High		5	50	6	**Boot Memory Chip-Select.** The device corresponding to this chip-select may be located either on the Local Bus or the Memory Bus, as specified in the MEM/LOC bit of the BOOTCS Physical Device Address Register (PDARs, Section 5.4). This signal is also listed in Table 4.
DQM	O	No	High		0	50	12	**Data Qualifier Mask**. SDRAM chip data I/O qualifier mask.
MAbank0[14:0]	O	No	Low		100	50	12	**Memory Address, Bank 0**. Multiplexed row/ column address for memory bank 0 (even bank).
MAbank1[14:0]	O	No	Low		100	50	12	**Memory Address, Bank 1**. Multiplexed row/ column address for memory bank 1 (odd bank).
MCAS#[1:0]	O	No	High		100	50	12	**Memory Column Address Strobes**. These signals are for physical banks 1 and 0, respectively, and are logically distinct.
MDC[7:0]	I/O	No	HiZ		100	50	6	**Memory Data Check**. Even-parity or ECC syndrome bits for MD[63:0].
MCS#[1:0]	O	No	High		100	50	12	**Memory Chip-Selects**. These signals are for physical banks 1 and 0, respectively, and are logically distinct.
MD[63:0]	I/O	No	HiZ		100	50	6	**Memory Data**.
MRAS#[1:0]	O	No	High		100	50	12	**Memory Row Address Strobes**. These signals are for physical banks 1 and 0, respectively, and are logically distinct.
MRDY#	I	Yes						**Memory Ready**. Access-ready timing for non-SDRAM memory (such as Flash). The timing associated with such devices can, alternatively, be specified in the Memory Access Timing Register (ACSTIME), as described in Section 6.6.2.
MWE#[1:0]	O	Yes	High		100	50	12	**Memory Write-Enables**. These signals are for physical banks 1 and 0, respectively, and are logically distinct.

Table 3: PCI-Bus Signals

Signal	I/O	5V Tolerant	Reset Value	Pullup/ down	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
ACK64#	I/O	Same as LOC_CLK in Table 4.						**PCI Acknowledge 64-Bit Transfer**. Asserted by the controller when it is ready to drive data as a target. This signal is carried on the LOC_CLK pin when PCI64# is asserted.
C/BE#[3:0]	I/O	Yes			33 or 66	50	12	**PCI Command and Byte-Enables**. During the address phase of a transaction, the signals carry the bus command. During the data phase, they carry byte-enables for the data on PCI_AD[31:0].
C/BE#[7:4]	I/O	Same as LOC_A[3:0] in Table 4.						**PCI Command and Byte-Enables** (64-bit). These signals are carried on the LOC_A[4:0] pins when PCI64# is asserted.
DEVSEL#	I/O	Yes		external pullup	10		12	**PCI Device Select**. Asserted by the controller to indicate that it is the target of the current access. Sampled by the controller to determine whether any device is responding to the current access.
FRAME#	I/O	Yes		external pullup	10		12	**PCI Cycle Frame**. Asserted by the controller as master to indicate the duration of an access. Sampled by the controller to determine the duration of an access.
GNT#[4:0]	I/O	Yes			2		12	**PCI Bus Grant**. Asserted by the controller as PCI Central Resource (PCICR# asserted) to indicate that a requesting device may control the PCI Bus. In Stand-Alone Mode, (PCICR# negated and GNT#[4:1] are unused inputs) GNT#[0] is sampled by the controller to determine if it has been granted its request on REQ#[0] for control of the PCI Bus.
IDSEL	I	Yes						**PCI Initialization Device Select**. Selects the controller as the target for Configuration Read and Write transactions. During Central Resource operation (PCICR# asserted), IDSEL outputs may be provided by resistively coupling to PCI_AD[31:16] signals. See section 3.7.4. of the PCI Local Bus Specification, Revision 2.1.
INTA#	I/O	Yes		external pullup	0		12	**PCI Interrupt A**. INTA# is an output if PCICR# is negated. INTA# is never driven High (pseudo open-drain). See Section 5.5.2 and Section 5.5.3 for interrupt prioritization and enabling.
INTB#	I	Yes		external pullup				**PCI Interrupt B**. See Section 5.5.2 and Section 5.5.3 for interrupt prioritization and enabling.
INTC#	I	Yes		external pullup				**PCI Interrupt C**. See Section 5.5.2 and Section 5.5.3 for interrupt prioritization and enabling.
INTD#	I	Yes		external pullup				**PCI Interrupt D**. See Section 5.5.2 and Section 5.5.3 for interrupt prioritization and enabling.
INTE#	I	Yes		external pullup				**Auxiliary Interrupt**. See Section 5.5.2 and Section 5.5.3 for interrupt prioritization and enabling.
IRDY#	I/O	Yes		external pullup	10		12	**PCI Initiator Ready**. Asserted by the controller as master to indicate that it is driving valid data on a write, or that it is prepared to accept data on a read. Sampled by the controller in conjunction with TRDY#.

Table 3: PCI-Bus Signals (continued)

Signal	I/O	5V Tolerant	Reset Value	Pullup/ down	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
LOCK#	I/O	Yes		external pullup	10		12	**PCI Exclusive Access**. Indicates an atomic operation that may take multiple bus transactions to complete.
M66EN	I	Yes						**PCI 66 MHz Enable**. Enables 66 MHz operation of the PCI Bus. When M66EN is asserted, all devices on the PCI Bus must run at 66 MHz.
PCI_AD[31:0]	I/O	Yes			33 or 66		12	**PCI Multiplexed Address and Data**.
PCI_AD[63:32]		Same as LOC_AD[31:0] in Table 4.						**PCI Multiplexed Address and Data** (64-bit). If PCI64# is asserted, bits 63:32 of the PCI Bus are carried on the LOC_AD[31:0] pins.
PAR	I/O	Yes			33 or 66		12	**PCI Parity**. The even-parity bit for PCI_AD[31:0] and C/BE#[3:0].
PAR64	I/O	Same as LOC_A[4] in Table 4.						**PCI Parity** (64-bit). The even-parity bit for PCI_AD[63:32] and C/BE#[7:4]. Only valid when PCI64# is asserted.
PCI64#	I	Yes						**PCI 64-Bit**. When PCI64# is asserted, 64-bit PCI-Bus operation is enabled and Local Bus operation is disabled. (Section 7.6 and Section 8.5). In this case: • the LOC_AD[31:0] pins carry the PCI_AD[63:32] signals. • the LOC_A[3:0] pins carry the C/BE#[7:4] signals. • the LOC_A[4] pin carries the PAR64 signal. • the LOC_ALE pin carries the REQ64# signal. • the LOC_CLK pin carries the ACK64# signal. PCI64# is a static signal and must be valid and unchanging during and after reset.
PCICR#	I	Yes						**PCI Central Resource**. Identifies the controller as the PCI Central Resource (Section 7.8). If PCICR# is asserted: • PCLK[4:0] are all outputs. • REQ#[4:0] are all inputs. • GNT#[4:0] are all outputs. • INTA# is an input. • PCIRST# is an output. • The controller configures 64-bit PCI operation with its REQ64# output. • The controller generates PCI Configuration Space cycles. **PCICR**# is a static signal and must be valid and unchanging during and after reset. See Section 7.8 for details.
PCIRST#	I/O	Yes			0		12	**PCI Reset**. PCIRST# is an input, except that it is an output if PCICR# is asserted. See Section 12.3 for details on PCIRST# during reset.

Table 3: PCI-Bus Signals (continued)

Signal	I/O	5V Tolerant	Reset Value	Pullup/ down	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
PCLK[4:0]	I/O	Yes			133		12	**PCI Clock**. The maximum frequency can be 66 MHz. When PCICR# is negated, PCLK[0] is an input and PCLK[4:1] are floated. When PCICR# is asserted, PCLK[4:0] are all outputs. The controller always uses PCLK[0] as its PCI-Bus clock.
PCLKIN	I	Yes						**PCI Clock Input**. External input for PCLK[4:0].
PERR#	I/O	Yes		external pullup	0		12	**PCI Parity Error**. Reports even-parity data errors across the PCI_AD[31:0], C/BE#[3:0] and PAR signals, or across the PCI_AD[63:32], C/BE#[7:4], and PAR64 signals.
REQ#[4:0]	I/O	Yes			5		12	**PCI Bus Request**. Sampled by the controller as PCI Central Resource to determine if a PCI device wishes to control the PCI Bus. In Stand-Alone Mode, the controller asserts REQ#[0] to request control of the PCI Bus, and REQ#[4:1] are unused inputs. Compare the description of GNT#[4:0].
REQ64#	I/O	Same as LOC_ALE in Table 4.						**PCI 64-Bit Request**. Asserted by the controller when it is ready to drive data as a master. This signal is carried on the LOC_ALE pin when PCI64# is asserted.
SERR#	I/O	Yes		external pullup	0		12	**PCI System Error**. Reports even-parity address errors on PCI_AD[31:0], C/BE#[3:0] and PAR, or on PCI_AD[63:32], C/BE#[7:4] or PAR64; data errors on the Special Cycle command; or any other catastrophic system error. SERR# is never driven High (pseudo open-drain).
STOP#	I/O	Yes		external pullup	10		12	**PCI Stop**. Asserted by the controller as target to request that a transfer be stopped. Sampled by the controller in conjunction with TRDY#
TRDY#	I/O	Yes		external pullup	10		12	**PCI Target Ready**. Asserted by the controller as target to indicate that it is driving valid data on a read, or that it is prepared to accept data on a write. Sampled by the controller in conjunction with IRDY#.

Table 4: Local-Bus Signals

Signal	I/O	5V Tolerant	Reset Value	Pullup/ down	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
BootCS#	O	Same as BootCS# in Table 2.						**Boot Memory Chip-Select.** The device corresponding to this chip-select may be located either on the Local Bus or the Memory Bus, as specified in the MEM/LOC bit of the BOOTCS Physical Device Address Register (PDAR), Section 5.4.
DCS#[8:2]	I/O	Yes	HiZ		10	50	6	**Device Chip-Selects** (or other functions). The devices corresponding to these chip-selects may be located either on the Local Bus or the Memory Bus, as specified in the MEM/LOC bit of the corresponding Physical Device Address Register (PDAR), Section 5.4. After reset, software can configure the DCS#[8:2] signals as follows: • DCS#[2] = UART_RTS# (active-low output) or general-purpose I/O. • DCS#[3] = UART_CTS# (active-low input) or general-purpose I/O. • DCS#[4] = UART_DCD# (active-low input) or general-purpose I/O. • DCS#[5] = UART_XIN (clock input) or general-purpose I/O. • DCS#[6] = DMA_ACK# (active-low output) or general-purpose I/O. • DCS#[7] = DMA_REQ# (active-low input) or general-purpose I/O. • DCS#[8] = DMA_EOT# (active-low input) or general-purpose I/O. See the following sections for details: • Additional UART Signals, Section 10.2. • DMA Hardware Handshaking, Section 9.4. • Device Chip-Select Function Register (DCSFN), Section 8.6.3. • Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4.
LOC_A[4:0]	I/O	Yes	Low		66	50	12	**Local-Bus Byte-Enables and Low-Address Bits** (or other functions). During the first clock of a Local-Bus cycle, LOC_A[3:0] carry active-low byte-enables (in effect, BE#[3:0] for the Local Bus). During the remainder of a non-block bus cycle, LOC_A[4:0] carry the five low-address bits (the same bits that were carried on LOC_AD[4:0] bits when LOC_ALE was active). The function of the LOC_A[4:0] signals changes when a Local-Bus master takes control of the Local Bus (See Section 8.4.1). If PCI64# is asserted: • LOC_A[3:0] = PCI-Bus C/BE#[7:4]. • LOC_A[4] = PCI-Bus PAR64. See Section 8.5 for details.

Table 4: Local-Bus Signals (continued)

Signal	I/O	5V Tolerant	Reset Value	Pullup/ down	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
LOC_AD[31:0]	I/O	Yes	HiZ		66	50	12	**Local-Bus Address and Data** (or other functions)**.** Local 32-bit multiplexed address/ data bus. If PCI64# is asserted: • LOC_AD[31:0] = PCI-Bus PCI_AD[63:32]. See Section 8.5 for details.
LOC_ALE	I/O	Yes	Low		66	50	12	**Local-Bus Address Latch Enable** (or other function). Asserted in the same clock as the access. If PCI64# is asserted: • LOC_ALE = PCI-Bus REQ64#. See Section 8.5 for details.
LOC_BG# or **HLDA**	O	Yes	High		5	50	6	**Local-Bus Grant** (or other function). Indicates that the controller has relinquished the Local Bus to a requesting master on the Local Bus. This signal becomes HLDA in Intel bus-arbitration mode (Section 8.6.1).
LOC_BGACK#	I	Yes						**Local-Bus Grant Acknowledge**. Indicates that an Local-Bus master has taken control of the Local Bus.
LOC_BR# or **HOLD**	I	Yes		external pullup				**Local-Bus Request** (or other function). Asserted by a Local-Bus master to request control of the Local Bus. This signal becomes HOLD in Intel bus-arbitration mode (Section 8.6.1). LOC_BR# may require an external pullup, depending on the application.
LOC_CLK	O	Yes	High		100	50	12	**Local-Bus Clock** (or other function). Generated by controller. The frequency is SysClock divided by 4 or by 2. If PCI64# is asserted: • LOC_CLK = PCI-Bus ACK64#. See Section 8.5 for details.
LOC_FR#	I/O	Yes	High		10	50	6	**Local-Bus Frame**. Indicates that a Local Bus cycle is taking place.
LOC_RD#	I/O	Yes	High		10	50	6	**Local-Bus Read**. Used for Local Bus devices that implement separate read and write control signals.
LOC_RDY#	I/O	Yes	High	external pullup	10	50	6	**Local-Bus Ready**. Acknowledge signal for devices on the Local Bus that do not respond in a fixed amount of time, as specified in the Local Bus Chip-Select Timing Registers (LCSTn), Section 8.6.2. LOC_RDY# may require an external pullup, depending on the application.
LOC_WR#	I/O	Yes	High		10	50	6	**Local-Bus Write or Read**. Used as Write, along with LOC_RD#, for devices that implement separate read and write control signals. Used as Write/Read# for devices that implement a single write/read control signal.

Table 5: DMA Hardware-Handshake Signals

Signal	I/O	5V Tolerant	Reset Value	Pullup/ down	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
DMA_ACK#	O	Same as DSC#[6] Table 4.						**DMA Acknowledge**. Used by the controller to acknowledge a DMA transfer request from an external device. If DMA_ACK# is used, however, DMA_REQ# must also be used. This signal can be implemented by software, after reset, as an alternative to DSC#[6] signal. See Section 9.4 for details.
DMA_REQ#	I	Same as DSC#[7] Table 4.						**DMA Request**. Used by an external device to request a DMA transfer. This signal can be implemented by software, after reset, as an alternative to DSC#[7] signal. See Section 9.4 for details.
DMA_EOT#	I	Same as DSC#[8] Table 4.						**DMA End of Transfer**. Used by an external device to abort a DMA transfer, but only if the DMA source is doing the handshaking. This signal can be implemented by software, after reset, as an alternative to DSC#[8] signal. See Section 9.4 for details.

Table 6: Serial-Port (UART) Signals

Signal	I/O	5V Tolerant	Reset Value	Pullup/ down	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
UART_CTS#	I	Same as DSC#[3] Table 4.						**Serial-Port Clear To Send**. This signal can be implemented by software, after reset, as an alternative to DSC#[3] signal. See Section 10.2 for details.
UART_DCD#	I	Same as DSC#[4] Table 4.						**Serial-Port Data Carrier Detect**. This signal can be implemented by software, after reset, as an alternative to DSC#[4] signal. See Section 10.2 for details.
UART_DSR#	I	Yes	HiZ					**Serial-Port Data Set Ready**.
UART_DTR#	I/O	Yes	HiZ	internal pulldown (50K ohm)	1	50	6	**Serial-Port Data Terminal Ready**. This signal is sampled during reset (Section 12.4) in order to set the controller's ID number in a multi-controller configuration (Section 5.3).
UART_RTS#	O	Same as DSC#[2] Table 4.						**Serial-Port Read To Send**. This signal can be implemented by software, after reset, as an alternative to DSC#[2] signal. See Section 10.2 for details.
UART_RxDRDY#	I	Yes	HiZ					**Serial-Port Receive Data**.
UART_TxDRDY#	I/O	Yes	HiZ	internal pulldown (50K ohm)	1	50	6	**Serial-Port Transmit Data**. This signal is sampled during reset (Section 12.4) in order to set the controller's ID number in a multi-controller configuration (Section 5.3).
UART_XIN	I	Same as DSC#[5] Table 4.						**Serial-Port External Crystal Input**. This signal can be implemented by software, after reset, as an alternative to DSC#[5] signal. See Section 10.2 for details.

Table 7: Utility Signals

Signal	I/O	5V Tolerant	Reset Value	Pullup/ down	Toggle Rate (MHz)	AC Load (pF)	DC Drive (mA)	Description
SMC	I	No						**Scan Mode Control**. Selects test type. Low for normal operation.
TEST#	I	Yes						**Test-Mode Enable**. Enables test mode. High for normal operation.
TEST_SEL	I	Yes						**Test Select**. Selects test type. Low for normal operation.

4.0 Register and Resource Summary

4.1 Register Summary

Table 8 summarizes the controller’s internal register set. This listing is organized by the base-address offset, shown in the left-most column of the table. The base address for the register set is specified by the INTCS Physical Device Address Register (Section 5.4). Detailed descriptions of each register are given in the sections listed in the right-most *Reference* column of the table.

The PCI-related registers are shown in two separate blocks in Table 8. The main PCI-Bus Registers begin at offset 0x00E0, and the PCI Configuration Space Registers begin at offset 0x0200. The PCI Configuration Space Registers can actually be accessed via two different methods, as described in Section 7.13.

If you configure the controller’s CPU interface to operate in Big-Endian mode (Section 5.2.6), see Section 13.0 for the implications of accessing registers in this mode.

Table 8: Register Summary

Offset From Base [a]	Register Name	Acronym	Size (bytes)	CPU-Bus R/W	Reset Value	Reference
Physical Device Address Registers (PDARs)—*See Section 5.4 on page 45*						
0x0000	SDRAM Bank 0	SDRAM0	8	R/W	0x0 0000 00D0	Section 5.4 on page 45
0x0008	SDRAM Bank 1	SDRAM1	8	R/W	0x0 0000 00D0	Section 5.4 on page 45
0x0010	Device Chip-Select 2	DCS2	8	R/W	0x0 0000 0000	Section 5.4 on page 45
0x0018	Device Chip-Select 3	DCS3	8	R/W	0x0 0000 0000	Section 5.4 on page 45
0x0020	Device Chip-Select 4	DCS4	8	R/W	0x0 0000 0000	Section 5.4 on page 45
0x0028	Device Chip-Select 5	DCS5	8	R/W	0x0 0000 0000	Section 5.4 on page 45
0x0030	Device Chip-Select 6	DCS6	8	R/W	0x0 0000 0000	Section 5.4 on page 45
0x0038	Device Chip-Select 7	DCS7	8	R/W	0x0 0000 0000	Section 5.4 on page 45
0x0040	Device Chip-Select 8	DCS8	8	R/W	0x0 0000 0000	Section 5.4 on page 45
0x0048	*reserved*	—	8	R	0x0 0000 0000	—
0x0050	*reserved*	—	8	R	0x0 0000 0000	—
0x0058	*reserved*	—	8	R	0x0 0000 0000	—
0x0060	PCI Address Window 0	PCIW0	8	R/W	0x0 0000 00C0	Section 5.4 on page 45
0x0068	PCI Address Window 1	PCIW1	8	R/W	0x0 0000 00C0	Section 5.4 on page 45
0x0070	Controller Internal Registers and Devices	INTCS	8	R/W	0x0 1Fh0 00EF [b]	Section 5.4 on page 45
0x0078	Boot ROM Chip-Select	BOOTCS	8	R/W	0x0 1FC0 002F [c]	Section 5.4 on page 45
CPU Interface Registers—*See Section 5.5 on page 50*						
0x0080	CPU Status	CPUSTAT	8	R/W	0x0000 0000 0000 0N00	Section 5.5.1 on page 50
0x0088	Interrupt Control	INTCTRL	8	R/W	0X8888 8888 8888 8888	Section 5.5.2 on page 52
0x0090	Interrupt Status 0	INTSTAT0	8	R	0x0000 0000 0000 0000	Section 5.5.3 on page 55
0x0098	Interrupt Status 1 and CPU Interrupt Enable	INTSTAT1	8	R/W	0x0001 0000 0000 0000	Section 5.5.4 on page 55
0x00A0	Interrupt Clear	INTCLR	8	R/W	0x0000 0000 0000 0000	Section 5.5.5 on page 56
0x00A8	PCI Interrupt Control	INTPPES	8	R/W	0x0000 0000 0000 0000	Section 5.5.6 on page 57
0x00B0	*reserved*	—	8	R	0x0000 0000 0000 0000	—
0x00B8	*See PCI-Bus Registers, below*					
Memory-Interface Registers—*See Section 6.6 on page 72*						

Table 8: Register Summary (continued)

Offset From Base [a]	Register Name	Acronym	Size (bytes)	CPU-Bus R/W	Reset Value	Reference
0x00C0	Memory Control	MEMCTRL	8	R/W	0x0000 0000 0000 0080	Section 6.6.1 on page 72
0x00C8	Memory Access Timing	ACSTIME	8	R/W	0x0000 0000 0000 001F	Section 6.6.2 on page 74
0x00D0	Memory Check Error Status	CHKERR	8	R	0x0000 0000 0000 0000	Section 6.6.3 on page 74
0x00D8	*reserved*	—	8	R	0x0000 0000 0000 0000	—
PCI-Bus Registers—*See Section 7.11 on page 91*						
0x00E0	PCI Control	PCICTRL	8	R/W	0X6000 0000 8000 0000	Section 7.11.1 on page 91
0x00E8	PCI Arbiter	PCIARB	8	R/W	0x0050 0011 1100 003F	Section 7.11.2 on page 98
0x00F0	PCI Master (Initiator) 0	PCIINIT0	8	R/W	0x0000 0000 0000 8406	Section 7.11.3 on page 101
0x00F8	PCI Master (Initiator) 1	PCIINIT1	8	R/W	0x0000 0000 0000 8406	Section 7.11.3 on page 101
0x00B8 [d]	PCI Error	PCIERR	8	R/W	0x0000 0000 0000 0000	Section 7.11.4 on page 103
See also the PCI Configuration Space Registers, starting at offset 0x0200, below						
Local-Bus Registers—*See Section 8.6 on page 120*						
0x0100	Local Bus Configuration	LCNFG	8	R/W	0x0 0000 0000	Section 8.6.1 on page 121
0x0108	*reserved*	—	8	R	0x0 0000 0000	—
0x0110	Local Bus Chip-Select Timing 2	LCST2	8	R/W	0x0 0000 0000	Section 8.6.2 on page 122
0x0118	Local Bus Chip-Select Timing 3	LCST3	8	R/W	0x0 0000 0000	Section 8.6.2 on page 122
0x0120	Local Bus Chip-Select Timing 4	LCST4	8	R/W	0x0 0000 0000	Section 8.6.2 on page 122
0x0128	Local Bus Chip-Select Timing 5	LCST5	8	R/W	0x0 0000 0000	Section 8.6.2 on page 122
0x0130	Local Bus Chip-Select Timing 6	LCST6	8	R/W	0x0 0000 0000	Section 8.6.2 on page 122
0x0138	Local Bus Chip-Select Timing 7	LCST7	8	R/W	0x0 0000 0000	Section 8.6.2 on page 122
0x0140	Local Bus Chip-Select Timing 8	LCST8	8	R/W	0x0 0000 0000	Section 8.6.2 on page 122
0x0148	*reserved*	—	8	R	0x0 0000 0000	—
0x0150	Device Chip-Select Muxing and Output Enables	DCSFN	8	R/W	0x0 0000 0000	Section 8.6.3 on page 125
0x0158	Device Chip-Selects As I/O Bits	DCSIO	8	R/W	0x0 0000 0000	Section 8.6.4 on page 128
0x0160	*reserved*	—	8	R	0x0 0000 0000	—
0x0168	*reserved*	—	8	R	0x0 0000 0000	—
0x0170	*reserved*	—	8	R	0x0 0000 0000	—
0x0178	Local Boot Chip-Select Timing	BCST	8	R/W	0x0 003F 8E3F	Section 8.6.5 on page 129
DMA Registers—*See Section 9.5 on page 133*						
0x0180	DMA Control 0	DMACTRL0	8	R/W	0x0000 0000 0000 0000	Section 9.5.1 on page 133
0x0188	DMA Source Address 0	DMASRCA0	8	R/W	0x0000 0000 0000 0000	Section 9.5.2 on page 136
0x0190	DMA Destination Address 0	DMADESA0	8	R/W	0x0000 0000 0000 0000	Section 9.5.3 on page 136
0x0198	DMA Control 1	DMACTRL1	8	R/W	0x0000 0000 0000 0000	Section 9.5.1 on page 133
0x01A0	DMA Source Address 1	DMASRCA1	8	R/W	0x0000 0000 0000 0000	Section 9.5.2 on page 136
0x01A8	DMA Destination Address 1	DMADESA1	8	R/W	0x0000 0000 0000 0000	Section 9.5.3 on page 136
0x01B0	*reserved*	—	8	R	0x0000 0000 0000 0000	—
0x01B8	*reserved*	—	8	R	0x0000 0000 0000 0000	—
Timer Registers—*See Section 5.6 on page 58*						
0x01C0	SDRAM Refresh Control	T0CTRL	8	R/W	0x0000 0001 0000 0186	Section 5.6.1 on page 58
0x01C8	SDRAM Refresh Counter	T0CNTR	8	R/W	0x0000 0000 0000 0000	Section 5.6.2 on page 59
0x01D0	CPU-Bus Read Time-Out Control	T1CTRL	8	R/W	0x0000 0000 0000 0000	Section 5.6.3 on page 59
0x01D8	CPU-Bus Read Time-Out Counter	T1CNTR	8	R/W	0x0000 0000 0000 0000	Section 5.6.4 on page 60
0x01E0	General-Purpose Timer Control	T2CTRL	8	R/W	0x0000 0000 0000 0000	Section 5.6.5 on page 60
0x01E8	General-Purpose Timer Counter	T2CNTR	8	R/W	0x0000 0000 0000 0000	Section 5.6.6 on page 61

Table 8: Register Summary (continued)

Offset From Base [a]	Register Name	Acronym	Size (bytes)	CPU-Bus R/W	Reset Value	Reference
0x01F0	Watchdog Timer Control	T3CTRL	8	R/W	0x0000 0000 0000 0000	Section 5.6.7 on page 61
0x01F8	Watchdog Timer Counter	T3CNTR	8	R/W	0x0000 0000 0000 0000	Section 5.6.8 on page 62
PCI Configuration Space Registers—*See Section 7.13 on page 105*						
0x0200 [e]	PCI Vendor ID	VID	2	R	0x1033	Section 7.13.1 on page 107
0x0202 [e]	PCI Device ID	DID	2	R	0x005A	Section 7.13.2 on page 107
0x0204 [e]	PCI Command	PCICMD	2	R/W	0x0000 or 0x0006 [f]	Section 7.13.3 on page 107
0x0206 [e]	PCI Status	PCISTS	2	R/W	0x02A0	Section 7.13.4 on page 108
0x0208 [e]	PCI Revision ID	REVID	1	R	0x01	Section 7.13.5 on page 109
0x0209 [e]	PCI Class Code	CLASS	3	R	0x06 0000	Section 7.13.6 on page 110
0x020C [e]	PCI Cache Line Size	CLSIZ	1	R/W	0x00	Section 7.13.7 on page 110
0x020D [e]	PCI Latency Timer	MLTIM	1	R/W	0x00	Section 7.13.8 on page 110
0x020E [e]	PCI Header Type	HTYPE	1	R	0x00	Section 7.13.9 on page 110
0x020F [e]	BIST	*unimplemented*	1	R	0x00	—
0x0210 [e]	PCI Base Address Register Control	BARC	8	R/W	0x0000 0000 0000 0004	Section 7.13.10 on page 110
0x0218 [e]	PCI Base Address Register 0	BAR0	8	R/W	0x0000 0000 0000 0000	Section 7.13.10 on page 110
0x0220 [e]	PCI Base Address Register 1	BAR1	8	R/W	0x0000 0000 0000 0000	Section 7.13.10 on page 110
0x0228 [e]	PCI Cardbus CIS Pointer	*unimplemented*	4	R	0x0000 0000	—
0x022C [e]	PCI Sub-System Vendor ID	SSVID	2	R/W	*depends on various conditions*	Section 7.13.11 on page 111
0x022E [e]	PCI Sub-System ID	SSID	2	R/W	*depends on various conditions*	Section 7.13.12 on page 111
0x0230 [e]	Expansion ROM Base Address	*unimplemented*	4	R	0x0000 0000	—
0x0234 [e]	*reserved*	—	6	R	0x00	—
0x023C [e]	PCI Interrupt Line	INTLIN	1	R/W	0xFF	Section 7.13.13 on page 112
0x023D [e]	PCI Interrupt Pin	INTPIN	1	R	0x01	Section 7.13.14 on page 112
0x023E [e]	PCI Min_Gnt	*unimplemented*	1	R	0x00	—
0x023F [e]	PCI Max_Lat	*unimplemented*	1	R	0x00	—
0x0240 [e]	PCI Base Address Register 2	BAR2	8	R/W	0x0000 0000 0000 0000	Section 7.13.10 on page 110
0x0248 [e]	PCI Base Address Register 3	BAR3	8	R/W	0x0000 0000 0000 0000	Section 7.13.10 on page 110
0x0250 [e]	PCI Base Address Register 4	BAR4	8	R/W	0x0000 0000 0000 0000	Section 7.13.10 on page 110
0x0258 [e]	PCI Base Address Register 5	BAR5	8	R/W	0x0000 0000 0000 0000	Section 7.13.10 on page 110
0x0260 [e]	PCI Base Address Register 6	BAR6	8	R/W	0x0000 0000 0000 0000	Section 7.13.10 on page 110
0x0268 [e]	PCI Base Address Register 7	BAR7	8	R/W	0x0000 0000 0000 0000	Section 7.13.10 on page 110
0x0270 [e]	PCI Base Address Register 8	BAR8	8	R/W	0x0000 0000 0000 0000	Section 7.13.10 on page 110
0x0278 [e]	PCI Base Address Register BOOT	BARB	8	R/W	0x0000 0000 0000 0004	Section 7.13.10 on page 110

Table 8: Register Summary (continued)

Offset From Base [a]	Register Name	Acronym	Size (bytes)	CPU-Bus R/W	Reset Value	Reference
0x0280: 0x02FF	*reserved*	—	128	R	0x00	—
Serial-Port Registers—*See Section 10.4 on page 139*						
0x0300	UART Receiver Data Buffer	UARTRBR	1	R	0x0000 0000 0000 00xx	Section 10.4.1 on page 139
0x0300	UART Transmitter Data Holding	UARTTHR	1	W	0x0000 0000 0000 00xx	Section 10.4.2 on page 139
0x0308	UART Interrupt Enable	UARTIER	1	R/W	0x0000 0000 0000 0000	Section 10.4.3 on page 139
0x0300	UART Divisor Latch LSB	UARTDLL	1	R/W	0x0000 0000 0000 00xx	Section 10.4.4 on page 140
0x0308	UART Divisor Latch MSB	UARTDLM	1	R/W	0x0000 0000 0000 00xx	Section 10.4.5 on page 140
0x0310	UART Interrupt ID	UARTIIR	1	R	0x0000 0000 0000 0001	Section 10.4.6 on page 140
0x0310	UART FIFO Control	UARTFCR	1	W	0x0000 0000 0000 0000	Section 10.4.7 on page 141
0x0318	UART Line Control	UARTLCR	1	R/W	0x0000 0000 0000 0000	Section 10.4.8 on page 142
0x0320	UART Modem Control	UARTMCR	1	R/W	0x0000 0000 0000 0000	Section 10.4.9 on page 143
0x0328	UART Line Status	UARTLSR	1	R/W	0x0000 0000 0000 0060	Section 10.4.10 on page 144
0x0330	UART Modem Status	UARTMSR	1	R/W	0x0000 0000 0000 0000	Section 10.4.11 on page 144
0x0338	UART Scratch	UARTSCR	2	R/W	0x0000 0000 0000 00xx	Section 10.4.12 on page 145

a. At reset, the base address for the register set of a single controller, or the Main Controller in a multi-controller configuration, is 0x0 1FA0 0000. For the base address of other controllers in a multi-controller configuration, see Section 5.3.

b. The "h" nibble in this reset value changes in multi-controller configurations (Section 5.3).

c. The BOOTCS reset value depends on the size of the PCI bus, size of boot ROM, and other conditions. See Section 5.4.2 and Section 5.3.

d. This is a non-consecutive address.

e. These are the controller's internal addresses for accessing the PCI Configuration Space Registers. To obtain these addresses, a value of 0x0200 has been added to the offset of the Configuration Space Registers shown in Table 26 on page 105. There are two paths for accessing each of these registers. See Section 7.13.

f. PCICMD resets to 0x0000 when PCICR# is negated, or to 0x0006 when PCICR# is asserted.

4.2 Resource-Accessibility Summary

Table 9 summarizes the accessibility of controller resources to the CPU and DMA logic.

Table 9: System Resources Accessible to the CPU and DMA

Targets Accessible Through Controller	Accessible By CPU	Accessible By DMA	Reference
Boot ROM	Yes	Yes	Section 6.4 on page 64
SDRAM Memory	Yes	Yes	Section 6.5 on page 67
PCI Memory Space [a]	Yes	Yes	Section 7.4 on page 81
PCI I/O space [a]	Yes	Yes	Section 7.4.6 on page 85
PCI Configuration Registers [a]	Yes	Yes	Section 7.12 on page 103
External (DCS[8:2]) Devices [b]	Yes	Yes	Section 8.6.3 on page 125
Controller's Internal Registers [c]	Yes	Yes	Table 8 on page 31

a. Via PCI Address Windows (see Section 5.4) with controller as PCI master.

b. Any device selected by a DCS[8:2] signal. Such devices can reside on the Memory or Local Bus.

c. Includes Physical Device Address Registers (PDARs), System and CPU control registers, Memory-Bus control registers, PCI-Bus control registers and PCI configuration registers, Local-Bus control registers, DMA control registers, UART control registers, Timer control registers

Table 10 summarizes the accessibility of controller resources to PCI-Bus masters.

Table 10: System Resources Accessible to PCI-Bus Masters

Targets Accessible Through Controller	Accessible By PCI-Bus Masters	Reference
Boot ROM	Yes	Section 6.4 on page 64
SDRAM Memory	Yes	Section 6.5 on page 67
PCI I/O space	No	
PCI Configuration Space	No [a]	Section 7.12 on page 103
External (DCS[8:2]) Devices [b]	Yes	Section 8.6.3 on page 125
Controller's Internal Registers [c]	Yes	Table 8 on page 31

a. A PCI-Bus master cannot access non-controller PCI Configuration Space via the controller, but it can access the controller's own internal PCI configuration registers either directly (see Table 8 on page 31) or when the PCI Central Resource asserts the appropriate IDSEL signal.

b. Any device selected by a DCS[8:2] signal. Such devices can reside on the Memory or Local Bus.

c. Includes Physical Device Address Registers (PDARs), System and CPU control registers, Memory-Bus control registers, PCI-Bus control registers and PCI configuration registers, Local-Bus control registers, DMA control registers, UART control registers, Timer control registers

Table 11 summarizes the accessibility of controller resources to Local-Bus masters.

Table 11: System Resources Accessible to Local-Bus Masters

Targets Accessible Through Controller	Accessible By Local-Bus Masters [a]	Reference
Boot ROM	Yes [b]	Section 6.4 on page 64
SDRAM Memory	Yes	Section 6.5 on page 67
PCI Memory Space [c]	Yes	Section 7.4 on page 81
PCI I/O space [c]	Yes	Section 7.4.6 on page 85
PCI Configuration Space [c]	Yes	Section 7.12 on page 103
External (DCS[8:2]) Devices [d]	Yes [b]	Section 8.6.3 on page 125
Controller's Internal Registers [e]	Yes	Table 8 on page 31

a. Accesses by Local-Bus masters are implemented by the controller's DMA logic, as described in Section 8.4.2.

b. Device must be on the Memory Bus.

c. Via PCI Address Windows (see Section 5.4) with controller as PCI master.

d. Any device selected by a DCS[8:2] signal. Such devices can reside on the Memory or Local Bus.

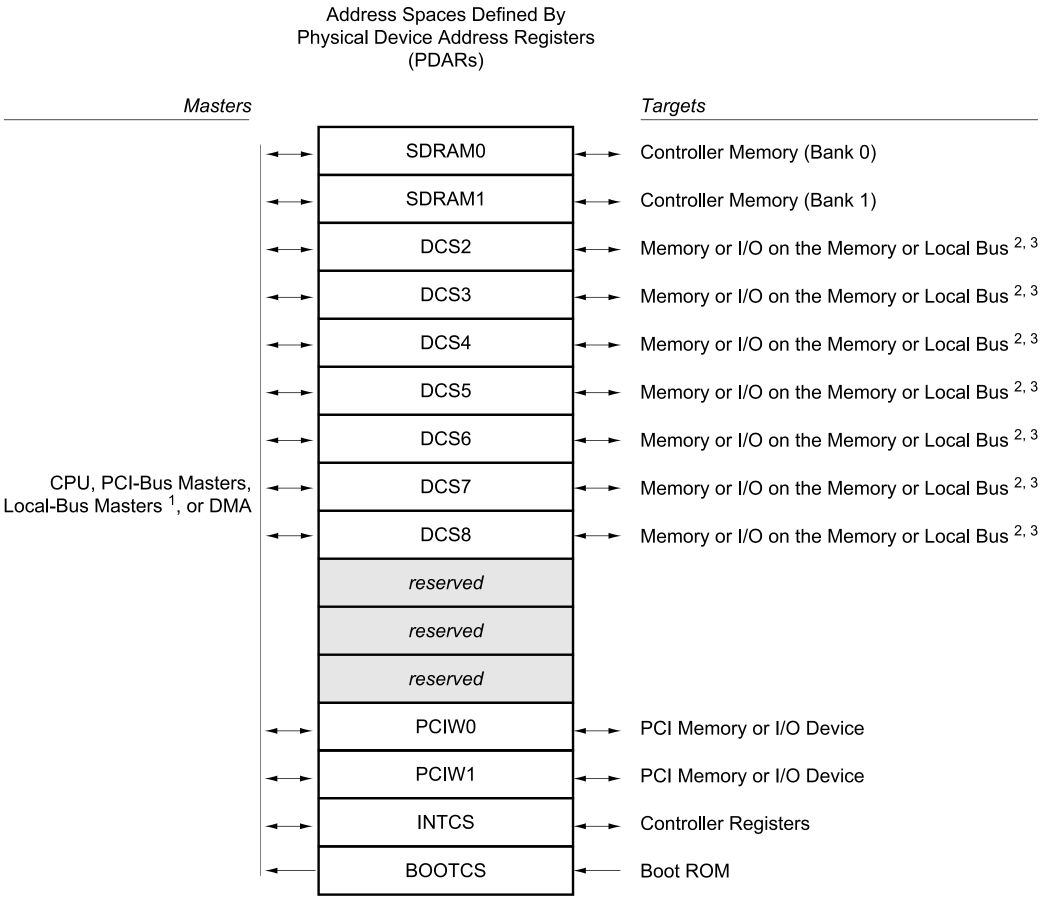
e. Includes Physical Device Address Registers (PDARs), System and CPU control registers, Memory-Bus control registers, PCI-Bus control registers and PCI configuration registers, Local-Bus control registers, DMA control registers, UART control registers, Timer control registers

4.3

Address Space Summary

Figure 8 summarizes the accessibility of address spaces supported by the controller’s Physical Device Address Registers (PDARs), which are described fully in Section 5.4.

Figure 8: Access Supported By Physical Device Address Registers (PDARs)



Masters	Address Spaces Defined By Physical Device Address Registers (PDARs)	*Targets*
CPU, PCI-Bus Masters, Local-Bus Masters [1], or DMA	SDRAM0	Controller Memory (Bank 0)
	SDRAM1	Controller Memory (Bank 1)
	DCS2	Memory or I/O on the Memory or Local Bus [2, 3]
	DCS3	Memory or I/O on the Memory or Local Bus [2, 3]
	DCS4	Memory or I/O on the Memory or Local Bus [2, 3]
	DCS5	Memory or I/O on the Memory or Local Bus [2, 3]
	DCS6	Memory or I/O on the Memory or Local Bus [2, 3]
	DCS7	Memory or I/O on the Memory or Local Bus [2, 3]
	DCS8	Memory or I/O on the Memory or Local Bus [2, 3]
	reserved	
	reserved	
	reserved	
	PCIW0	PCI Memory or I/O Device
	PCIW1	PCI Memory or I/O Device
	INTCS	Controller Registers
	BOOTCS	Boot ROM

Note 1: Accesses requested by Local-Bus masters are performed by the controller's DMA logic.

Note 2: Only if the associated DCS[n] signal is not configured for UART control, DMA hardware handshaking, or general-purpose I/O.

Note 3: Local-Bus masters cannot access Local-Bus targets through the controller, but they can access Local-Bus targets directly.

5074-064.eps

5.0 CPU/System Interface and Registers

The controller can interface directly to a VR5000 CPU, in full compliance with the *VR5000 Bus Interface User's Manual, Revision 1.1*. The controller can operate with or without direct connection to a VR5000 CPU. When it operates without direct connection, another CPU in the system would configure the controller, as shown in Figure 6 on page 17.

The CPU interface operates at a maximum frequency of 100 MHz and supports a peak block-transfer throughput of 800 MB/sec and a maximum sustained throughput of 640 Mbytes/sec.

Through the controller, the CPU can gain access to memory, the PCI Bus, the Local Bus, and the controller's internal registers (Table 8). All CPU bus-cycle types and data sizes supported by the VR5000 SysAD bus are supported by the controller, except that—due to the pipelined nature of the controller's CPU interface—the Pipelined Write Mode is the only non-block write mode supported.

Section 12.4.2 on page 152 describes the CPU initialization procedures, the CPU operating modes imposed by the controller. Multi-controller configurations (Section 5.3) allow either multiple VRC5074 controllers or a single VRC5074 controller and other external agents to reside on the CPU interface. Figure 7 on page 18 shows an example. Such system designs must pay special attention to loading issues on the CPU bus.

5.1 CPU and System Configuration and Monitoring

Software configures and monitors the CPU interface and the controller's general system functions by using the following registers:

- ❑ CPU and Controller Initialization, Section 12.4 on page 151.
- ❑ Physical Device Address Registers (PDARs), Section 5.4 on page 45.
- ❑ CPU Interface Registers, Section 5.5 on page 50.
- ❑ Timer Registers, Section 5.6 on page 58.

5.2 CPU Interface

5.2.1 Signal Connections to CPU

Table 12 summarize the signal connections between the CPU and the controller.

Table 12: CPU-Controller Signal Connections

CPU		Controller Signal	
Signal	**R/W**	**R/W**	**Signal**
BigEndian	I	I/O	BigEndian
ColdReset#	I	O	ColdReset#
ExtRqst#	I (tied High)		—
Int#[5:0]	I	O	Int#[5:0]
JTCK	I [a]		—
JTDI	I [a]		—
JTDO	O [a]		—

Table 12: CPU-Controller Signal Connections (continued)

CPU		Controller Signal	
Signal	**R/W**	**R/W**	**Signal**
JTTMS#	I [a]		—
ModeClock	O	I	ModeClock
ModeIn	I	O	ModeOut
NMI#	I	O	NMI#
RdRdy#	I (tied Low)		—
Release#	O [a]		—
Reset#	I	O	Reset#
ScCWE#[1:0]	O [a]		—
ScDCE#[1:0]	O [a]		—
ScDOE#	I	O	ScDOE#
ScLine[15:0]	I/O [a]		—
ScMatch	I	I	ScMatch
ScCLR#	O [a]		—
ScTCE#	I/O [a]		—
ScTDE#	O [a]		—
ScTOE#	O [a]		—
ScValid#	I/O [a]		—
ScWord[1:0]	I/O	I/O	ScWord[1:0]
SysAD[63:0]	I/O	I/O	SysAD[63:0]
SysADC[7:0]	I/O	I/O	SysADC[7:0]
SysClock	I	I	SysClock
SysCmd[8:0]	I/O	I/O	SysCmd[8:0]
SysCmdP	I/O [a]		—
ValidIn#	I	I/O	CntrValid#
ValidOut#	O	I	CPUValid#
VccOk	I	O	CntrVccOk
WrRdy#	I	I/O	WrRdy# [b]

a. The controller does not connect to this signal.

b. In multi-controller configurations, all external agents' MCWrRdy# signals must be ORed together and registered in an external device. The output of this device must then be wired to the CPU's WrRdy# input and to the controllers' WrRdy# inputs.

5.2.2 CPU-Interface Data Path

The controller samples addresses and data from the CPU on the rising edge of SysClock. On the controller side of the CPU interface, a 16 x 8-byte (128-byte) CPU-to-controller FIFO (the *CPU-Interface FIFO)* buffers SysAd information. The FIFO can hold 16 dwords of SysAd items (address or data) and is surrounded by two pipeline stages. On the CPU side of the interface, every signal is buffered through a row of registers.

If the CPU issues a memory read while the controller's CPU-Interface FIFO is empty, the request bypasses the FIFO and is loaded directly into a row of registers. If the FIFO or controller-side register row is not empty, addresses do not bypass the FIFO. If the CPU interface is idle, a read request bypasses both the FIFO and the controller-side register row, saving a minimum of one clock compared to the non-idle case. The CPU's RdRdy input signal is not driven by the controller and should be tied Low. Since the controller supports only a single pending read, the read is implicitly stalled until the read data is returned.

For CPU reads, the controller drives response data from the target onto the SysAd bus and asserts the CntrValid# signal. For block reads, the controller negates ScDOE# after it has been determined that the secondary cache (if implemented) has missed on the request. ScDOE# remains negated until the third of four dwords is returned to the CPU. Since the re-assertion of ScDOE# indicates that the controller will drive the last dword in the next cycle, the third dword is held back and ScDOE# remains negated until the fourth dword is driven by the responding resource. This additional clock for the third and fourth dwords is necessary due to the unpredictable timing relationship between the third and fourth dwords. The timing for ScDOE# is the same in a multi-controller configuration as it is in a single-controller configuration.

Data for CPU writes always goes through the CPU-Interface FIFO before being loaded into the controller-side register row. The controller requests the appropriate resource when the address and cycle-type information are loaded into the register row. When the associated resource is available, the address and data at the output of the FIFO is clocked into the targeted resource and the FIFO advances.

5.2.3 SysAd Flow Control

The CPU may be stalled if it requests access to a target that is not ready. This occurs when the outstanding requests have filled the controller’s CPU-Interface FIFO. The controller monitors the status of the FIFO and stalls the CPU if it needs to prevent a FIFO overflow. Stalling occurs either implicitly or explicitly. The CPU is implicitly stalled if a read resource is unavailable; the controller does not proceed until it receives the requested data for the pending read cycle. The controller always leaves room in the FIFO for a read request, so that reads are never explicitly stalled.

The CPU is explicitly stalled when the buffer contains six items of SysAd information (address or data); in this case, the controller negates WrRdy#. Although the FIFO can hold a maximum of 16 items of SysAd information, the action of stalling the CPU must begin early, due to the pipelined nature of both the controller and the CPU interface.

For example, the following case is one in which the FIFO will fill up. When WrRdy# is negated, the CPU may issue two more requests (a non-block write followed by a block write) before it is stalled on the third request. These two additional requests represent seven more items in the FIFO. Including two items in the controller’s pipeline, the total in the FIFO by the time the CPU stalls is 15 items. This leaves one place in the FIFO for the CPU to issue a read request. At this point, the CPU is stalled. If the third request issued following the negation of WrRdy# is a write request, that request is explicitly stalled. If the third request is a read, however, that request goes into the last place in the buffer and the CPU is implicitly stalled.

If the FIFO begins to empty, or if the above worst-case scenario does not occur, i.e. the FIFO does not contain 15 items when the CPU is stalled, the controller reasserts and negates WrRdy# based on the FIFO depth and the pending stalled write request. If the stalled request is a non-block write request, the controller reasserts WrRdy# for one clock when the FIFO contains less than 13 items. Similarly, if the stalled request is a block write request, the controller reasserts WrRdy# for one clock once the buffer contains less than 10 items. If the FIFO empties to five items, the controller asserts WrRdy# until the FIFO fills back up to six or more items.

If the CPU is stalled implicitly on a read request, the controller reasserts WrRdy# when the FIFO empties to five items. The FIFO always empties before the completion of the read request.

5.2.4 Parity Checking and Generation

By default, when the DISPC and DISCPUPC bits are cleared in the CPU Status Register (CPUSTAT, Section 5.5.1), the controller checks even parity during CPU writes and generates even parity during CPU reads. The SysAd bus is only checked during data transfers, not address transfers. Write parity is checked when the information is in the controller's CPU-interface register row. Read parity is generated when the data is driven from the resource to the CPU interface, after byte swapping (if necessary). If the CPCEEN bit is set in the Interrupt Control Register (INTCTRL, Section 5.5.2) an interrupt is generated when a CPU read or write parity error is detected.

When returning bad data to the CPU during a block read (cache-line fill), the controller sets bit SysCmd[5] and bad parity on SysADC[7:0] for *each* data word of the block that is bad. However, the VR5000 CPU only looks at command bit 5 of the *first* data word in a block. Thus, if the data error is in a word of the block other than the first word, the CPU will only notice the error when the data is fetched from the CPU's cache, in which case a cache exception (not a bus-error exception) is generated.

5.2.5 CPU Reads

The CPU's RdRdy# input should be tied Low (asserted). Reads are self-throttling, because only one outstanding read is generated by the CPU and supported by the controller.

5.2.5.1 Read Requests that Hit the Secondary Cache

A block read issued by the CPU may be a request that was speculatively issued to both the controller and the secondary cache. If the secondary cache hits on the request, the controller's CPU interface and (potentially) the targeted resource must abort the request. The controller does this by monitoring ScMatch, which is an input to both the CPU and the controller. When ScMatch is asserted two clocks after the address is issued, the controller aborts the read from the targeted resource (if the request has made it through the FIFO) and removes the request from the FIFO.

5.2.5.2 Non-Matching Read Address

When a read request is issued by the CPU, but the address does not match any of those programmed into the controller's Physical Device Address Registers (Section 5.4), the controller responds to the CPU by driving all 0s on the SysAd bus. The TMODE bits in the CPU Status Register (CPUSTAT, Section 5.5) determine whether good or bad parity will be generated for this response. An optional, programmable interrupt may be generated when this condition occurs, as specified by the CNTDEN bit in the Interrupt Status Register 0 (INTSTAT0, Section 5.5.2).

For multi-controller configurations (Section 5.3), if no device responds to the read request after a programmable time-out interval has elapsed, as specified in the CPU-Bus Read Time-Out Control Register (T1CTRL, Section 5.6.3), the Main Controller responds by driving all 0s on the SysAd bus. As in single-controller configurations, good or bad parity may be generated for this response, and an interrupt may be generated when the condition occurs.

5.2.5.3 Branches to Unaligned Addresses

The controller implements a hardware fix for the VR5000 CPU bug that prevents the CPU from correctly handling a doubleword fetch due to a branch to an unaligned address.

The controller uses the SysAD[63:0] bits and the SysCmd[2:0] bits to internally generate byte enables, per Table 4.14 of the *Vr5000 Bus Interface User's Manual*. For a 4-byte read to an address that is not doubleword-aligned, the controller correctly generates the byte enables and returns the proper data to the CPU, on the proper byte lanes.

5.2.6 Endian Configuration

The controller's CPU interface supports either big- or little-endian byte ordering. However, all of the controller's logic, except the CPU interface, operates solely in little-endian mode. To implement a big-endian CPU interface, do either of the following:

- ❑ Set the Big Endian (BE) bit in the CPU's Config Register during the CPU and controller initialization sequence (Section 12.4), or
- ❑ Tie the controller's and the CPU's BigEndian signal High.

If either or both of these conditions occur, the controller will swap incoming and outgoing bytes on the SysAd bus so that the CPU can operate in big-endian mode while the controller operates in little-endian mode. The software implications of this, and some related PCI-device examples, are described in Section 13.0.

5.3 Multi-Controller Configurations

The controller can support multiple external agents, including multiple VRC5074 controllers. Figure 7 on page 18 illustrates such a system design. This is called *multi-controller mode*, or a *multi-controller configuration*. The logic for this support handles functions such as which controller initializes the CPU, which controller responds to Boot ROM accesses, separation of the default register address space of each controller, compensation for externally combining the individual WrRdy# signals, and handling bus cycles that are not responded to.

5.3.1 Distinguishing Between Multiple Controllers

In a single-controller configuration, the base address of the controller's internal configuration registers is 0x1FA0_0000 after reset. In a multi-controller configuration, the UART_DTR# and UART_TxDRDY# signals are sampled at reset to identify each controller and assign separate address spaces for their internal registers. The sampled ID determines the base address of each controller's register set. An ID of 00 identifies the *Main Controller*, as shown in Table 13. Software can read this ID in the MAINCTRL field of the CPU Status Register (CPUSTAT), Section 5.5.1.

Table 13: Reset Configuration Signals for Multi-Controller Configurations

Signal Sampled at Reset		Controller ID Number	Base Address Of Controller's Internal Registers After Reset (PDAR = INTCS)	Base Address Of Boot ROM After Reset (PDAR = BOOTCS)
UART_DTR#	UART_TxDRDY#			
0	0	00 (Main Controller)	0x0 1FA0_0000 [a]	0x0 1FC0 0000

Table 13: Reset Configuration Signals for Multi-Controller Configurations

Signal Sampled at Reset		Controller ID Number	Base Address Of Controller's Internal Registers After Reset (PDAR = INTCS)	Base Address Of Boot ROM After Reset (PDAR = BOOTCS)
UART_DTR#	UART_TxDRDY#			
0	1	01	0x0 1F80_0000	*disabled*
1	0	10	0x0 1F60_0000	*disabled*
1	1	11	0x0 1F40_0000	*disabled*

a. This is the base address for all single-controller configurations, and for the Main Controller in a multi-controller configuration.

UART_DTR# and UART_TxDRDY# have 50k-ohm pulldowns internally. Thus, in a single-controller configuration, no connection to these signals is necessary.

5.3.2 The Main Controller

The Main Controller is responsible for any activity that is not performed by any other controller. At boot time, the Main Controller performs the CPU's initialization sequence (Section 12.4) by driving the clock, address, and command to the Serial Mode EEPROM (if present), reading in the configuration information (if Serial Mode EEPROM is present) or providing the default (if Serial Mode EEPROM is not present), correcting any illegal cases, and sending initialization information to the CPU. Other controller(s) in the system monitor the initialization sequence in order to obtain the configuration information that is relevant to them. After initialization, the Main Controller responds to Boot ROM fetches.

The concepts of Main Controller and PCI Central Resource (Section 7.8) are unrelated. The controller can be a Main Controller for a given CPU, but that CPU might not be the Main CPU in the system, and the Main Controller for that CPU, or any other CPU in the system, might not provide the PCI Central Resource for the system.

5.3.3 Programming

All controllers in a multi-controller configuration must have their TMODE bits programmed in their CPU Status Register (CPUSTAT), Section 5.5.1, as soon as possible after the system boots. These bits indicate to the controller that this is a multi-controller configuration and how the Main Controller should handle read requests that are not responded to. The bits must be programmed before read requests are issued to devices other than the Main Controller.

If the TMODE bits are not programmed before read requests are issued to devices other than the Main Controller, the controller behaves as though in single-controller configuration. A read to another controller causes a no-target decode in the Main Controller. The Main Controller responds immediately with all zeros. There is, then, a high likelihood that either this data will be taken as the response for the read request or that the true response and the Main Controller's no-target response will collide, causing bus contention.

In a multi-controller configuration, the Main Controller waits for the CPU-Bus Read Time-Out Control Register (T1CTRL), Section 5.6.3 and Section 5.6.4, to terminate before responding to the request with all zeros. The time-out counter must have been initialized for this feature to work. This gives the targeted controller time to respond to the read request.

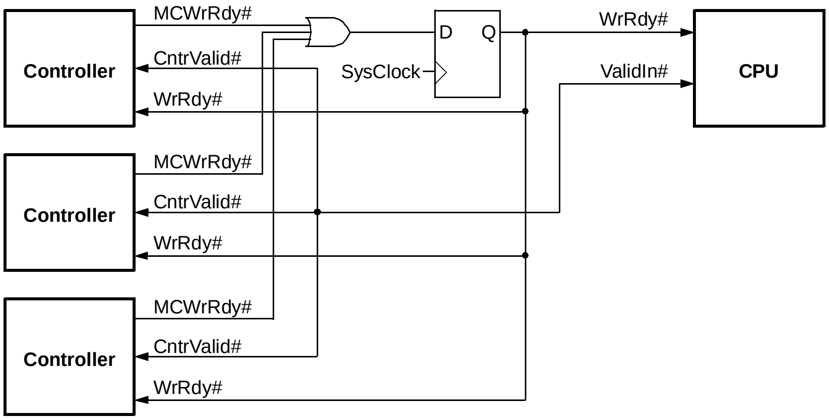
When outstanding read requests are responded to, the Main Controller automatically resets the CPU-Bus Read Time-Out Counter. Read requests in any of the controller pipelines are discarded when a response is provided to the CPU. This is possible because the CPU has only one read outstanding at any time, so if a read response occurs, a read in the pipeline is by definition not destined for this controller. If a write request does not decode in the controller, the write data is disregarded regardless of whether the system implements a single- or multi-controller configuration.

5.3.4 CntrValid#, WrRdy# and MCWrRdy#

In a multi-controller configuration, the bidirectional CntrValid# signal is shared by all controllers. On a CPU access, all controllers decode the access but only one controller (the *active controller)* decodes the address as being for it. The active controller drives CntrValid# and all other controllers take it as an input, so that they can keep track of what is happening on the CPU bus.

WrRdy# is an input in a multi-controller configuration, as opposed to an output in single-controller configurations. Due to the speed of the CPU's bus interface, all of the MCWrRdy# outputs from all controllers must be externally ORed and registered (on SysClock) to generate WrRdy# to the CPU and all controllers. Figure 9 shows the connections.

Figure 9: Multi-Controller Signal Connections



If the CPU is writing to memory attached to one of the controllers, that controller will negate its MCWrRdy# output while the other controllers will assert their MCWrRdy# outputs. The OR gate will then cause WrRdy# to the CPU and all other controllers to be negated, thus indicating that further writes are being held off.

Programming the TMODE bits to indicate a multi-controller configuration causes the controller to compensate for the external combination of all external agents' WrRdy# signals. The controller expects this extra clock delay when monitoring the WrRdy# input in a multi-controller configuration, and it adjusts the CPU-Interface FIFO water marks accordingly.

5.3.5 Access Targeting

In a multi-controller configuration in which one or more VRC5074 controllers, and possibly other devices, share a common CPU bus, each VRC5074 controller watches all activity on the shared bus to determine which accesses are intended for it. If that VRC5074 controller determines that the current access is not intended for it, that controller flushes writes after the CPU-Interface FIFO (Section 5.2.2) and drops reads either when it sees another device’s answer or after its own read time-out.

5.4 Physical Device Address Registers (PDARs)

The bottom 36 bits (bits 35:0) of the CPU's SysAD bus are the valid physical address bits. These are decoded by the controller according to masks in the controller's 13 Physical Device Address Registers (PDARs). Figure 8 on page 36 shows how the PDARs facilitate accesses by various bus masters to various bus targets.

Table 14 summarizes the characteristics of the PDARs. The text that follows specifies the contents of each PDAR.

Table 14: Physical Device Address Registers (PDARs)

Register	Symbol	Offset	R/W	Reset Value	Description
SDRAM Bank 0	SDRAM0	0x0000	R/W	0x0 0000 00D0	SDRAM memory bank 0.
SDRAM Bank 1	SDRAM1	0x0008	R/W	0x0 0000 00D0	SDRAM memory bank 1.
Device Chip-Select 2 [a]	DCS2	0x0010	R/W	0x0 0000 0000	Configures DCS#[2] signal.
Device Chip-Select 3 [a]	DCS3	0x0018	R/W	0x0 0000 0000	Configures DCS#[3] signal.
Device Chip-Select 4 [a]	DCS4	0x0020	R/W	0x0 0000 0000	Configures DCS#[4] signal.
Device Chip-Select 5 [a]	DCS5	0x0028	R/W	0x0 0000 0000	Configures DCS#[5] signal.
Device Chip-Select 6 [a]	DCS6	0x0030	R/W	0x0 0000 0000	Configures DCS#[6] signal.
Device Chip-Select 7 [a]	DCS7	0x0038	R/W	0x0 0000 0000	Configures DCS#[7] signal.
Device Chip-Select 8 [a]	DCS8	0x0040	R/W	0x0 0000 0000	Configures DCS#[8] signal.
reserved	RFU9	0x0048	R	0x0 0000 0000	—
reserved	RFUa	0x0050	R	0x0 0000 0000	—
reserved	RFUb	0x0058	R	0x0 0000 0000	—
PCI Address Window 0	PCIW0	0x0060	R/W	0x0 0000 00C0	Configures PCI Address Window 0. This address window can be accessed by the CPU or DMA, with the controller acting as the PCI-Bus master. See Section 7.4.2 for an example of address generation.
PCI Address Window 1	PCIW1	0x0068	R/W	0x0 0000 00C0	Configures PCI Address Window 1. This address window can be accessed by the CPU or DMA, with the controller acting as the PCI-Bus master. See Section 7.4.2 for an example of address generation.
Controller Internal Registers and Devices	INTCS	0x0070	R/W	0x0 1Fn0 00EF	Configures controller's internal registers. The reset value changes in a multi-controller configuration (see Section 5.3.3).
Boot Chip-Select [a]	BOOTCS	0x0078	R/W	0x0 1FC0 002F [b]	Configures BootCS# signal. The reset value changes in a multi-controller configuration (see Section 5.3.1).

a. When the controller is configured for 32-bit PCI operation (PCI64# negated), the boot memory and the seven DCS devices can be individually configured by the MEM/LOC bit in the PDAR (Section 5.4) to appear on the memory bus or the Local Bus. When the controller is configured for 64-bit PCI operation (PCI64# asserted), these devices always appear on the memory bus.

b. The BOOTCS reset value depends on the size of the PCI bus, size of boot ROM, and other conditions. See Section 5.4.2 and Section 5.3.

5.4.1 Initialization State of PDARs

After the Serial Mode EEPROM initializes the CPU at reset (Section 12.0), the PDARs turn off all physical address space except the chip-selects for the controller's internal register space (INTCS) and the Boot ROM (BOOTCS). The PDARs are located at the first 16 dwords of the controller’s internal register space. For a single-controller configuration, or for the Main Controller in a multi-controller configuration (Section 5.3.3), these 16 dwords are located at offsets 0x0078:0x0000 from base address 0x0_1FA0_0000. Boot ROM for a single-controller configuration is decoded to base address 0x0_1FC0_0000, although this changes for a multi-controller configuration (Table 13)**.**

After reset, the PDARs may be programmed to occupy any valid physical address space and to decode physical address ranges from 4GB down to 2MB. A maximum of 15 address bits, SysAD[35:21], can be decoded. 16MB ranges must start at 16MB boundaries; 1GB ranges must start at 1GB boundaries. If the address mappings of two Physical Device Address Registers overlap, the lower-numbered Physical Device Address Register decode is selected.

When programming a PDAR, the register should be read immediately after writing it. This ensures that address decoders are properly configured.

5.4.2 PDAR Fields

All PDARs have the same bit organization, except where noted below.

Bit 3:0 MASK *Address-Decoding Mask and Enable.*

Mask Value	Valid For PDARs	Description
0x0	All except INTCS	Physical Address decode OFF
0x1:0x3	All except INTCS	*reserved*/OFF
0x4	All except INTCS	4 Address bits SysAD[35:32] are masked and compared (4GB address space).
0x5	All except INTCS	5 Address bits SysAD[35:31] are masked and compared (2GB address space).
0x6	All except INTCS	6 Address bits SysAD[35:30] are masked and compared (1GB address space).
0x7	All except INTCS	7 Address bits SysAD[35:29] are masked and compared (512MB address space).
0x8	All except INTCS	8 Address bits SysAD[35:28] are masked and compared (256MB address space).
0x9	All except INTCS	9 Address bits SysAD[35:27] are masked and compared (128MB address space).
0xA	All except INTCS	10 Address bits SysAD[35:26] are masked and compared (64MB address space).
0xB	All except INTCS	11 Address bits SysAD[35:25] are masked and compared (32MB address space).
0xC	All except INTCS	12 Address bits SysAD[35:24] are masked and compared (16MB address space).
0xD	All except INTCS	13 Address bits SysAD[35:23] are masked and compared (8MB address space).
0xE	All except INTCS	14 Address bits SysAD[35:22] are masked and compared (4MB address space).
0xF	All PDARs	15 Address bits SysAD[35:21] are masked and compared (2MB address space).

This field specifies the number of high-order SysAD[35:21] address bits to be masked and compared with the ADDR field during the decoding of this device's base address. The field resets to 0x0 for all PDARs except INTCS and BOOTCS, both of which reset to 0xF. The value of this field for the INTCS PDAR always reads as 0xF and cannot be changed. Some values of this field disable address decoding. See Section 5.4.3 for a PDAR address-translation example.

Bit 4 MEM/LOC *Memory or Local Bus Chip-Selects.*
1 = Memory Bus.
0 = Local Bus.
Valid for DCS[8:2] and BOOTCS only. Resets to 0 for DCS[8:2]. Software sets the MEM/LOC bit to tell controller which port a physical device is accessible from. At reset, the BOOTCS PDAR’s MEM/LOC bit is set to 1 if the PCI64# signal is asserted, and it is cleared to 0 if PCI64# is negated; there is a Serial Mode EEPROM bit that can override this default (Section 12.4).
The timing characteristics of chip-selects configured for the Local Bus are specified in the Local Bus Chip-Select Timing Registers (LCSTn), Section 8.6.2, and the Local Boot Chip-Select Timing Register (BCST), Section 8.6.5.

Bit 5 VISPCI *Visible on PCI Bus.*
1 = visible.
0 = not visible.
Resets to 0 for all except INTCS and BOOTCS, which reset to 1. Software must set this bit to 1 to allow PCI accesses to this device.

Bit 7:6 WIDTH *Data Width of Physical Device.*

Width Value	Description
0x0	Physical device data width 8-bits
0x1	Physical device data width 16-bits
0x2	Physical device data width 32-bits
0x3	Physical device data width 64-bits

Writable for DCS[8:2] and BOOTCS. The width value for INTCS, SDRAM0, SDRAM1, PCIW0, and PCIW1 are fixed at 0x3 (64 bits). BOOTCS resets as 0x0 and has Serial Mode EEPROM bits (Section 12.4) that configure its width. Resets to 0x0 for all others.

Bit 20:8 *reserved* Hardwired to 0.

Bit 35:21 ADDR *Physical Address For This Device.*
These bits, after masking with the value of the MASK field, are compared with the incoming high-order address bits to decode the physical address space of this device. This field resets to 0x0 for all PDARs, except INTCS and BOOTCS. See Section 5.4.3 for a PDAR address-translation example.
In a multi-controller configuration (Section 5.3), the reset values for the INTCS and BOOTCS registers change, based on the controller ID. If this is the Main

Controller (controller ID 0x0), the ADDR field of the INTCS and BOOTCS registers resets to 0x0 1FA and 0x0 1FC, respectively. If this is not the Main Controller, the ADDR field of the INTCS and BOOTCS registers resets to the value shown below:

Controller ID	INTCS Base Address Indicated by ADDR field of INTCS PDAR at Reset	BOOTCS Base Address Indicated by ADDR field of BOOTCS PDAR at Reset
0x0 (Main Controller)	0x0 1FA0 0000	0x0 1FC0 0000
0x1	0x0 1F80 0000	0x0 0000 0000
0x2	0x0 1F60 0000	0x0 0000 0000
0x3	0x0 1F40 0000	0x0 0000 0000

Bit 63:36 *reserved* Hardwired to 0.

5.4.3 PDAR Address Decoding Example

When the CPU generates a 36-bit physical address, the ADDR and MASK fields of the PDARs determine where that access goes. For example, suppose that the CPU address range 0x0_2000_0000 through 0x0_3FFF_FFFF should go to SDRAM Bank 0 (the SDRAM0 PDAR). This is equivalent to saying that when the SysAD[35:29] address bits are b0000_001, the access should go to SDRAM Bank 0.

In this example, the MASK and ADDR fields of the PDAR for SDRAM Bank 0 would be programmed as follows:

- MASK = 0x7, which means only compare [35:29]
- ADDR[35:21] = b0000_001x_xxxx_xxx

The value in the ADDR field specifies the high-order address bits that must match the incoming address, and the MASK field specifies which ADDR bits are to be used in the address comparison. In this example, a MASK value of 0x7 means only compare [35:29], and this is equivalent to a mask of b1111_1110_0000_000. Thus, when the CPU generates an access to SDRAM Bank 0, the controller uses only address bits SysAD[28:0] from the CPU. These are the bits that were not masked by the SDRAM0 PDAR.

Figure 10: PDAR Address Decoding Example

35														21	20	0	
0	0	0	0	0	0	1	x	x	x	x	x	x	x	x	▨	▨	ADDR field of PDAR
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	▨	▨	MASK field of PDAR
0	0	0	0	0	0	1	SysAD[28:0]										Address sent to SDRAM0

5.5
CPU Interface Registers

Table 15: CPU Interface Registers

Register	Symbol	Offset	R/W	Reset Value	Description
CPU Status	CPUSTAT	0x0080	R/W	0x0000 0000 0000 0h00 [a]	Miscellaneous CPU status and control.
Interrupt Control	INTCTRL	0x0088	R/W	0x8888 8888 8888 8888	Interrupt enable and priority.
Interrupt Status 0	INTSTAT0	0x0090	R	0x0000 0000 0000 0000	Interrupt status 0.
Interrupt Status 1 and CPU Interrupt Enable	INTSTAT1	0x0098	R/W	0x0001 0000 0000 0000	Interrupt status 1 and CPU interrupt enable.
Interrupt Clear	INTCLR	0x00A0	R/W	0x0000 0000 0000 0000	Interrupt clear.
PCI Interrupt Control	INTPPES	0x00A8	R/W	0x0000 0000 0000 0000	Interrupt input signals polarity and edge vs. level sensitivity control.

[a] The “h” nibble in this reset value changes in multi-controller configurations (Section 5.3).

5.5.1
CPU Status Register (CPUSTAT)

The TMODE field (bits 5:4) of this register should be programmed as soon as possible after reset. See Section 5.3.3 for details.

Bit 0 CLDRST *Cold Reset.*
1 = cold-reset the entire system.
0 = writing 0 has no effect; the bit always reads 0.
Setting this bit resets the controller, causes it to assert ColdReset# to the CPU, and (if PCICR# is asserted) causes it to assert PCIRST# on the PCI Bus. The same actions that are performed by setting this bit are also performed during power-up (VccOk input asserted). Compare the PCICRST field (bit 63) of the PCI Control Register (PCICTRL), Section 7.11.1.

Bit 1 WARMRST *Warm Reset.*
1 = warm-reset the CPU.
0 = clear the 1.
Setting this bit causes the controller to assert Reset# to the CPU. The controller and PCI devices are not reset. After a warm reset, this bit reads 1. The bit can be cleared by writing 0. Compare the PCIWRST field (bit 62) of the PCI Control Register (PCICTRL), Section 7.11.1.

Bit 2 DISPC *Disable Parity Checking.*
1 = disable parity checking of CPU write data.
0 = enable parity checking of CPU write data.
The controller checks even parity when this bit is cleared. See Section 5.2.4 for a description of CPU-related parity generation and checking.

Bit 3 DISCPUPC *Disable Parity Generation.*
1 = disable parity generation for CPU reads.
0 = enable parity generation for CPU reads.
The controller generates even parity when this bit is cleared. See Section 5.2.4 for a description of CPU-related parity generation and checking.

Bits 5:4 TMODE *No-Target Read Response.*

TMODE Value	Description
0x0	Single-controller configuration. Return zeros with bad parity.
0x1	Single-controller configuration. Return zeros with good even parity.
0x2	Multi-controller configuration. If this is the Main Controller, return zeros with bad parity after time-out.
0x3	Multi-controller configuration. If this is the Main Controller, return zeros with good even parity after time-out.

This field determines how the controller responds to a CPU read request that (a) does not decode to this controller's address space, as specified by this controller's PDARs (Section 5.4), (b) encounters a timeout of the CPU-Bus Read Timer (Section 5.6.4), or (c) encounters a timeout of the Local-Bus Ready Timer (SUBSCWID plus CONWID fields of the Local Bus Chip Select Registers, LCSTn, Section 8.6.2). If a TMODE value of 2 or 3 is programmed, the CPU-Bus Read Timer must be initialized in order for this feature to work. Failure to load a value into this timer causes the system to hang for a CPU read request to which no controller responds. The TMODE field should be programmed as soon as possible after reset. See Section 5.3.3 for details.

Bits 7:6 *reserved* Hardwired to 0.

Bits 9:8 CTRLNUM *Controller ID Number.* (read-only)

Controller ID	Base Address of Registers
00	0x1FA0_0000 (Main Controller)
01	0x1F80_0000
10	0x1F60_0000
11	0x1F40_0000

These read-only bits are set by the UART_DTR# and UART_TxDRDY# signals during cold reset (Section

12.0) and determine the controller's address space in a multi-controller configuration (Section 5.3).

Bit 10 MAINCTRL *Main Controller.* (read-only)
1 = Main Controller.
0 = not the Main Controller.

Bits 63:11 *reserved* Hardwired to 0.

5.5.2 Interrupt Control Register (INTCTRL)

Section 11.0 on page 147 gives an overview of interrupts. Each nibble of the INTCTRL register contains the enable bit and priority bits for each source of a controller interrupt. The three least-significant bits of each nibble contain the interrupt priority assignment. The most-significant bit of the nibble contains the interrupt enable. When set, the interrupting source is enabled to interrupt the CPU. Encodings 0 through 5 correspond to CPU interrupts 0 through 5. Encoding 6 corresponds to the CPU's NMI# input.

Bits 2:0 CPCEPRI *CPU-Interface Parity-Error Interrupt Priority.*

Priority Value	Description
0x0	This interrupt is assigned to CPU interrupt level 0, Int#[0].
0x1	This interrupt is assigned to CPU interrupt level 1, Int#[1].
0x2	This interrupt is assigned to CPU interrupt level 2, Int#[2].
0x3	This interrupt is assigned to CPU interrupt level 3, Int#[3].
0x4	This interrupt is assigned to CPU interrupt level 4, Int#[4].
0x5	This interrupt is assigned to CPU interrupt level 5, Int#[5].
0x6	This interrupt is assigned to CPU non-maskable interrupt, NMI#.
0x7	*reserved*

Bit 3 CPCEEN *CPU-Interface Parity-Error Interrupt Enable.*
1 = enable.
0 = disable.

Bits 6:4 CNTDPRI *CPU No-Target Decode Interrupt Priority.*
Same values as Bits 2:0.

Bit 7 CNTDEN *CPU No-Target Decode Interrupt Enable.*
1 = enable.
0 = disable.

Bits 10:8 MCEPRI *Memory-Check Error Interrupt Priority.*
Same values as Bits 2:0.

Bit 11 MCEEN *Memory-Check Error Interrupt Enable*.
1 = enable.
0 = disable.
The error-checking mode (parity or ECC) is specified

		in the Memory Control Register (MEMCTRL), Section 6.6.1. The type of memory-check error is reported in the Memory Check Error Status Register (CHKERR), Section 6.6.3.
Bits 14:12	DMAPRI	*DMA Controller Interrupt Priority.* Same values as Bits 2:0.
Bit 15	DMAEN	*DMA Controller Interrupt Enable*. 1 = enable all DMA interrupt sources. 0 = disable all DMA interrupt sources. This bit is a global enable for the DMA interrupt sources that are individually enabled by the IE bit in the DMA Control Registers 0 and 1 (DMACTRLn), Section 9.5.1. Clearing all bits in DMACTRLn is the same as clearing the DMAEN bit.
Bits 18:16	UARTPRI	*UART Interrupt Priority.* Same values as Bits 2:0.
Bit 19	UARTEN	*Global UART-Interrupt Enable*. 1 = enable all UART interrupt sources. 0 = disable all UART interrupt sources. This bit is a global enable for all UART interrupt sources that are individually enabled in the UART Interrupt Enable Register (UARTIER), Section 10.4.3. Clearing all bits in UARTIER is the same as clearing the UARTEN bit.
Bits 22:20	WDOGPRI	*Watchdog Timer Interrupt Priority*. Same values as Bits 2:0.
Bit 23	WDOGEN	*Watchdog Timer Interrupt Enable*. 1 = enable. 0 = disable. Setting this bit enables interrupts on time-out of Timer 3 Counter Register (T3CNTR, Section 5.6.8).
Bits 26:24	GPTPRI	*General-Purpose Timer Interrupt Priority.* Same values as Bits 2:0.
Bit 27	GPTDEN	*General-Purpose Timer Interrupt Enable*. 1 = enable. 0 = disable. Setting this bit enables interrupts on time-out of Timer 2 Counter Register (T2CNTR, Section 5.6.6).
Bits 30:28	LBRTDPRI	*Local-Bus Ready Timer Interrupt Priority.* Same values as Bits 2:0. This bit enables interrupts on time-out of the 12-bit counter in the SUBSCWID+CONWID fields of the Local Bus Chip Select Registers (LCSTn, Section 8.6.2). This interrupt applies to both reads and writes on the Local Bus.

Bit 31	LBRTDEN	*Local-Bus Ready Interrupt Enable*. 1 = enable. 0 = disable.
Bits 34:32	INTAPRI	*PCI Interrupt Signal INTA# Priority.* Same values as Bits 2:0.
Bit 35	INTAEN	*PCI Interrupt Signal INTA# Enable*. 1 = enable. 0 = disable.
Bits 38:36	INTBPRI	*PCI Interrupt Signal INTB# Priority*. Same values as Bits 2:0.
Bit 39	INTBEN	*PCI Interrupt Signal INTB# Enable*. 1 = enable. 0 = disable.
Bits 42:40	INTCPRI	*PCI Interrupt Signal INTC# Priority*. Same values as Bits 2:0.
Bit 43	INTCEN	*PCI Interrupt Signal INTC# Enable*. 1 = enable. 0 = disable.
Bits 46:44	INTDPRI	*PCI Interrupt Signal INTD# Priority*. Same values as Bits 2:0.
Bit 47	INTDEN	*PCI Interrupt Signal INTD# Enable*. 1 = enable. 0 = disable.
Bits 50:48	INTEPRI	*PCI Interrupt Signal INTE# Priority*. Same values as Bits 2:0.
Bit 51	INTEEN	*PCI Interrupt Signal INTE# Enable*. 1 = enable. 0 = disable.
Bits 55:52	*reserved*	Hardwired to 0.
Bits 58:56	PCISPRI	*PCI SERR# Interrupt Priority*. Same values as Bits 2:0.
Bit 59	PCISEN	*PCI SERR# Interrupt Enable*. 1 = enable. 0 = disable.
Bits 62:60	PCIEPRI	*PCI Internal Error Interrupt Priority.* Same values as Bits 2:0. This field sets the priority of interrupts enabled by the PCIEEN bit, described immediately below. A PCI Internal Error indicates that something bad happened during a PCI transaction; the fault could lie either with the PCI device or the controller.
Bit 63	PCIEEN	*PCI Internal Error Interrupt Enable.* 1 = enable.

0 = disable.
This bit is a global enable for all PCI interrupt sources that are individually enabled by bits 53:48 of the PCI Control Register (PCICTRL), Section 7.11.1. See PCI-Master Parity Detection (Section 7.4.5) and PCI-Target Parity Detection (Section 7.5.4) for details.

5.5.3
Interrupt Status Register 0 (INTSTAT0)

Each 16-bit halfword of this register indicates which of the 16 sources are requesting an interrupt for the lower four non-maskable interrupts to the CPU, Int#[3:0].

Bits 15:0 IL0STAT *CPU Int#[0] Status*.

Status Bit	Source of Interrupt
0	CPU parity-check error (even parity)
1	CPU no-target decode
2	Memory Check Error
3	DMA Controller
4	UART
5	Watchdog timer
6	General-purpose timer
7	Local Bus read time-out
12:8	PCI interrupts INTE# through INTA#
13	*reserved*
14	PCI SERR#
15	PCI internal error.

Bits 31-16 IL1STAT *CPU Int#[1] Status*.
Same values as Bits 15:0.

Bits 47:32 IL2STAT *CPU Int#[2] Status*.
Same values as Bits 15:0.

Bits 63:48 IL3STAT *CPU Int#[3] Status*.
Same values as Bits 15:0.

5.5.4
Interrupt Status 1/CPU Interrupt Enable Register (INTSTAT1)

Each of the lower three 16-bit halfwords of this register indicates which of the 16 sources are requesting an interrupt for the upper two non-maskable interrupts to the CPU, Int#[5:4], and the non-maskable interrupt to the CPU, NMI#. The upper portion of this register has the enables for controller interrupt output buffers.

Bits 15:0 IL4STAT *CPU Int#[4] Status.*
Same values as Bits 15:0 of INTSTAT0 register.

Bits 31-16 IL5STAT *CPU Int#[5] Status.*
Same values as Bits 15:0 of INTSTAT0 register.

Bits 47:32	NMISTAT	*CPU NMI# Status.* Same values as Bits 15:0 of INTSTAT0 register.
Bit 48	IL0OE	*Int#[0] Controller Output Enable*. 1 = enable. 0 = disable.
Bit 49	IL1OE	*Int#[1] Controller Output Enable*. enable. 1 = enable. 0 = disable.
Bit 50	IL2OE	*Int#[2] Controller Output Enable*. enable. 1 = enable. 0 = disable.
Bit 51	IL3OE	*Int#[3] Controller Output Enable*. enable. 1 = enable. 0 = disable.
Bit 52	IL4OE	*Int#[4] Controller Output Enable*. enable. 1 = enable. 0 = disable.
Bit 53	IL5OE	*Int#[5] Controller Output Enable*. enable. 1 = enable. 0 = disable.
Bit 54	NMIOE	*NMI# Controller Output Enable*. buffer enable. 1 = enable. 0 = disable.
Bit 63:55	*reserved*	Hardwired to 0.

5.5.5 Interrupt Clear Register (INTCLR)

Writing a 1 to any of these bits causes the corresponding interrupt source to be cleared, but only if the interrupt is edge-triggered. All of the controller’s interrupt outputs to the CPU, Int[5:0], are level-sensitive. PCI interrupts can be specified as level-sensitive or edge-triggered in the PCI Interrupt Control Register (INTPPES), Section 5.5.6. Writing a 0 to any of these bits has no effect. The bits always read 0.

Bit 15:0 ISCLR *Clear Interrupt.*

Clear Bit	Source of Interrupt
0	CPU parity-check error (even parity)
1	CPU no-target decode
2	Memory-Check Error
3	DMA Controller
4	UART
5	Watchdog timer
6	General-purpose timer
7	Local Bus read time-out
12:8	PCI interrupt signals, INTE# through INTA#
13	*reserved* (always 0)
14	PCI SERR#
15	PCI internal error

1 = clear interrupt.
0 = writing 0 has no effect. The bits always read 0.

Bit 63:16 *reserved* Hardwired to 0.

5.5.6 PCI Interrupt Control Register (INTPPES)

Bit 0 INTAPOL *INTA# Signal Polarity.*
1 = active-Low.
0 = active-High (reset value).

Bit 1 INTAEDGE *INTA# Signal Edge.*
1 = level-sensitive.
0 = edge-triggered (reset value).

Bit 2 INTBPOL *INTB# Signal Polarity.*
1 = active-Low.
0 = active-High (reset value).

Bit 3 INTBEDGE *INTB# Signal Edge.*
1 = level-sensitive.
0 = edge-triggered (reset value).

Bit 4 INTCPOL *INTC# Signal Polarity.*
1 = active-Low.
0 = active-High (reset value).

Bit 5 INTCEDGE *INTC# Signal Edge.*
1 = level-sensitive.
0 = edge-triggered (reset value).

Bit 6 INTDPOL *INTD# Signal Polarity.*
1 = active-Low.
0 = active-High (reset value).

Bit 7 INTDEDGE *INTD# Signal Edge.*
1 = level-sensitive.
0 = edge-triggered (reset value).

Bit 8 INTEPOL *INTE# Signal Polarity.*
1 = active-Low.
0 = active-High (reset value).

Bit 9 INTEEDGE *INTE# Signal Edge.*
1 = level-sensitive.
0 = edge-triggered (reset value).

Bit 63:10 *reserved* Hardwired to 0.

5.6 Timer Registers

The controller has four timers:

- *SDRAM Refresh Timer (Timer0):* A 16-bit timer that causes an SDRAM refresh when it expires. The controller automatically reloads this free-running timer.
- *CPU-Bus Read Timer (Timer1):* A 24-bit timer used to determine CPU bus read time-outs in a multi-controller configuration. See the description of the TMODE field in the CPU Status Register (CPUSTAT, Section 5.5.1). When a CPU read begins, this timer is automatically loaded and begins to count, if enabled.
- *General-Purpose Timer (Timer2):* A 32-bit timer that generates a CPU interrupt when it expires, if the interrupt is enabled in the Interrupt Control Register (INTCTRL, Section 5.5.2). The controller automatically reloads this free-running timer.
- *Watchdog Timer (Timer3):* A 32-bit timer that generates a CPU interrupt when it expires, if the interrupt is enabled in the Interrupt Control Register (INTCTRL, Section 5.5.2). The controller automatically reloads this free-running timer.

Normally these timers count SysClock ticks, but one timer can be specified as a prescale input to another timer. To be used as a prescaler, the timer must be enabled.

The Local-Bus also has a ready-signal timer. This is configured in the Local Bus Chip-Select Timing Registers (LCSTn), Section 8.6.2.

Table 16: Timer Registers

Register	Symbol	Offset	R/W	Reset Value	Description
SDRAM Refresh Control	T0CTRL	0x01C0	R/W	0x0000 0001 0000 0186	SDRAM refresh control.
SDRAM Refresh Counter	T0CNTR	0x01C8	R/W	0x0000 0000 0000 0000	SDRAM refresh counter.
CPU-Bus Read Time-Out Control	T1CTRL	0x01D0	R/W	0x0000 0000 0000 0000	CPU-bus read time-out control.
CPU-Bus Read Time-Out Counter	T1CNTR	0x01D8	R/W	0x0000 0000 0000 0000	CPU-bus read time-out counter.
General-Purpose Timer Control	T2CTRL	0x01E0	R/W	0x0000 0000 0000 0000	General-purpose timer control.
General-Purpose Timer Counter	T2CNTR	0x01E8	R/W	0x0000 0000 0000 0000	General-purpose timer counter.
Watchdog Timer Control	T3CTRL	0x01F0	R/W	0x0000 0000 0000 0000	Watchdog timer control.
Watchdog Timer Counter	T3CNTR	0x01F8	R/W	0x0000 0000 0000 0000	Watchdog timer counter.

5.6.1 SDRAM Refresh Control Register (T0CTRL)

This register is initialized to 0x1 0000 0186 at reset.

Bits 15:0 T0RLVAL *Timer 0 Refresh Counter Reload Value.*
This value, in SysClock ticks, is automatically reloaded into the refresh counter after the counter reaches zero. The refresh counter counts down from

this value. Thus, the time of the count cycle corresponds to 1 plus this register’s value. The default value (0x186 = 390) is the refresh rate for an SDRAM chip that requires 4096 refresh cycles every 32 ms (i.e., one refresh every 7.8125 microseconds) for SysClock running at 50 MHz. This is very conservative but it allows for successful boot, after which the reload value can be increased.

Bits 31-16	*reserved*	Hardwired to 0.
Bit 32	T0EN	*Timer 0 Enable*. 1 = enable (reset value). 0 = disable. Enabling the timer starts it counting.
Bit 33	T0PREN	*Timer 0 Prescale Enable*. 1 = enable. 0 = disable (reset value). If the prescaler is enabled, the controller only starts counting when the prescaler reaches zero.
Bits 35:34	T0PRSRC	*Timer 0 Prescale Source*. 00 = *reserved* 01 = Timer 1. 10 = Timer 2. 11 = Timer 3.
Bits 63:36	*reserved*	Hardwired to 0.

5.6.2 SDRAM Refresh Counter Register (T0CNTR)

Bits 15:0	T0VAL	*Timer 0 Current Timer Value*. The timer value, in SysClock ticks. Refresh is generated upon reaching 0.
Bits 63:16	*reserved*	Hardwired to 0.

5.6.3 CPU-Bus Read Time-Out Control Register (T1CTRL)

This timer is used to time-out CPU read requests to which no resource responds. The timer functions differently, depending on whether the TMODE field in the CPU Status Register (CPUSTAT), Section 5.5.1, specifies a single-controller or multi-controller configuration and whether the MAINCTRL field in CPUSTAT specifies a Main Controller.

Bits 23:0	T1RLVAL	*Timer 1 Counter Reload Value*. If the TMODE field in CPUSTAT specifies a single-controller configuration (TMODE = 0x0 or 0x1) or if this controller is not the Main Controller in a multi-controller configuration (TMODE = 0x2 or 0x3 and MAINCTRL = 0), the timer is free-running: the T1RLVAL value, in SysClock ticks, is automatically re-loaded into the counter after the counter reaches

zero. The counter starts counting when it is enabled by the T1EN bit and counts down from this value. Thus, the time of the count cycle corresponds to 1 plus this register’s value.
If the TMODE field in CPUSTAT specifies a multi-controller configuration and this is the Main Controller (TMODE = 0x2 or 0x3 and MAINCTRL = 1), the T1RLVAL value is loaded at the beginning of any CPU read cycle, and the counter only counts while the read is in progress.

Bits 31:24	*reserved*	Hardwired to 0.
Bit 32	T1EN	*Timer 1 Enable*. 1 = enable. 0 = disable (reset value). Enabling the timer starts it counting.
Bit 33	T1PREN	*Timer 1 Prescale Enable*. 1 = enable. 0 = disable. If the prescaler is enabled, the controller only starts counting when the prescaler reaches zero.
Bits 35:34	T1PRSC	*Timer 1 Prescale Source*. 00 = Timer 0. 01 = *reserved* 10 = Timer 2. 11 = Timer 3.
Bits 63:36	*reserved*	Hardwired to 0.

5.6.4 CPU-Bus Read Time-Out Counter Register (T1CNTR)

Bits 23:0	T1VAL	*Timer 1 Current Timer Value*. A CPU-bus read time-out is generated when this value, in SysClock ticks, reaches 0. CPU-bus read time-outs are controlled by the TMODE bits of the CPU Status Register (Section 5.5.1). The T1VAL value should be greater than the worst-case response time required by your slowest device; if a slow device returns data after the T1VAL time-out, the state of hardware may become corrupt, requiring a reset.
Bits 63:24	*reserved*	Hardwired to 0.

5.6.5 General-Purpose Timer Control Register (T2CTRL)

Bits 31:0	T2RLVAL	*Timer 2 Counter Reload Value*. This timer is free-running: the T2RLVAL value, in SysClock ticks, is automatically re-loaded into the counter after the counter reaches zero. The counter starts counting when it is enabled by the T2EN bit and counts down from this value. Thus, the time of

		the count cycle corresponds to 1 plus this register's value.
Bit 32	T2EN	*Timer 2 Enable*. 1 = enable. 0 = disable (reset value). Enabling the timer starts it counting.
Bit 33	T2PREN	*Timer 2 Prescale Enable*. 1 = enable. 0 = disable. If the prescaler is enabled, the controller only starts counting when the prescaler reaches zero.
Bits 35:34	T2PRSRC	*Timer 2 Prescale Source*. 00 = Timer 0. 01 = *reserved* 10 = Timer 2. 11 = Timer 3.
Bits 63:36	*reserved*	Hardwired to 0.

5.6.6 General-Purpose Timer Counter Register (T2CNTR)

Bits 31:0	T2VAL	*Timer 2 Current Timer Value*. The general-purpose timer interrupt is generated upon reaching 0, if this interrupt has been enabled by setting the GPTDEN field of the Interrupt Control Register (Section 5.5.2).
Bits 63:32	*reserved*	Hardwired to 0.

5.6.7 Watchdog Timer Control Register (T3CTRL)

Bits 31:0	T3RLVAL	*Timer 3 Counter Reload Value*. This timer is free-running: the T3RLVAL value, in SysClock ticks, is automatically re-loaded into the counter after the counter reaches zero. The counter starts counting when it is enabled by the T3EN bit and counts down from this value. Thus, the time of the count cycle corresponds to 1 plus this register's value.
Bit 32	T3EN	*Timer 3 Enable*. 1 = enable. 0 = disable (reset value). Enabling the timer starts it counting.
Bit 33	T3PREN	*Timer 3 Prescale Enable*. 1 = enable. 0 = disable. If the prescaler is enabled, the controller only starts counting when the prescaler reaches zero.
Bits 35:34	T3PRSRC	*Timer 3 Prescale Source*. 00 = Timer 0.

		01 = Timer 1. 10 = Timer 2. 11 = *reserved*
Bits 63:36	*reserved*	Hardwired to 0.

5.6.8
Watchdog Timer Counter Register (T3CNTR)

Bits 31:0	T3VAL	*Timer 3 Current Timer Value*. The watchdog timer interrupt is generated upon reaching 0, if this interrupt has been enabled by setting the WDOGEN field of the Interrupt Control Register (Section 5.5.2).
Bits 63:32	*reserved*	Hardwired to 0.

6.0 Main-Memory Interface and Registers

The controller’s memory interface runs at the SysClock frequency (up to 100 MHz). It supports dword and block (burst) accesses to one or two physical banks of SDRAM main memory. Several types of SDRAM chips are supported (each with two to four on-chip virtual banks). A 16-bit counter supports programmable refresh rates.

The CPU, PCI-Bus masters, and Local-Bus masters can access SDRAM memory directly in the system memory space, or they can configure the controller’s DMA registers for DMA transfers, as described in Section 9.0. Memory accesses by the CPU, PCI-Bus masters, and DMA cause the memory interface to prefetch. The controller prioritizes requests on the basis of which physical memory bank is currently open, and thus which request can be serviced most quickly.

The controller generates and checks byte-wide parity or ECC (single-error correction, double-error detection) with 64+8 bits of SDRAM. There is no performance penalty for this generation and checking. The controller also supports Flash or ROM devices for boot or other memory spaces.

6.1 Memory Configuration and Monitoring

Software configures and monitors the memory interface using the following registers:

- ❑ Physical Device Address Registers (PDARs), Section 5.4 on page 45.
- ❑ Interrupt Control Register (INTCTRL), Section 5.5.2 on page 52.
- ❑ Interrupt Status Register 0 (INTSTAT0), Section 5.5.3 on page 55.
- ❑ Interrupt Status 1/CPU Interrupt Enable Register (INTSTAT1), Section 5.5.4 on page 55.
- ❑ Interrupt Clear Register (INTCLR), Section 5.5.5 on page 56.
- ❑ SDRAM Refresh Control Register (T0CTRL), Section 5.6.1 on page 58.
- ❑ SDRAM Refresh Counter Register (T0CNTR), Section 5.6.2 on page 59.
- ❑ Memory-Interface Registers, Section 6.6 on page 72.

6.2 Physical Loads

At 100 MHz, the memory interface typically supports up to three physical loads on each data bit, depending on the quality of board layout and the electrical characteristics of the devices used (including any sockets). These loads can be any three of the following address-decode ranges specified by the Physical Device Address Registers described in Section 5.4:

- ❑ SDRAM Bank 0 (PDAR = SDRAM0)
- ❑ SDRAM Bank 1 (PDAR = SDRAM1)
- ❑ Boot ROM (PDAR = BOOTCS, sometimes referred to as Memory Bank 2)
- ❑ Memory or Local-Bus Device 8 (PDAR = DCS8)
- ❑ Memory or Local-Bus Device 7 (PDAR = DCS7)
- ❑ Memory or Local-Bus Device 6 (PDAR = DCS6)
- ❑ Memory or Local-Bus Device 5 (PDAR = DCS5)
- ❑ Memory or Local-Bus Device 4 (PDAR = DCS4)

- ❑ Memory or Local-Bus Device 3 (PDAR = DCS3)
- ❑ Memory or Local-Bus Device 2 (PDAR = DCS2)
- ❑ A row of transceiver/buffers with DCS[8:2] and BOOTCS behind it.

The SDRAM in the first two address ranges, SDRAM0 and SDRAM1, can be bank-interleaved. Any of the other address ranges can be populated with Flash or ROM memory.

When the controller is configured for 32-bit PCI operation (PCI64# negated), the boot memory and the seven DCS[8:2] devices can be individually configured by the MEM/LOC bit in the PDAR (Section 5.4) to appear on the memory bus or the Local Bus. When the controller is configured for 64-bit PCI operation (PCI64# asserted), these devices must be on the memory bus. The BOOTCS# and DCS#[8:2] chip-selects for these loads need not be buffered, because each of these bits supports only a single load. Figure 3 on page 14 shows an example of buffered loads on the memory bus.

If more than three loads are placed on the memory bus, the bus will slow down. Such configurations (e.g., Figure 4 on page 15) require either a slower SysClock or buffering on the data, address, and write-enable signals to those devices, so that the controller only sees three physical loads—two SDRAM loads and one additional load for the buffers.

6.3 Write FIFOs

The memory interface has the following internal write FIFOs, each of which holds 64 bytes of data plus associated address and command bits:

- ❑ 8-dword (64-byte) CPU Write FIFO.
- ❑ 8-dword (64-byte) PCI Write FIFO.
- ❑ 8-dword (64-byte) DMA Write FIFO.

These FIFOs are each capable of accepting writes at a maximum speed of 640MB/sec. For each of these sources, two addresses, one to each of the two physical banks, can be buffered. All of these addresses can be writes (single dword writes or 4-dword block writes), allowing up to six 4-dword block write requests (address and command) and data to be held in the three 8-dword write FIFOs.

6.4 Boot- ROM and External-Device Addressing

Boot ROM can be located either on the memory bus or on the Local Bus, as specified by Bit 256 of the controller initialization data (Section 12.4.2) and in the MEM/LOC bit of the BOOTCS PDAR (Section 5.4). If PCI64# asserted, indicating a 64-bit PCI Bus, Boot ROM must be located on the memory bus, because the Local Bus cannot exist when a 64-bit PCI Bus is implemented. If PCI64# negated at reset, the controller’s default CPU-initialization location for Boot ROM is the Local Bus.

The Boot ROM bus width is set by the Serial Mode EEPROM initialization data stream (Table 34 on page 154). If you have no Serial Mode EEPROM, the default width is 8 bits. You can change this after boot, but you cannot boot without a Serial Mode EEPROM if the default is wrong.

Flash memory may be used for Boot ROM. Boot memory, or a row of buffers bridging the controller to a secondary bank of memory or devices, is sometimes called the *third* *memory bank* or *memory bank 2*. The first and second physical banks, SDRAM0 and SDRAM1, are the *main-memory banks*.

6.4.1 Memory-Bus Addressing Of Boot ROM and External Devices

The address signals for the two main-memory physical banks, MAbank0[14:0] and MAbank1[14:0], may also be used to address Boot ROM and other devices located on the memory bus—those devices associated with the BOOTCS and DCS8:2 PDARs. The address signals for the two physical banks of main memory must be concatenated on the motherboard to obtain a linear 30-bit address for the boot memory or other devices. This 30-bit address bus can support devices up to 64 bits in width, as follows:

- Address bits 29:0 for 8-bit devices.
- Address bits 30:1 for 16-bit devices.
- Address bits 31:2 for 32-bit devices.
- Address bits 32:3 for 64-bit devices.

Figure 11 shows an example of a Flash device being used for Boot ROM, plus two physical banks of SDRAM main memory.

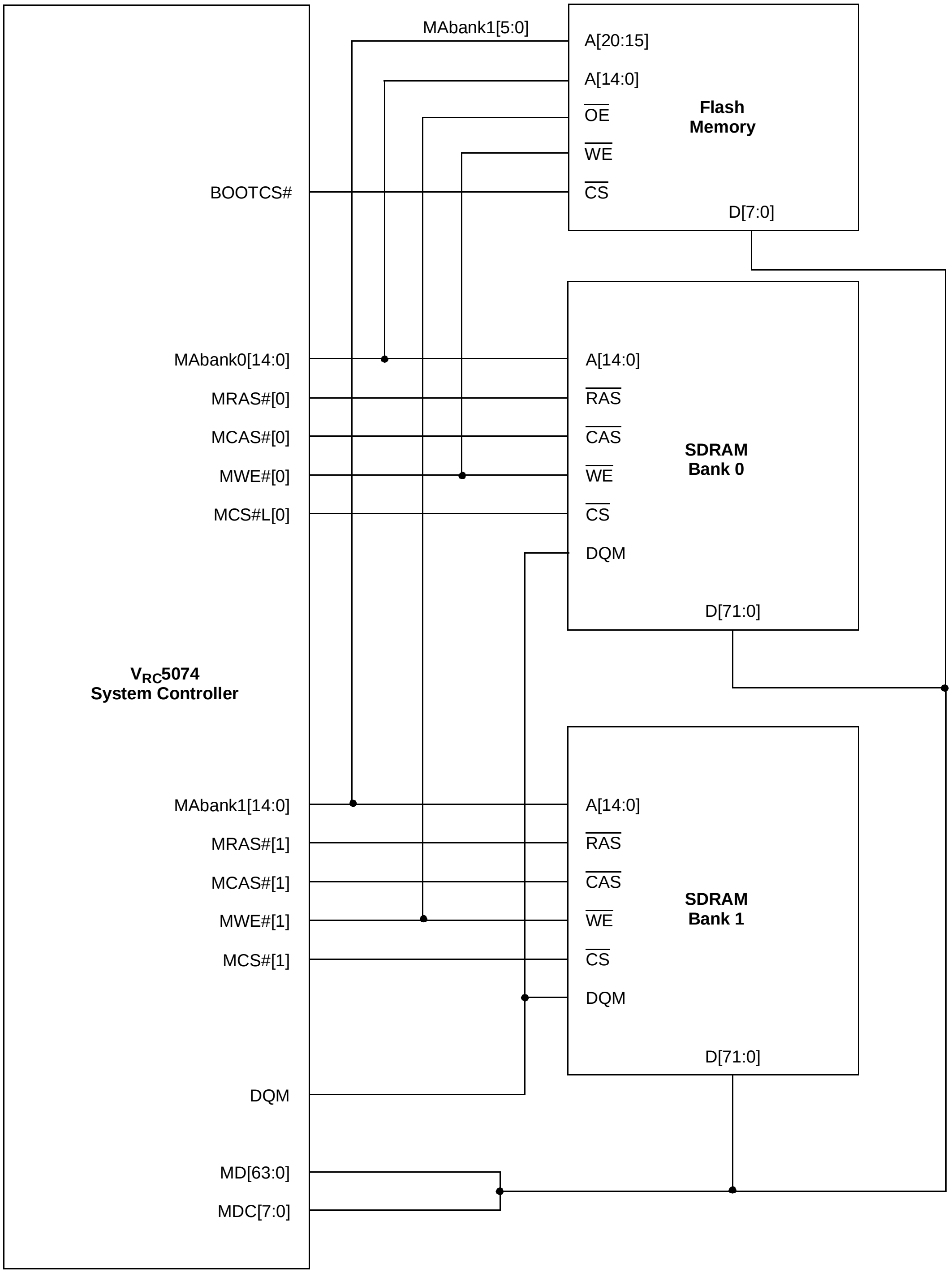
The selection of width for Boot ROM and other devices is controlled by the WIDTH field in the BOOTCS and DCS8:2 PDARs. The controller's MWE#[0] and MWE#[1] signals serve as the memory-chips' write-enable ($\overline{\text{WE}}$) and output-enable ($\overline{\text{OE}}$), respectively. The MCS#[1:0], MRAS#[1:0], and the MCAS#[1:0] signals are negated during accesses to Boot ROM and external devices, so as to disable the SDRAMs.

When BOOTCS and DCS8:2 devices are located on the Local Bus, rather than the memory bus, they are addressed with the LOC_AD[31:0] signals, as described in Section 8.0. Much greater timing control is available when DCS8:2 devices are located on the Local Bus rather than the Memory Bus. There is also potentially severe SDRAM-performance degradation during accesses to DCS8:2 devices that are located on the Memory Bus.

6.4.2 Boot-Memory Timing

Boot memory timing and the enabling of the MRDY# input signal are specified in the Memory Access Timing Register (ACSTIME), Section 6.6.2.

Figure 11: Bank-Interleaved SDRAM Main Memory With Flash Boot Memory



6.5 SDRAM Main Memory

Main memory consists of two physical banks of SDRAM configured by two PDARs, SDRAM0 and SDRAM1. Both banks support only SDRAM chips, and they both must be populated by the same type of SDRAM chip running at SysClock or greater from among the following types:

- ❑ *256 Mb, 4-bank*, including but not limited to:
  - 64M word x 4 bit x 4 bank
  - 32M word x 8 bit x 4 bank
  - 16M word x 16 bit x 4 bank
  - 8M word x 32 bit x 4 bank
- ❑ *64 Mb, 4-bank*, including but not limited to:
  - 16M word x 4 bit x 4 bank
  - 8M word x 8 bit x 4 bank
- ❑ *64 Mb, 2-bank*, including but not limited to:
  - 16M word x 4 bit x 2 bank
  - 8M word x 8 bit x 2 bank
- ❑ *16 Mb, 2-bank*, including but not limited to:
  - 4M word x 4 bit x 2 bank
  - 2M word x 8 bit x 2 bank
  - 1M word x 16 bit x 2 bank

The SDRAMTYP field of the MEMCTRL register (Section 6.6.1) specifies the type of SDRAM chips installed.

All SDRAM accesses are full-dword (64 bit) accesses. The controller internally implements partial-dword (less than 64-bit) write requests as read-merge-writes: it first reads from the write address, then merges the partial-dword write data into the read data, then writes the full dword to memory. Because of this, partial-dword writes take longer than full-dword writes.

6.5.1 Bank-Interleaving

In the context of systems containing SDRAM memory, the term *bank* has two different meanings. Sets of SDRAM chips connected to the controller’s MAbank0[14:0] and MAbank1[14:0] address signals are called *physical banks*, but individual SDRAM chips are organized internally into *virtual banks*.

- ❑ *Physical Banks:* Physical-bank interleaving provides a performance boost because accesses can be under way to both banks simultaneously. Unless otherwise stated, the term *bank-interleaving* refers to the interleaving of physical banks. Figure 11 shows a bank-interleaved SDRAM configuration with a Flash device for boot memory.
- ❑ *Virtual Banks:* The virtual banks internal to SDRAM chips can also provide some performance boost by allowing the controller to have multiple virtual banks open simultaneously (two or four virtual banks for each physical bank), thus providing a greater chance of a page hit on a memory access. In the list of SDRAM types at the beginning of Section 6.5, the “bank” in “256 Mb, 4-bank” refers to virtual banks, not physical banks.

The remaining discussion in this data sheet (except as noted) refers to physical banks, not virtual banks.

SDRAM in main memory can be located all in one physical bank (0 or 1) or in both physical banks. If both banks are populated, both must be populated with the same type of SDRAM chips, from among the types listed at the beginning of Section 6.5. The two physical banks can be bank-interleaved to improve performance. Bank-interleaving is specified in the ILEAVD filed of the MEMCTRL register (Section 6.6.1). If only one bank is populated, the other bank can, for example, be configured for field-upgrades by providing a SIMM or DIMM connector, as long as the SDRAM chips in both banks are of the same type.

If bank-interleaving is enabled in a two-bank configuration, the two address ranges defined by the PDARs for these banks (SDRAM0 and SDRAM1) are split between the two physical SDRAM rows, so that half of each physical row corresponds to one of the memory banks. If bank-interleaving is disabled in this configuration, the address ranges correspond to physical rows 0 and 1.

6.5.2
SDRAM Chip Initialization

The controller automatically configures the Mode Registers inside each SDRAM chip when the ENABLE bit is set in the Memory Control Register (MEMCTRL), Section 6.6.1. Accesses to SDRAM are held off until initialization is complete. The values written to the Mode Registers inside each SDRAM chip are fixed at:

- $\overline{\text{CAS}}$ latency = 3.
- Burst length = 4.
- Wrap type = interleaved.

These values will work with any type of 100 MHz SDRAM that uses 3-tick latency. The $\overline{\text{CAS}}$ latency is set to 3 because of the 100 MHz speed of the Memory Bus (a $\overline{\text{CAS}}$ latency of 2 can only be used with 66 MHz bus speed).

6.5.3
Direct Connections, SIMMs, and DIMMs

The controller is designed for direct connection to one or two banks of identical SDRAM chips, with no additional pipeline stages on the memory bus. SIMM or DIMM connectors can be used on either or both banks, but the SIMM or DIMM SDRAM chips must be identical, and the clock speed and board layout will determine whether external buffering is needed for the clock and control signals. SIMMs and DIMMs with external register stages are not supported.

6.5.4
Address-Multiplexing Modes

The controller supports address multiplexing by staggering the address lines, as shown in Table 17. The mapping is different, depending on whether or not the physical banks (SDRAM0 and SDRAM1 PDARs) are interleaved.

The physical banks are selected by the SysAD[13] signal. The virtual banks within each SDRAM chip are selected by the SysAD[12:11] signals. When physical-bank interleaving is implemented, four or eight virtual banks are visible to the controller (two or four virtual banks for each physical bank).

Table 17: MAbank-to-SysAD Address Mapping

MAbank Signals	Bank-Interleaved SysAD Mapping						Non-Bank-Interleaved SysAD Mapping					
	16Mb SDRAM		64Mb SDRAM		256Mb SDRAM		16Mb SDRAM		64Mb SDRAM		256Mb SDRAM	
	Row	Column	Row	Column	Row	Column	Row	Column	Row	Column	Row	Column
0	10	3	10	3	10	3	10	3	10	3	10	3
1	14	4	14	4	14	4	14	4	14	4	14	4
2	15	5	15	5	15	5	15	5	15	5	15	5
3	16	6	16	6	16	6	16	6	16	6	16	6
4	17	7	17	7	17	7	17	7	17	7	17	7
5	18	8	18	8	18	8	18	8	18	8	18	8
6	19	9	19	9	19	9	19	9	19	9	19	9
7	20	PDAR number [a]	20	PDAR number [a]	20	PDAR number [a]	20	13	20	13	20	13
8	21	23	21	25	21	26	21	23	21	25	21	26
9	22	24	22	26	22	27	22	24	22	26	22	27
10	11	b0	23	b0	23	b0	11	b0	23	b0	23	b0
11	12	12	24	27	24	28	12	12	24	27	24	28
12	b0	b0	11	11	25	29	b0	b0	11	11	25	29
13	12	12	12	12	11	11	12	12	12	12	11	11
14	11	11	11	11	12	12	11	11	11	11	12	12

a. SDRAM is 0 or 1, as determined by SysAD[13].

6.5.5

Performance

The speed of SDRAM memory accesses is determined by the type, speed and interleaving of the memory devices installed. Table 18 lists simulated memory performance for a 100 MHz memory bus.

Table 18: Main Memory Performance for 100 MHz SDRAM

Bypass Activation [a]	Bank-Inter-leaving	RAS Page Hit/Miss	R/W	CPU Access Type	Clocks [b]		Total Memory Bandwidth Used (MB/sec)	
					First Access	Second Access	Min	Max
CPU Access to Non-Bank-Interleaved Memory								
No	No	Hit	R	Single	13	11		
				Burst	11+4	11+4		
		Miss		Single	27	11		
				Burst	24+4	11+4		
		N/A	W	Single			368	400
				Burst			504	560
Yes	No	N/A	R	Single	16	11		
				Burst	16+4	11+4		
			W	Single			344	400
				Burst			520	560
CPU Access to Bank-Interleaved Memory								

Table 18: Main Memory Performance for 100 MHz SDRAM (continued)

Bypass Activation [a]	**Bank-Inter-leaving**	**RAS Page Hit/Miss**	**R/W**	**CPU Access Type**	**Clocks [b]**		**Total Memory Bandwidth Used (MB/sec)**	
					First Access	**Second Access**	**Min**	**Max**
No	No	N/A	R	Single	49	11		
				Burst	14+4	11+4		
			W	Single			150	160
				Burst			528	544
No	Yes	N/A	R	Single	20	11		
				Burst	18+4	11+4		
			W	Single			368	400
				Burst			504	560
DMA Reads and Writes to a Single Memory Bank, with CPU Accessing the Same Memory Bank								
No	No	N/A	R	Single	43	11		
				Burst	28+4	21+4		
			W	Single			320	400
				Burst			480	736
DMA Reads and Writes to Memory Bank 1, with CPU Accessing Memory Bank 0								
No	No	N/A	R	Single	22	16		
				Burst	76+4	62+4		
			W	Single			464	552
				Burst			688	784
DMA Writes to Memory Bank 1, with CPU Accessing Memory Bank 0								
No	No	N/A	R	Single	35	16		
				Burst	32+4	15+4		
			W	Single			424	512
				Burst			736	800
No	Yes	N/A	R	Single	43	10		
				Burst	23+4	13+4		
			W	Single			296	408
				Burst			480	736

[a] Bypass activation occurs when a CPU request arrives while the CPU and memory interfaces are both idle.

[b] For single-dword reads, this is the latency from the assertions of CPUValid# to CntrValid#. For burst reads this is given as *x+y* where *x* is the initial latency and *y* is the number of clocks to transfer the four dwords in the burst.

Accesses to SDRAM are all pipelined for high performance, and SDRAM bank-interleaving results in higher performance. However, the use of Flash memory or other non-SDRAM devices can reduce this performance, because the SDRAM pipeline stalls during the slower accesses to such memory. Flash memory can be placed on the memory bus or on the Local Bus, except that the Local Bus cannot be implemented while a 64-bit PCI Bus is implemented. If Flash memory devices must be placed on the memory bus (which would be the case in a 64-bit PCI Bus implementation), performance can be maximized by copying the contents of Flash, after boot, to SDRAM memory, and accessing from the SDRAM.

If a non-SDRAM device is on the memory bus, the worst-case latency from CPU to SDRAM occurs for is a block access (32-bytes) to a byte-wide device. This takes 32 individual accesses, each one lasting up to approximately 31 clocks. This gives a latency of close to 1000 clocks for the CPU.

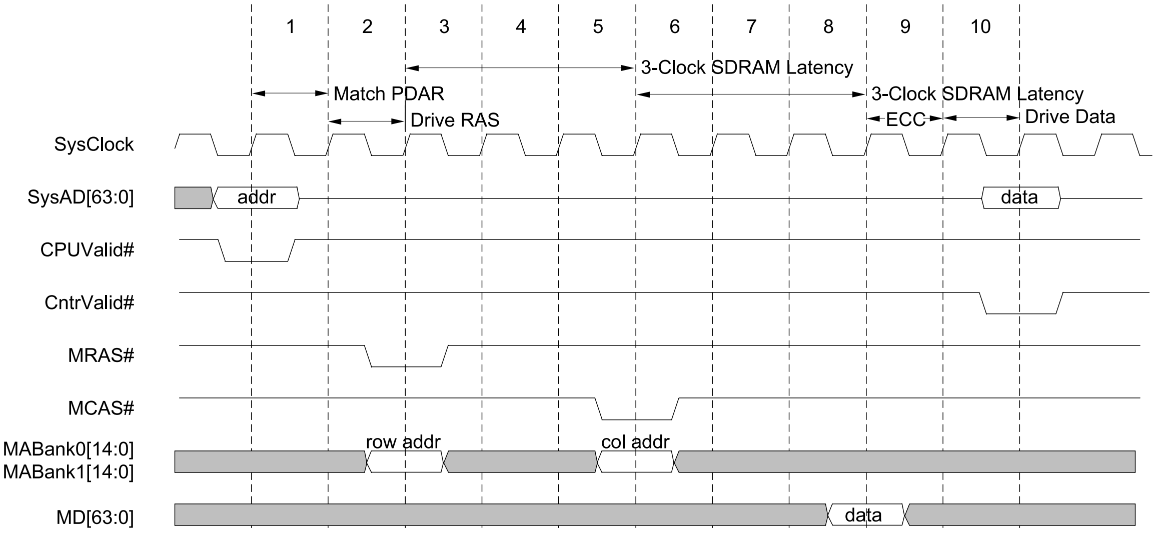
If only SDRAM is on the memory bus, the worst-case latency from CPU to SDRAM occurs when DMA and PCI keep memory busy. In this case, worst-case total latency from CPU to SDRAM is 76 clocks.

6.5.6 Memory Timing

The controller is designed for 100 MHz SDRAM chips using a 100 MHz SysClock. However, the board layout is critical. Some layouts may require faster SDRAM chips to ensure proper timing margins. The timing associated with SDRAMs is fixed by the system clock and a 3-clock $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ latency (3-clock latency from $\overline{\text{RAS}}$ to $\overline{\text{CAS}}$, and 3-clock latency from $\overline{\text{CAS}}$ to data on reads). Thus, SDRAM SIMMs with external register stages are not supported.

Figure 12 shows minimum timing of 10 clocks for a CPU read with bypass enabled and memory interface idle. The total SDRAM latency from address-valid to first-data-valid is 10 SysClocks, of which 6 are due to SDRAM latency, 1 to error-correction, 1 to driving the SDRAM bus, 1 to driving the CPU bus, and 1 to internal controller latency. See Section 14.0 for timing diagrams.

Figure 12: CPU-To-SDRAM Read Timing



5074-049.eps

6.5.7 Memory Refresh

The controller supports SDRAM refresh using CAS-before-RAS refresh on all of the SDRAM types. The rate of the refresh clock is determined by a programmable 16-bit counter, the SDRAM Refresh Counter Register (T0CNTR), Section 5.6.2. The refresh logic requests access to the SDRAM each time the counter expires. This counter resets to a conservative default value and may be changed in the SDRAM Refresh Control Register (T0CTRL, Section 5.6.1) after reset. Refresh is interspersed with all other memory accesses. The refresh logic can accumulate a maximum of two refresh requests while waiting for access to the memory.

6.5.8 Error Checking

The controller checks even byte-parity or 8-bit ECC on data cycles during memory reads, and it generates even byte-parity or 8-bit ECC on data cycles during memory writes. The CHKMODE and CHKDIS bits of the Memory Control Register (MEMCTRL), Section 6.6.1, enable these functions. Errors or reported in the Memory Check Error Status Register (CHKERR), Section 6.6.3.

All accesses to SDRAM are full dword (64-bit). Any partial-dword write (less than 64-bit) requires a read, internal merge, and write. This works whether you are using ECC or byte-parity. You cannot implement byte-parity and support byte writes with SDRAM, because there is no way to do individual bit writes to the parity RAM. Instead, you must do read-modify-writes, which the controller does in its error checking.

Memory dwords that have correctable ECC errors are not reported as bad parity on the CPU bus. Only memory dwords that have uncorrectable (multi-bit) ECC errors are reported as bad parity on the CPU bus. For details on how parity errors are reported on the CPU bus, see Section 5.2.4.

6.5.9 Memory Sharing in Multi-Controller Configurations

In multi-controller configurations (Section 5.3), SDRAM memory attached to one controller can be accessed by a bus master associated with another controller (CPU, PCI-Bus master, Local-Bus master, or DMA) if both controllers connect to a PCI Bus, so that the access can be made via the PCI Bus. Alternatively, the CPU can be used to copy or move data from one controller’s memory to another controller’s memory.

6.6 Memory-Interface Registers

Table 19: Main Memory Control Registers

Register	Symbol	Offset	R/W	RESET VALUE	Description
Memory Control	MEMCTRL	0x00C0	R/W	0x0000 0000 0000 0080	Miscellaneous main memory control.
Memory Access Timing	ACSTIME	0x00C8	R/W	0x0000 0000 0000 001F	Main memory access timing.
Memory Check Error Status	CHKERR	0x00D0	R	0x0000 0000 0000 0000	Main memory check error status.

6.6.1 Memory Control Register (MEMCTRL)

Bit 1:0 SDRAMTYP *SDRAM Type.*

Value	SDRAM Type
0x0	16Mb SDRAM, 2-bank
0x1	64Mb SDRAM, 2-bank
0x2	64Mb SDRAM, 4-bank
0x3	256Mb SDRAM, 4-bank

The SDRAM memory controller supports two inter-

		leaved memory-space banks. The SDRAM chips in both banks must all be the same type. The term bank in the SDRAMTYP table above refers not to such bank-interleaving of the memory space, but rather to banks inside the SDRAM chips themselves. See Section 6.5 for an explanation of this terminology. SDRAMTYP must not be changed after the ENABLE bit (bit 4) is set.
Bit 2	CHKMODE	*Error Checking Mode*. 1 = 8-bit ECC on MD[63:0]. 0 = even parity on each byte of MD[63:0]. Selects the type of error generation and checking on data cycles. ECC generation and checking is single-error correction, double-error detection (SECDED). The MDC[7:0] signals carry the check data.
Bit 3	CHKDIS	*Memory-Check Disable*. 1 = disable. 0 = enable. Setting this bit disables memory checks on reads and writes, forces zeros on outbound MDC[7:0] check bits, and disables generation of memory-check interrupts, which are enabled by the MCEEN bit of the Interrupt Control Register (INTCTRL, Section 5.5.2).
Bit 4	ENABLE	*Memory Controller Enable*. 1 = enable. 0 = disable. The controller automatically configures the Mode Registers in each SDRAM chip when the ENABLE bit is set. The values written to the Mode Register are described in Section 6.5.2. Accesses to SDRAM are held off until initialization is complete. If the memory PDAR is initialized but the ENABLE bit is not set, any access to memory will hang indefinitely.
Bit 5	*reserved*	Hardwired to 0.
Bit 6	ILEAVD	*Bank-Interleaving*. 1 = enable bank-interleaving, based on the low address bit, SysAD[13]. 0 = disable bank-interleaving. With bank-interleaving enabled, any access with SysAD[13]=0 goes to physical SDRAM bank 0 (addressed by the MAbank0[14:0] bus), and any access with SysAD[13]=1 goes to physical SDRAM bank 1 (addressed by the MAbank1[14:0] bus). When bank-interleaving is disabled, any access to the SDRAM0 PDAR goes to physical bank 0, and

		any access to the SDRAM1 PDAR goes to physical bank 1.
Bit 7	HOLDLD	*Memory Output Hold-Latch Disable.* 1 = disable. 0 = enable. On each output address and control bit, after the corresponding flip-flop, there is a latch before the output buffer. This latch-enable function is directly off the SysClock input signal and is not connected to the on-chip system clock driven by the internal PLL. Thus, this latch can guarantee holds to external circuitry.
Bit 63:8	*reserved*	Hardwired to 0.

6.6.2 Memory Access Timing Register (ACSTIME)

This register controls the timing of non-SDRAM devices on the memory bus (called *external devices* on the memory bus), such as Flash or ROM. These devices correspond to the BootCS# and DCS#[8:2] chip-select signals and to the BOOTCS and DCS[8:2] PDARs.

Bits 4:0	ACCT	*Access Cycle Time.* This field specifies the number of SysClocks required to read or write an external device on the main memory bus. MWE#[1] is used during reads as the output-enable signal ($\overline{\text{OE}}$) of the external devices, and MWE#[0] is used as the read/write enable signal ($\overline{\text{WE}}$) of the external devices. The minimum time that $\overline{\text{OE}}$ or $\overline{\text{WE}}$ will be Low is always one SysClock cycle. This register allows that time to be extended. This field resets to 0x1F, corresponding to a 310 nsec timing in a 100 MHz SysClock system.
Bits 7:5	*reserved*	Hardwired to 0.
Bit 8	DISMRDY	*Disable Memory Ready.* 1 = disable MRDY# signal. 0 = enable MRDY# signal. When this bit is 0, the MRDY# input can terminate an external access. However, this access will also be terminated by ACCT count completion (bits 4:0, above). ACCT can be viewed as a bus time-out. Thus, MRDY# can be used to shorten the timing of an external device access on the main memory bus.
Bits 63:9	*reserved*	Hardwired to 0.

6.6.3 Memory Check Error Status Register (CHKERR)

Bit 35:0	CEADDR	*Memory-Check Error Address*. The address at which the most recent memory-check error occurred.
Bit 47:36	*reserved*	Hardwired to 0.

Bit 55:48 CESYN *Memory-Check Error Syndrome*.

If the CHKMODE bit is set to ECC in the Memory Control Register (Section 6.6.1), this byte contains the syndrome from the data on which the check error occurred, as described in Table 20. If the memory CHKMODE is Parity, this byte contains the XORed even-parity check.

Table 20: ECC-Check Syndromes

Sorted By Syndrome	
ECC Syndrome	**Memory Bit In Error**
0x01	check bit 0
0x02	check bit 1
0x04	check bit 2
0x08	check bit 3
0x0B	data bit 17
0x0E	data bit 16
0x10	check bit 4
0x13	data bit 18
0x15	data bit 19
0x16	data bit 20
0x19	data bit 21
0x1A	data bit 22
0x1C	data bit 23
0x20	check bit 5
0x23	data bit 8
0x25	data bit 9
0x26	data bit 10
0x29	data bit 11
0x2A	data bit 12
0x2C	data bit 13
0x31	data bit 14
0x34	data bit 15
0x40	check bit 6
0x4A	data bit 33
0x4F	data bit 32
0x52	data bit 34
0x54	data bit 35
0x57	data bit 36
0x58	data bit 37
0x5B	data bit 38
0x5D	data bit 39
0x62	data bit 56
0x64	data bit 57
0x67	data bit 58
0x68	data bit 59
0x6B	data bit 60
0x6D	data bit 61

Sorted By Bit Number	
ECC Syndrome	**Memory Bit In Error**
0x01	check bit 0
0x02	check bit 1
0x04	check bit 2
0x08	check bit 3
0x10	check bit 4
0x20	check bit 5
0x40	check bit 6
0x80	check bit 7
0xCE	data bit 0
0xCB	data bit 1
0xD3	data bit 2
0xD5	data bit 3
0xD6	data bit 4
0xD9	data bit 5
0xDA	data bit 6
0xDC	data bit 7
0x23	data bit 8
0x25	data bit 9
0x26	data bit 10
0x29	data bit 11
0x2A	data bit 12
0x2C	data bit 13
0x31	data bit 14
0x34	data bit 15
0x0E	data bit 16
0x0B	data bit 17
0x13	data bit 18
0x15	data bit 19
0x16	data bit 20
0x19	data bit 21
0x1A	data bit 22
0x1C	data bit 23
0xE3	data bit 24
0xE5	data bit 25
0xE6	data bit 26
0xE9	data bit 27
0xEA	data bit 28

Table 20: ECC-Check Syndromes (continued)

Sorted By Syndrome	
ECC Syndrome	**Memory Bit In Error**
0x70	data bit 62
0x75	data bit 63
0x80	check bit 7
0x8A	data bit 49
0x8F	data bit 48
0x92	data bit 50
0x94	data bit 51
0x97	data bit 52
0x98	data bit 53
0x9B	data bit 54
0x9D	data bit 55
0xA2	data bit 40
0xA4	data bit 41
0xA7	data bit 42
0xA8	data bit 43
0xAB	data bit 44
0xAD	data bit 45
0xB0	data bit 46
0xB5	data bit 47
0xCB	data bit 1
0xCE	data bit 0
0xD3	data bit 2
0xD5	data bit 3
0xD6	data bit 4
0xD9	data bit 5
0xDA	data bit 6
0xDC	data bit 7
0xE3	data bit 24
0xE5	data bit 25
0xE6	data bit 26
0xE9	data bit 27
0xEA	data bit 28
0xEC	data bit 29
0xF1	data bit 30
0xF4	data bit 31

Sorted By Bit Number	
ECC Syndrome	**Memory Bit In Error**
0xEC	data bit 29
0xF1	data bit 30
0xF4	data bit 31
0x4F	data bit 32
0x4A	data bit 33
0x52	data bit 34
0x54	data bit 35
0x57	data bit 36
0x58	data bit 37
0x5B	data bit 38
0x5D	data bit 39
0xA2	data bit 40
0xA4	data bit 41
0xA7	data bit 42
0xA8	data bit 43
0xAB	data bit 44
0xAD	data bit 45
0xB0	data bit 46
0xB5	data bit 47
0x8F	data bit 48
0x8A	data bit 49
0x92	data bit 50
0x94	data bit 51
0x97	data bit 52
0x98	data bit 53
0x9B	data bit 54
0x9D	data bit 55
0x62	data bit 56
0x64	data bit 57
0x67	data bit 58
0x68	data bit 59
0x6B	data bit 60
0x6D	data bit 61
0x70	data bit 62
0x75	data bit 63

Bit 56 PCHKERR *Parity-Check Error Occurred.*
1 =even-parity error occurred.
0 = error cleared.
This bit is set by the controller when an error occurs. The bit is cleared by setting bit 2 of the Interrupt Source Clear Register (Section 5.5.5).

Bit 57 ECHKERR *ECC-Check Error Occurred.*
1 = error occurred.

		0 = error cleared. This bit is set by the controller when such an error occurs. The bit can be cleared by setting Bit 2 of the ISCLR field in the Interrupt Source Clear Register (Section 5.5.5).
Bit 58	MCHKERR	*Multi-Bit ECC Check Error* 1 = multi-bit ECC error occurred. 0 = no such error. This bit is set by the controller when such an error occurs. The bit can be cleared by setting Bit 2 of the ISCLR field in the Interrupt Source Clear Register (Section 5.5.5).
Bit 63:59	*reserved*	Hardwired to 0.

7.0 PCI-Bus Interface and Registers

The controller's PCI-Bus interface complies fully, both functionally and electrically, with the *PCI Local Bus Specification, Revision 2.1*. No external logic or buffering is necessary. The interface implements 3.3 V PCI-compliant pads (5 V tolerant) using the NEC CB-C9 process technology.

The interface operates at any of the following configurations:

- ❑ 66 MHz, 64-bit bus (maximum sustained bandwidth 533 MB/sec)
- ❑ 66 MHz, 32-bit bus (maximum sustained bandwidth 267 MB/sec)
- ❑ 33 MHz, 64-bit bus (maximum sustained bandwidth 267 MB/sec)
- ❑ 33 MHz, 32-bit bus (maximum sustained bandwidth 133 MB/sec)

The controller can be a PCI-Bus master on behalf of the CPU, DMA, or Local-Bus masters. The controller can also be a PCI target, providing external PCI-Bus masters with access to controller resources (memory, Local-Bus devices, and internal controller registers such as those that configure DMA, UART, and timers). The controller supports external PCI-Bus masters with access to the PCI-Bus memory space, but not to the PCI I/O space.

The controller supports burst transfers when it is both a PCI-Bus master and target. No wait states are required. Burst lengths of up to 2MB are supported for both reads and writes. The controller also supports 64-bit addressing (Dual Address Cycles), irrespective of whether a 64-bit or 32-bit PCI data bus is implemented, and it supports locked cycles (Exclusive Access) as PCI target and master.

The controller can provide PCI Central Resource services for up to five other PCI devices. When the controller is not providing PCI Central Resource functions it is operating in PCI Stand-Alone Mode. Access to the controller's PCI Configuration Space is supported when the controller is operating either as the PCI Central Resource or in Stand-Alone Mode.

For details of PCI interrupt wiring, see Section 2.2.6 of the *PCI Local Bus Specification*.

Throughout this document, *dword* or *doubleword* means 8 bytes and *qword* or *quadword* means 16 bytes. These definitions are MIPS-compatible and differ from those in the *PCI Local Bus Specification*, where a dword is 4 bytes and a qword is 8 bytes.

7.1 PCI-Bus Configuration and Monitoring

Software configures and monitors the PCI-Bus interface using the following registers:

- ❑ Physical Device Address Registers (PDARs), Section 5.4 on page 45.
- ❑ Interrupt Control Register (INTCTRL), Section 5.5.2 on page 52.
- ❑ Interrupt Status Register 0 (INTSTAT0), Section 5.5.3 on page 55.
- ❑ Interrupt Status 1/CPU Interrupt Enable Register (INTSTAT1), Section 5.5.4 on page 55.
- ❑ Interrupt Clear Register (INTCLR), Section 5.5.5 on page 56.
- ❑ PCI Interrupt Control Register (INTPPES), Section 5.5.6 on page 57.
- ❑ PCI-Bus Registers, Section 7.11 on page 91.
- ❑ PCI Configuration Space Registers, Section 7.13 on page 105.

7.2

Read and Write Buffers

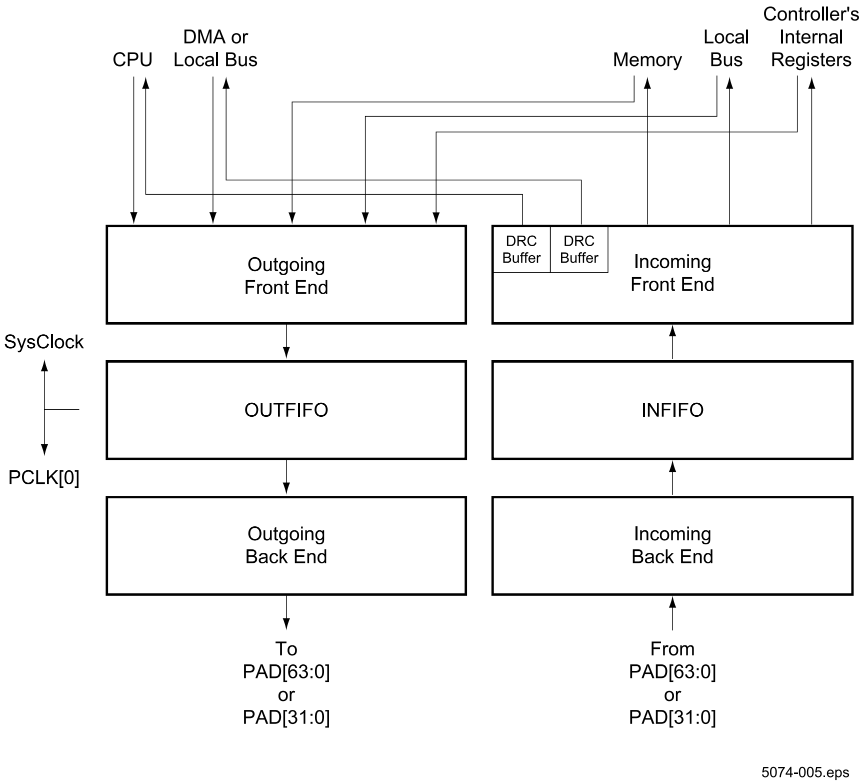
The PCI-Bus data paths are shown in Figure 13. These paths include the following buffers for addresses and data flowing into or out of the controller:

- ❑ 32-entry x 8-byte (256-byte) *PCI Output FIFO* (OUTFIFO) for addresses and data flowing from controller masters and resources to the PCI Bus.
- ❑ 32-entry x 8-byte (256-byte) *PCI Input FIFO* (INFIFO) for addresses and data flowing from the PCI Bus to controller masters and resources.
- ❑ 4-entry x 8-byte (32-byte) *CPU Delayed Read Completion (DRC) Buffer* for data flowing from the PCI Bus to the CPU.
- ❑ 4-entry x 8-byte (32-byte) *DMA Delayed Read Completion (DRC) Buffer* for data flowing from the PCI Bus to the DMA logic.

Both the OUTFIFO and INFIFO hold address and data in a multiplexed fashion. Each of the 32 entries holds 8 bytes for address or data, plus an additional 13 bits for transaction type, byte-enables, command and status information. Thus, each FIFO can hold several small write bursts or one large write burst.

Interrupts on the PCI Bus are not stalled until the FIFOs empty. The controller can relay PCI interrupts to the CPU immediately.

Figure 13: PCI-Interface Data Paths



7.3

PCI Commands Supported

Table 21 summarizes the PCI commands supported by the controller as a master and target on the PCI Bus.

Table 21: PCI Commands Supported

C/BE#[3:0]	Command	As PCI Master (Controller-to-PCI)	As PCI Target (PCI-to-Controller)
0000	Interrupt Acknowledge	Yes [a]	*ignored*
0001	Special Cycle	Yes [a]	*ignored*
0010	I/O Read	Yes	*ignored*
0011	I/O Write	Yes	*ignored*
010x	*reserved*	— [a]	*ignored*
0110	Memory Read	Yes	Yes, as Delayed Read
0111	Memory Write	Yes	Yes, as Posted Write
100x	*reserved*	— [a]	*ignored*
1010	Configuration Read	Yes	Yes, as Delayed Read
1011	Configuration Write	Yes	Yes, as Delayed Write
1100	Memory Read Multiple	Yes	Yes, as Delayed Read

Table 21: PCI Commands Supported (continued)

C/BE#[3:0]	Command	As PCI Master (Controller-to-PCI)	As PCI Target (PCI-to-Controller)
1101	Dual Address Cycle	Yes	Yes
1110	Memory Read Line	Yes	Yes, as Delayed Read
1111	Memory Write and Invalidate	Yes [a]	Yes, as Posted Write

[a] In normal operation, the controller does not generate these commands. However, for test purposes the TYPE field in the PCI Master (Initiator) Register (PCIINITn, Section 7.11.3) can be written so as to generate any PCI command.

7.4 PCI Master Transactions (Controller-to-PCI)

The controller supports bidirectional data transfers between the CPU, DMA, or Local-Bus masters and PCI-Bus memory and I/O targets by becoming a PCI-Bus master. The initiator of such a transaction (the CPU, DMA, or a Local-Bus master) accesses the PCI-Bus resource through a local physical address that corresponds to one of two PCI Address Windows.

The controller can generate all PCI command types (Table 21) as a PCI-Bus master. For CPU instruction-cache fills (4-dword block), the controller reads 4 dwords from the PCI target, beginning with the first dword in the cache line (address = 0), and returns them to the CPU in the correct sub-block order. As Figure 13 shows, read data is assembled in the controller’s 4-entry x 8-byte CPU Delayed Read Completion (DRC) Buffer before being sent to the CPU.

7.4.1 PCI Address Window Registers

PCI memory and I/O targets are accessed through the controller’s two PCI Address Windows. These windows are configured by the PCI Address Window Registers (PCIW0 and PCIW1, Section 5.4). The registers have two corresponding PCI Master (Initiator) Registers (PCIINIT0 and PCIINIT1, Section 7.11.3) which specify PCI address and other information regarding transactions initiated through the PCI Address Windows.

Section 7.4.2 gives an example of how the controller decodes PCI-Bus addresses from the values on the SysAD[31:0] bus, the ADDR and MASK fields of the PCIW0 and PCIW1 PDARs, and the PCIADD fields of the PCIINITn registers.

7.4.2 PCI Address Decoding Example

When the CPU generates a 36-bit physical address to a PCI device, the ADDR and MASK fields of the PDARs determine where that access goes. For example, suppose that the CPU address range 0x0_C000_0000 through 0x0_DFFF_FFFF should go to the PCI Bus through the PCI Address Window 0 (the PCIW0 PDAR). This is equivalent to saying that when the SysAD[35:29] address bits are b0000_110, the access should go to the PCI Bus.

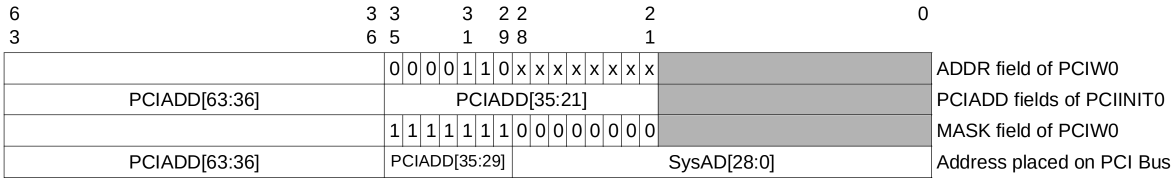
In this example, the MASK and ADDR fields of the PDAR for PCI Address Window 0 (PCIW0) would be programmed as follows:

- ❑ MASK = 0x7, which means only compare [35:29]
- ❑ ADDR[35:21] = b0000_110x_xxxx_xxx

The value in the ADDR field specifies the high-order address bits that must match the incoming address, and the MASK field specifies which ADDR bits are to be used in the address comparison. In this example, a MASK value of 0x7 means only compare [35:29], and this is equivalent to a mask of b1111_1110_0000_000.

When the CPU generates a PCI-Bus access, the controller uses only address bits SysAD[28:0] from the CPU. These are the bits that were not masked by the PCIW0 PDAR. A PCI device can have up to 64 address bits. The two PCIADD fields in the PCIINIT0 register (Section 7.11.3), PCIADD[63:36] and PCIADD[35:21], specify the remainder of the PCI address. Thus, in this example, the PCIADD[63:29] bits should be programmed with the upper PCI address bits, and the PCIADD[28:21] bits will be ignored.

Figure 14: PCI Address Decoding Example (64-Bit PCI Address)



63	36 35	31	29 28	21	0	
	0 0 0 0 1 1 0		x x x x x x x x			ADDR field of PCIW0
PCIADD[63:36]	PCIADD[35:21]					PCIADD fields of PCIINIT0
	1 1 1 1 1 1 1		0 0 0 0 0 0 0 0			MASK field of PCIW0
PCIADD[63:36]	PCIADD[35:29]		SysAD[28:0]			Address placed on PCI Bus

7.4.3
PCI-Master Writes

The controller supports combining and byte-merging on writes to the PCI Bus. These functions are enabled in the PCI Master (Initiator) Control Registers (PCIINITn, Section 7.11.3). The sections below describe the controller's handling of these functions. See Section 3.2.6 of the *PCI Local Bus Specification* for more details.

7.4.3.1
Combining

When the COMBINING bit is set in the PCIINITn register, the controller combines writes to sequential 64-bit dwords into a single PCI-Bus burst write. The TYPE field in the PCIINITn register should contain the value b011 (Memory Write).

Accesses do not need to be full dword writes in order to be combined. A byte write to address 0x0 (dword 0) followed by a byte write to address 0xF (dword 1) will be combined into a PCI burst. Each dword in the burst will have only a single byte-enable asserted. If a 32-bit PCI Bus is implemented, the burst consists of four 32-bit words. The first word has one byte-enable asserted, the second and third have none asserted, and the fourth word has one asserted.

7.4.3.2
Byte-Merging

When the MERGING bit is set in the PCIINITn register, the controller byte-merges a sequence of individual writes to the same 64-bit dword into a single PCI-Bus write. The TYPE field in the PCIINITn register should contain the value b011 (Memory Write). Byte-merging can be done in any order. A byte write to address 0x3, followed by a byte write to address 0x2, followed by a byte write to 0x0 could all be merged into a single write to dword 0x0.

If any byte in the upper half of a dword is written, subsequent writes to the lower half are not merged. Instead, the dword containing the modified upper half is written to the PCI Bus, and the subsequent write to the lower half of the dword is placed in the controller's merge buffer so that merging can continue. This prevents the potential reordering of writes on a 32-bit PCI Bus.

If merging is enabled and a dword has only been partially written (i.e., one of the upper four bytes has not been written to), the dword stays in the merge buffer indefinitely, waiting for an additional write that might be merged into this dword. The write does not actually occur on the PCI Bus until a subsequent PCI Master cycle is requested.

7.4.4 PCI-Master Reads

When a read request comes through the controller, as PCI-Bus master, it is issued onto the PCI Bus to the PCI target. The master from which the read originated (the CPU, DMA, or DMA on behalf of a Local-Bus master) is kept waiting until the read data is returned from the PCI target; several clocks elapse for the first data item, and the prefetching configuration (Section 7.4.4.2) determines latency for subsequent data items. The read data returns through the controller's INFIFO and one of the two 4-entry x 8-byte Delayed Read Completion (DRC) Buffers shown in Figure 13, even though the read may be completed on the PCI Bus as a normal (not delayed) read. Thus, if the CPU issues a PCI read, the CPU will be stalled until the read data returns. But the DMA will not be affected by the CPU's waiting.

7.4.4.1 Retried Reads

If a read request coming through the controller, as PCI-Bus master, is retried by the target, the read request is resubmitted at the front of the INFIFO. This allows other requests currently in the INFIFO to be processed first. If a read request has transferred some but not all of the requested data and gets a target disconnect, the request is immediately retried.

7.4.4.2 Prefetching on PCI-Master Reads

By default, the controller fetches the exact amount of data requested by the CPU, DMA, or Local-Bus master. If the TYPE field in the PCI Master (Initiator) Control Registers (PCIINITn, Section 7.11.3) is set to b011 (Memory Read) and the PREFETCHABLE bit is set to 1, additional data is prefetched from the PCI Bus during reads. The Memory Read command is forced to Memory Read Line or Memory Read Multiple, depending on the amount of data to be prefetched, as follows:

- ❑ For CPU single-dword reads, the controller prefetches the rest of the current 4-dword block, plus the number of additional 4-dword blocks specified by the SINGLE_PFB field of the PCIINITn register. If SINGLE_PFB is all 1s (0x1F), the controller prefetches forever.
- ❑ For CPU 4-dword block reads (CPU cache fills), the controller fetches the requested block, and it prefetches the number of additional blocks specified by the BLOCK_PFB field of the PCIINITn register. If BLOCK_PFB is all 1s (0x3F), the controller prefetches continuously.
- ❑ For DMA reads, the DMA logic indicates exactly how many dwords are needed to complete the current DMA. If the transfer is greater than 64 blocks, the controller prefetches forever.
- ❑ No prefetching is done for reads by Local-Bus masters.

A subsequent read by the same master is a prefetch hit if the PREFETCHABLE bit of the PCIINITn register is still set, and the address is consecutive with the end of the previous read. The subsequent read can be a dword, partial dword, or block, but its address must start on the dword following the last dword of the previous read.

Prefetching stops when:

- ❑ The prefetch block count has been fetched (unless prefetching forever).
- ❑ The INFIFO is full.
- ❑ The target disconnects or aborts the transaction.
- ❑ The address crosses a 2 MB boundary.
- ❑ The master does any write.
- ❑ The master does a read that is not a prefetch hit.
- ❑ The master does a read to the other PCI Address Window.
- ❑ Another master (CPU or DMA) does any PCI access.
- ❑ A PCI target read is serviced (i.e., read data is placed in the OUTFIFO).

Prefetched data is discarded when:

- ❑ The master does any write.
- ❑ The master does a read that is not a prefetch hit.
- ❑ The master does a read to the other PCI Address Window.
- ❑ Another master (CPU or DMA) does any PCI access.
- ❑ A PCI target read is serviced (i.e., read data is placed in the OUTFIFO).
- ❑ The master is idle and the INFIFO is full.

If the number of dwords to be fetched crosses a PCI cache-line boundary, as specified by the PCI Cache-Line Size Register (CLSIZ, Section 7.13.7), the Memory Read command is forced to a Memory Read Multiple command; otherwise the command is forced to a Memory Read Line command.

7.4.5 **PCI-Master Parity Detection**

If the controller, as PCI master, detects bad even parity on a read or write *data* cycle, the controller:

- ❑ Completes the access.
- ❑ Reports the parity error in the DPE bit of the PCI Status Register (PCISTS, Section 7.13.4).
- ❑ Generates a CPU interrupt, if enabled by the:
  - PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3),
  - PERIN bit in the PCI Control Register (PCICTRL, Section 7.11.1), and
  - PCIEEN bit in the Interrupt Control Register (INTCTRL, Section 5.5.2).
- ❑ On a read, asserts PERR#, if enabled by the:
  - PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3).
- ❑ On a CPU read, forces load parity to be returned to the CPU, if enabled by the:
  - PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3).
- ❑ On a DMA read, causes the DMA transfer to stop and sets the PRDERR bit in the DMA Control Registers (DMACTRLn, Section 9.5.1), if enabled by the:
  - PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3).
- ❑ Asserts SERR#, if enabled by the:
  - SERREN bit in the PCI Command Register (PCICMD, Section 7.13.3),

- PERSE bit in the PCI Control Register (PCICTRL, Section 7.11.1), and
- PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3).

7.4.6 PCI I/O Space Cycles

When the TYPE field of the PCI Master (Initiator) Control Registers (PCIINITn, Section 7.11.3) contains the value b001, reads and writes on the PCI Bus by the controller, as PCI master, are to the PCI I/O Space. The PCI specification allows only 32-bit I/O accesses. Thus, the ACCESS_32 bit in the PCIINITn register should be set.

Byte-merging can be used (i.e., the MERGING bit can be set in the PCIINITn register), but not combining (the COMBINING bit must be cleared in the PCIINITn register). Only individual 32-bit I/O accesses are supported. Do not attempt bursts to the PCI I/O Space. Combining must not be used. Do not attempt accesses greater than 32-bits (i.e. full dword).

The controller drives the low two bits of the PCI address correctly, according to which byte-enables are asserted. See Section 3.2.2 of the *PCI Local Bus Specification* for details.

7.5 PCI Target Transactions (PCI-to-Controller)

As a target on the PCI Bus, the controller responds to PCI Memory and Configuration (but not PCI I/O) transactions. All of the controller's resources (memory, Local-Bus devices, and internal controller registers such as those that configure DMA, UART, and timers) are accessible to PCI Bus masters. The controller accepts full-speed (no wait-state) burst reads and writes but implements them as delayed reads and delayed or posted writes, as shown in Table 21 on page 80.

The controller has eleven PCI target address ranges within which it responds as a target to PCI-Bus masters. These ranges are programmable through the PDARs (all except the two PCI Address Window PDARs, as described immediately below in Section 7.5.1). A target address range is only visible on the PCI Bus when the VISPCI bit is set in its PDAR.

Each PDAR, except the two PCI Address Window PDARs, has a corresponding Base Address Register (BAR) in PCI configuration space. Each BAR is 64-bits wide, with varying bits appearing hardwired to zero, as specified in the MASK field of the PDAR. The PDAR should be programmed before allowing a PCI master to access the BARs. When the controller decodes a PCI address for one of its BARs, it arbitrates with its internal CPU, DMA, and Local-Bus logic for access to the requested resource. Bursts are disconnected with a Target Disconnect if the address crosses a 2MB boundary, which is the smallest granularity a controller BAR can have.

7.5.1 PCI Loop-Back Accesses

When the controller is a PCI-Bus master, it is possible for the controller to also be a PCI-Bus target. That is, the CPU, DMA, or a Local-Bus master can access another *controller* resource via a loop-back on the PCI Bus, by using a PCI Address Window rather than addressing the resource directly. However, the analogous type of loop-back access is not possible when the controller is responding as a PCI-Bus target to a request by a PCI-Bus master. That is, a PCI-Bus master cannot access a PCI-Bus target via a loop-back through the controller.

Because the two PCI Address Window PDARs cannot be accessed by PCI-Bus masters, they have no corresponding Base Address Registers (BARs) in the controller's PCI configuration space.

7.5.2 PCI-Target Writes

The controller implements PCI target writes as delayed or posted writes. Addresses and data for posted writes are held in the INFIFO, shown in Figure 13. Any number of writes can be posted and up to four delayed transactions can be pending simultaneously. Any additional delayed transactions are unconditionally retried until one of the four currently pending transactions completes.

Configuration writes are delayed writes. The address and data for a configuration write is placed in the INFIFO, and the PCI transaction is terminated with Retry. When the delayed write has been performed into the Configuration Register, the transaction is allowed to complete.

7.5.3 PCI-Target Reads

The controller implements PCI target reads as delayed reads. Delayed reads are split into two parts: a delayed-read request part, issued by the PCI-Bus master, and a delayed-read completion part, which is the data returned by the targeted controller resource. Delayed reads have the advantage of freeing the PCI Bus for other transactions while the target of the delayed read takes its time returning the read data.

The delayed-read request (address and command) is placed in the INFIFO, and the PCI transaction is terminated with Retry. The appropriate controller resource is read from, and the delayed-read completion data is placed in the OUTFIFO. When the transaction is retried, the target data is driven onto the PCI Bus.

When prefetching data for PCI target reads, the controller considers the BAR PREFETCHABLE bit (Section 7.13.10), the PCI command, the incrementing type specified by PCI_AD[1:0], and the value in the PCI Cache Line Size Register (Section 7.13.7), as shown in Table 22. If the PCI Prefetch Count exceeds 31 dwords, it is ignored and the controller prefetches forever.

Table 22: Prefetching Variables For PCI Target Reads

BAR PREFETCHABLE Bit	PCI Command	Incrementing Type Specified by PCI_AD[1:0]	Prefetch Count (amount of data prefetched from controller resource)
0	Memory Read	Any	None
1	Memory Read	Linear	remainder of PCI Cache Line
0	Memory Read Line	Linear	remainder of PCI Cache Line
1	Memory Read Line	Linear	remainder of PCI Cache Line plus 1 more PCI Cache Line
0	Memory Read Multiple	Linear	remainder of PCI Cache Line plus 1 more PCI Cache Line
1	Memory Read Multiple	Linear	remainder of PCI Cache Line plus 2 more PCI Cache Lines
—	Configuration Read	—	None
—	Any	Anything other than Linear	None

Prefetching stops when:

- ❑ The OUTFIFO is full.
- ❑ The master terminates the transaction.
- ❑ The address crosses a 2Mbyte boundary.
- ❑ The Prefetch Count has been fetched (unless prefetching forever).

Unused prefetched data is discarded when:

- ❑ The master terminates the transaction.
- ❑ The Discard Timer for this delayed transaction expires.

7.5.4 PCI-Target Parity Detection

If the controller, as PCI target, detects bad even parity on an *address* cycle, the controller:

- ❑ Reports the parity error in the DPE bit of the PCI Status Register (PCISTS, Section 7.13.4).
- ❑ Generates a CPU interrupt, if enabled by the:
  - PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3),
  - AERIN bit in the PCI Control Register (PCICTRL, Section 7.11.1), and
  - PCIEEN bit in the Interrupt Control Register (INTCTRL, Section 5.5.2).
- ❑ Asserts SERR#, if enabled by the:
  - SERREN bit in the PCI Command Register (PCICMD, Section 7.13.3), and
  - AERSE bit in the PCI Control Register (PCICTRL, Section 7.11.1), and
  - PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3).
- ❑ Ignores the access.

If the controller, as PCI target, detects bad even parity on a PCI target write *data* cycle, the controller:

- ❑ Completes the write.
- ❑ Asserts PERR#, if enabled by the:
  - PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3).
- ❑ On a write to SDRAM, forces bad parity or ECC to be written to that address, if enabled by the:
  - PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3).
- ❑ On a write to the controller’s internal registers, including timers, DMA, and UART, the data is ignored (not written), if enabled by the:
  - PEREN bit in the PCI Command Register (PCICMD, Section 7.13.3).

7.6 64-Bit PCI Bus

The controller optionally supports a 64-bit PCI Bus. In this case, the controller’s 32-bit Local Bus cannot be used, because the LOC_AD[31:0] signals and other Local-Bus signals are reallocated for use on the PCI Bus, as described in Section 8.5.

Two functions are involved in this 64-bit PCI Bus support: the controller’s PCI-Bus width, and the PCI 64-bit Bus Extension, as defined in the PCI Specification. The two functions are enabled as follows:

- *Controller's 64-Bit PCI-Bus Width:* Enabled when PCI64# is asserted during the assertion of PCIRST#.
- *PCI 64-Bit Bus Extension (per PCI Specification):* Enabled when REQ64# is asserted by the PCI Central Resource during the assertion of PCIRST#, and dynamically negotiated on a per-transaction basis using REQ64# and ACK64#.

The possible configurations using these two signals are:

- If PCI64# is negated during PCIRST#, the controller implements a 32-bit PCI Bus, irrespective of the state of REQ64#.
- If (a) PCI64# is asserted during the assertion of PCIRST#, (b) the controller is not the PCI Central Resource and the REQ64# input from the PCI Central Resource is negated during the assertion of PCIRST#, the controller's 64-bit PCI behavior is disabled and the high 32 bits of the controller's 64-bit PCI Bus are always driven to prevent them from floating.
- If, during the assertion of PCIRST#, PCI64# is asserted and REQ64# is asserted (i.e., the PCI Central Resource asserts REQ64# to the controller or the controller itself is the PCI Central Resource), the controller attempts to initiate 64-bit data transactions whenever possible to improve performance. The controller also responds properly as a 64-bit target.

When the controller is configured for 64-Bit PCI-Bus width, the controller only initiates 32-bit transactions when the ACCESS_32 bit is set in the PCI Master (Initiator) Registers 0 and 1 (PCIINITn), Section 7.11.3. This bit must be set when the controller, as PCI-but master, accesses the PCI I/O Space or Configuration Space. The ACCESS_32 bit resets to 0.

7.7 Dual Address Cycle (DAC) Support

The controller supports Dual Address Cycles (DAC) as both a PCI master and target, thus supporting 64-bit addressing. As a PCI master, the controller automatically uses 64-bit addressing when the high PCIADD field (bits 63:32 in the corresponding PCI-INITn register) are non-zero.

7.8 PCI Central Resource Support

Every PCI Bus must have a PCI Central Resource that provides special functions for that bus. In systems which have multiple PCI Buses, each PCI Bus must have its own Central Resource. The controller performs PCI Central Resource functions when the PCICR# input is asserted to the controller at reset.

7.8.1 Central Resource Functions

The controller can optionally provide some or all of these PCI Central Resource functions. These functions are enabled when PCICR# is asserted at reset and software configures various fields in the PCI Control Register (PCICTRL), Section 7.11.1, and PCI Arbiter Register (PCIARB), Section 7.11.2.

When the PCICR# input is asserted to the controller at reset:

- *PCI Clocks:* The controller's PCLK[4:0] signals are all outputs that can be connected to the CLK input on up to five other PCI devices. The controller itself always uses PCLK[0] as its PCI-Bus clock. See Section 7.9 for details.
- *PCI-Bus Arbitration:* The controller's REQ#[4:0] signals are all inputs that can be connected to the REQ# output from up to five other PCI devices, and its GNT#[4:0] signals are all outputs that can be connected to the GNT# inputs from

these other PCI devices.

- ❑ *CPU Interrupts:* The controller’s INTA# signal is bidirectional, rather than an output, so that the controller can accept up to five PCI interrupts on INTA# through INTE#. It forwards these interrupts to the CPU, as specified in Section 5.5.2.
- ❑ *PCI Reset:* The controller’s PCIRST# signal is an output rather than input, and this signal can be connected to the RST# input on up to five other PCI devices. The controller asserts PCIRST# in any of the following cases:
    - On Power-Up.
    - When the CLDRST bit is set in the CPU Status Register (CPUSTAT), Section 5.5.1.
    - When the PCICRST or PCIWRST bit is set in the PCI Control Register (PCICTRL), Section 7.11.1.
- ❑ *64-Bit Bus Extension:* The controller configures 64-bit PCI-Bus operation by asserting its REQ64# output at the end of reset. See Section 7.6 for details.
- ❑ *PCI Configuration Cycles:* The controller is responsible for generating IDSEL inputs to other PCI devices on the same PCI Bus, during CPU accesses the PCI Configuration Space (Section 7.12). Section 7.12.3 illustrates how these IDSEL inputs may be generated.

7.8.2 Central Resource Terminology

When the controller is not providing PCI Central Resource functions, it is operating in PCI *Stand-Alone Mode*. For example, in a system where there is already a CPU and controller providing host and Central Resource functions, additional memory or Local-Bus capability can be provided by a second controller in PCI Stand-Alone Mode, as shown in Figure 5 on page 16 and Figure 6 on page 17. This capability is especially useful if the Main Controller has a 64-bit PCI, thus removing its Local Bus.

It is possible to have multiple controllers on a single CPU and have all controllers on the same PCI Bus. In this case, only one controller should provide the Central Resource functions and the others should operate in Stand-Alone Mode. The controller can support a CPU in Stand-Alone Mode. However, if the CPU is not the Main CPU in the system, care must be taken when configuring controller resources at boot time. For example, two CPUs should not simultaneously attempt to modify the controller configuration and control registers. The PCIWRST bit in the PCI Control Register (PCICTRL), Section 7.11.1, must be used carefully in this case.

The concepts of PCI Central Resource and Main Controller (Section 5.3.2) are unrelated. The controller can be a Main Controller for a given CPU, but that CPU might not be the Main CPU in the system, and the Main Controller for that CPU, or any other CPU in the system, might not provide the PCI Central Resource for the system.

7.8.3 External Arbitration

External arbitration logic can be used. For example, it would be needed if more than five PCI devices need to arbitrate with the controller, or if a custom arbitration protocol is desired. Even with external arbitration logic, the controller can optionally perform all other PCI Central Resource functions.

The controller’s arbiter is disabled by:

- ❑ Negating the PCICR# input, thus disabling *all* Central Resource functions by the controller, or

- Setting the ARBDISABLE bit (bit 63) of the PCI Arbiter Registers (PCIARB, Section 7.11.2), thus disabling only the arbitration function, but leaving the other Central Resource functions enabled.

When the controller's internal arbiter is disabled, REQ#[0] is an output to the external arbiter, GNT#[0] is an input from the external arbiter, and REQ#[4:1] and GNT#[4:1] are unused inputs.

7.9 PCI Clocking

The controller generates the PCI clocks, PCLK[4:0], based on either the external PCLKIN signal or a synchronous multiple of the CPU SysClock. The source for the controller's generation of PCLK[4:0] is controlled by the CLKSEL[2:0] field in the PCI Control Register (PCICTRL, Section 7.11.1), as shown in Table 23. At reset, the CLKSEL[0] bit takes the state of the M66EN (66 MHz Enable) input signal, and the CLKSEL[2:1] bits are set via the Serial Mode EEPROM bits. These bits should only be changed while the PCI Bus is reset; otherwise, PCLK[4:0] may glitch.

Table 23: PCLK[4:0] Source Specification

CLKSEL	PCLK Source	When used
000	PCLK frequency is 1/3 SysClock	For 33 MHz PCI Bus with SysClock > 66 MHz
001	PCLK frequency is 2/3 SysClock	For 66 MHz PCI Bus with SysClock > 66 MHz [a]
010	PCLK frequency is 1/2 SysClock	For 33 MHz PCI Bus with SysClock <= 66 MHz
011	PCLK frequency is equal to SysClock	For 66 MHz PCI Bus with SysClock <= 66 MHz
10x	PCLK is driven by signal PCLKIN	Any combination of SysClock and PCLK
11x	*reserved*	

[a] CLKSEL = 001 uses the controller's internal 2x multiplying PLL. To stay within the specified operating range of this PLL, 112.5 MHz >= SysClock >= 67.5 MHz.

If the PCI clock is generated by external logic via the PCLKIN input, the external clock logic must follow the M66EN signal on the PCI Bus; i.e., it must only generate a clock greater than 33 MHz when M66EN is asserted. The clock skew between PCLKIN and the PCLK[4:0] outputs is several nanoseconds.

The PCISYNC bit in the PCI Control Register (PCICTRL, Section 7.11.1) indicates whether PCLK[4:0] is synchronous to SysClock. This affects the performance of handshake signals between the PCI logic and the rest of the controller. The controller clears this bit at reset. When it is set by software, it indicates that signals passing between the PCLK[4:0] and SysClock timing domains are synchronized and do not need re-synchronization to avoid metastability. However, even if PCLK[4:0] are sourced from SysClock, and thus are a synchronous multiple of SysClock, PCLK[4:0] are not skew-controlled and the edges are not aligned to SysClock, due to clock skew internal to the controller. Thus, it is generally not advisable to set the PCISYNC bit.

The controller always uses the PCLK[0] input as its PCI clock, but the controller can either receive or generate the PCI clocking for the system. When the PCICR# input is negated (i.e. the controller is not providing PCI Central Resource functions), PCLK[0] is enabled as the PCI clock input, and PCLK[4:1] are floated. When PCICR# is asserted, PCLK[0] is enabled as the PCI clock input (for the controller's PCI interface), and PCLK[4:0] are all enabled as outputs. In this Central Resource configuration, PCLK[4:0] are five separate, identical copies of the PCI clock.

All devices on the PCI Bus must operate at the same clock speed. If different PCI clock speeds must be supported, this can be done in a multi-controller configuration (Section 5.3). For example, two controllers can be connected to a single CPU, with one controller running at 33 MHz and the other at 66 MHz, as shown in Figure 7 on page 18.

7.10 PCI Locked Cycles

The controller’s bidirectional LOCK# signal provides a mechanism for obtaining exclusive access to PCI targets, as defined in the *PCI Local Bus Specification*, Section 3.6. As a PCI master, the controller can assert LOCK#. As a PCI target, the controller responds to the assertion of LOCK#.

To implement locking when the controller is the PCI master, software for the initiator (CPU or DMA) sets the LOCK bit in the PCI Master (Initiator) Registers 0 and 1 (PCIINITn), Section 7.11.3, and then performs a PCI-Bus read. This locks the 16-byte read region using the LOCK# protocol. No other PCI device is allowed to access that 16-byte region during the read.

You can use the PCI LOCK# protocol to maintain semaphores. However the *PCI Local Bus Specification* recommends against this, advising instead that a software protocol be used. This is mainly for compatibility reasons, because not all systems may implement LOCK# properly (if at all). Furthermore, using LOCK# may be inefficient. In particular, you cannot have multiple simultaneous locks. Only have one 16-byte region can be locked on the entire bus at any one time.

This locking mechanism affects only PCI-Bus accesses. It does not prevent the CPU from accessing an area of the controller’s memory that is locked by a PCI-Bus master. However, the CPU can prevent PCI-Bus masters from accessing a 16-byte region of the controller’s memory by first setting the LOCK bit in the PCIINITn register and then accessing the controller’s memory with a loopback access.

7.11 PCI-Bus Registers

Table 24: PCI-Bus Registers

Register	Symbol	Offset	R/W	Reset Value	Description
PCI Control	PCICTRL	0x00E0	R/W	0x?000 0000 8000 000? [a]	Miscellaneous PCI control.
PCI Arbiter	PCIARB	0x00E8	R/W	0x0050 0011 1100 003F	PCI arbiter control.
PCI Master (Initiator) 0	PCIINIT0	0x00F0	R/W	0x0000 0000 0000 8406	Control for PCI Address Window 0.
PCI Master (Initiator) 1	PCIINIT1	0x00F8	R/W	0x0000 0000 0000 8406	Control for PCI Address Window 1.
PCI Error	PCIERR	0x00B8 [b]	R/W	0x0000 0000 0000 0000	Address of PCI internal error.

[a] The question marks (?) indicate that the reset value depends on the CLKSEL, PCIWRST and PLL_STBY fields of PCICTRL, which in turn depend on external inputs during reset.

[b] Note the non-consecutive address.

7.11.1 PCI Control Register (PCICTRL)

Bit 0 PCISYNC *PCI-Synchronized.*
1 = synchronized.
0 = not synchronized.
When set, this bit indicates that signals passing between the PCLK[4:0] and SysClock timing

domains are synchronized and do not need re-synchronization to avoid metastability. Resets to 0. *This bit is provided for testing purposes only. Do not set it to 1 for normal operation! Setting this bit will not provide a significant performance increase and may cause undesirable behavior.*

Bit 3:1 CLKSEL[2:0] *PCLK[4:0] Output Source Selections*.

CLKSEL Value	**Description**
000	PCLK[4:0] frequency is 1/3 SysClock
001	PCLK[4:0] frequency is 2/3 SysClock (uses 2x PLL)
010	PCLK[4:0] frequency is 1/2 SysClock
011	PCLK[4:0] frequency is equal to SysClock
10x	PCLK[4:0] is driven by PCLKIN signal
11x	*reserved*

Even when PCLK[4:0] are a synchronous multiple of SysClock, they are not skew-controlled, and the edges are not aligned with SysClock. This field has no effect when PCICR# is negated because in that case the PCLK[4:0] outputs float. At reset, the CLKSEL[0] bit takes the state of the M66EN input signal, and the CLKSEL[2:1] bits are set via the Serial Mode EEPROM bits 260:259.

Bit 7:4 CPUHOG *Minimum Number of Accesses by CPU.*
The minimum number of consecutive CPU accesses to PCI resources through the PCI Output FIFO (OUTFIFO) before the CPU is forced to allow another controller resource to take control of the PCI Bus. 1 to 15 means 1 to 15 consecutive accesses, 0 means 16 consecutive accesses. Resets to 0 (16 accesses). The limit counter starts counting with the first CPU access, irrespective of when another controller resource requests the PCI Bus. The PCI-Bus CPUHOG and DMAHOG fields are the software interface to the Programmable 2-Way Arbiter, shown in Figure 1 on page 12.

Bit 11:8 DMAHOG *Minimum Number of Accesses by DMA.*
The minimum number of consecutive PCI-Bus accesses the DMA may perform through the PCI Output FIFO (OUTFIFO) before the DMA is forced to allow another controller resource to take control of the PCI Bus. 1 to 15 means 1 to 15 consecutive accesses, 0 means 16 consecutive accesses. Resets to 0 (16 accesses). The limit counter starts counting with the first DMA access, irrespective of

		when another controller resource requests the PCI Bus. This behavior of the limit counter differs from that of the CPUHOG, PCIHOG and DMAHOG fields in the Local Bus Configuration Register (LCNFG, Section 8.6.1).
Bit 12	*reserved*	Hardwired to 0.
Bit 13	FAPER	*Force Address-Parity Errors*. 1 = force even-parity errors on addresses when controller is PCI master; i.e., generates odd parity. 0 = normal even-parity generation on addresses (reset value).
Bit 14	FDPER	*Force Data-Parity Errors*. 1 = force even-parity errors on data when controller is PCI master (writes) or target (reads); i.e., generates odd parity. 0 = normal even-parity generation on data (reset value).
Bit 15	FIFOSTALL	*PCI Output FIFO Stall*. (read-only) 1 = PCI Output FIFO (OUTFIFO) is stalled. 0 = PCI Output FIFO (OUTFIFO) not stalled (reset value).
Bit 23:16	RTYLIM	*Retry Limit.* Specifies how many consecutive retries the controller accepts from a single target. 0 means no limit, non-zero values are multiplied by 2^8 (8-bit shifted) to derive the actual retry limit. Resets to 0 (no limit).
Bit 31:24	DISCTIM	*Discard Time-Out.* When controller performs a delayed read as a PCI target, and the master does not repeat the request within the DISCTIM number of PCI clocks, PCLK[4:0], the controller discards the read data to prevent deadlocks. 0 means 2^{16} clocks, non-zero values are multiplied by 2^8 (8-bit shifted) to derive the actual discard time-out. Resets to 0x80 (2^{15} PCI clocks).

The following five bits enable the controller's capture, into the PCI Error Register (Section 7.11.4), of the PCI address at which a PCI error occurred. These are controller internal errors, in which the controller was the master and/or target of the PCI transaction. All bits reset to 0.

Bit 32	TACH	*Target-Abort Address Capture*. 1 = enable capture of target-abort address when

controller is PCI master.
0 = disable this capture.

Bit 33 MACH *Master-Abort Address Capture.*
1 = enable capture of master-abort address when controller is PCI master.
0 = disable this capture.

Bit 34 RTYCH *Retry-Limit Exceeded Address Capture.*
1 = enable capture of retry-limit-exceeded address when controller is PCI master.
0 = disable this capture.

Bit 35 PERCH *Data-Parity Error Address Capture*.
1 = enable capture of data-parity error address (on reads or writes) when controller is PCI master.
0 = disable this capture.
This bit is independent of the Parity Error Response (PEREN) bit in the PCI Configuration Command Register (Section 7.13.3).

Bit 36 DTIMCH *Discard-Timer Expired Address Capture.*
1 = enable capture of discard-timer expired address when controller is PCI target.
0 = disable this capture.
This is only an error when it occurs on reads in which the data is not prefetchable. If the data is prefetchable, then it is silently discarded. Data is prefetchable if the PREFETCHABLE bit in the PCI Base Address Register (BAR) for this device (Section 7.13.10) is set, or the PCI command was a Memory Read Line or Memory Read Multiple.

Bit 39:37 ERRTYPE *Error Type.* (read-only)

ERRTYPE	Meaning	Controller was
000	No error	
001	Target Abort	Master
010	Master Abort	Master
011	Retry Limit Exceeded	Master
100	Data Read Parity Error	Master
101	Data Write Parity Error	Master
110	Discard Timer Expired	Target
111	*reserved*	

Indicates the type of PCI error whose address was captured in the PCI Error Register (Section 7.11.4). Resets to 0. Cleared to 0 when the PCI Error Register is cleared.

The following seven bits enable the assertion of SERR# as an output. All bits reset to 0. The SERR# Enable (SERREN) bit in the PCI Command Register (Section 7.13.3) must be set in order to drive SERR#.

Bit 40	TASE	*Target-Abort SERR# Enable.* 1 = assert SERR# on target-abort when controller is PCI master. 0 = disable this assertion.
Bit 41	MASE	*Master-Abort SERR# Enable.* 1 = assert SERR# on master-abort when controller is PCI master. 0 = disable this assertion.
Bit 42	RTYSE	*Retry-Limit-Exceeded SERR# Enable.* 1 = assert SERR# on retry-limit-exceeded when controller is PCI master. 0 = disable this assertion.
Bit 43	PERSE	*Data-Parity Error SERR# Enable.* 1 = assert SERR# on data even-parity error (reads or writes) when controller is PCI master. 0 = disable this assertion. Such data parity errors only occur if the Parity Error Response (PEREN) bit is set in the PCI Command Register (Section 7.13.3).
Bit 44	DTIMSE	*Discard-Timer Expired SERR# Enable*. 1 = assert SERR# on discard-timer expired when controller is PCI target. 0 = disable this assertion. Discard-timer expired is only an error when it occurs on reads in which the data is not prefetchable. If the data is prefetchable, it is silently discarded. Data is prefetchable if the PREFETCHABLE bit in the PCI Base Address Register (BAR) for this device (Section 7.13.10) is set, or the PCI command was a Memory Read Line or Memory Read Multiple.
Bit 45	AERSE	*Address-Parity Error SERR# Enable.* 1 = assert SERR# on address even-parity error for all PCI transactions. 0 = disable this assertion. Address parity errors only occur if the Parity Error Response (PEREN) bit is set in the PCI Command Register (Section 7.13.3).
Bit 46	INT1SE	*Int#[1] SERR# Enable.* 1 = assert SERR# when Int#[1] is asserted. 0 = disable this assertion. This function should only be enabled when the con-

		troller needs to indicate a system error to a PCI host CPU, *and* the controller is not the Main CPU in the system. The function is independent of the state of the Int#[1] Controller Output Enable (IL1OE) bit in the Interrupt Status 1/CPU Interrupt Enable Register (Section 5.5.4). The PCI SERR# Interrupt Priority (PCISPRI) field of the Interrupt Control Register (Section 5.5.2) should not be equal to 0x1 (no loop-back).
Bit 47	*reserved*	Hardwired to 0.

The following six bits (53:48) enable the assertion of a PCI Internal Error interrupt, if such interrupts are enabled by the PCIEEN bit of the Interrupt Control Register (INTCTRL, Section 5.5.2). All six bits reset to 0. A *PCI Internal Error* indicates that something bad happened during a PCI transaction; the fault could lie either with the PCI device or the controller.

Bit 48	TAIN	*Target-Abort PCI Internal Error Enable.* 1 = assert PCI internal error on target-abort when controller is PCI master. 0 = disable this assertion.
Bit 49	MAIN	*Master-Abort PCI Internal Error Enable.* 1 = assert PCI internal error on master-abort when controller is PCI master. 0 = disable this assertion.
Bit 50	RTYIN	*Retry-Limit-Exceeded PCI Internal Error Enable.* 1 = assert PCI internal error on retry-limit-exceeded when controller is PCI master. 0 = disable this assertion.
Bit 51	PERIN	*Data-Parity Error PCI Internal Error Enable.* 1 = assert PCI internal error on data even-parity error (reads or writes) when controller is PCI master. 0 = disable this assertion. This bit is independent of the Parity Error Response (PEREN) bit in the PCI Command Register (Section 7.13.3).
Bit 52	DTIMIN	*Discard-Timer Expired PCI Internal Error Enable*. 1 = assert PCI internal error on discard-timer expired when controller is PCI target. 0 = disable this assertion. Discard-timer expired is only an error when it occurs on reads in which the data is not prefetchable. If the data is prefetchable, it is silently discarded. Data is prefetchable if the PREFETCHABLE bit in the PCI Base Address Register (BAR) for this device (Sec-

		tion 7.13.10) is set, or the PCI command was a Memory Read Line or Memory Read Multiple.
Bit 53	AERIN	*Address-Parity Error PCI Internal Error.* 1 = assert PCI internal error on address even-parity error for all PCI transactions. 0 = disable this assertion. Address parity errors only occur if the Parity Error Response (PEREN) bit is set in the PCI Command Register (Section 7.13.3).
Bit 55:54	*reserved*	Hardwired to 0.
Bit 56	INTAEN	*Int#[0]-On-INTA# Enable.* 1 = enable INTA# to be driven with Int#[0] value. 0 = disable (reset value). This function should only be enabled when the controller needs to interrupt the CPU when a PCI interrupt occurs, *and* the controller is not the Main CPU in the system. The function is independent of the state of the Int#[0] Controller Output Enable (IL0OE) bit in the Interrupt Status 1/CPU Interrupt Enable Register (Section 5.5.4). The Interrupt Signal INTA# Priority (INTAPRI) field of the Interrupt Control Register (Section 5.5.2) should not be equal to 0x0 (no loopback).
Bit 58:57	*reserved*	Hardwired to 0.
Bit 59	LATDIS	*Input-Latch Disable.* 1 = disable. 0 = enable (reset value). The PCI signals have input latches which are normally closed when the PCI clock is High. This helps to ensure the 0-ns hold time on these inputs. When this bit is set, the input latches are transparent.
Bit 60	PLL_SYNC	*PLL Synchronization.* 1 = resets the divide-by-2 at the output of the controller’s internal 2x multiplying PLL. Used for test purposes. 0 = no explicit synchronization (reset value).
Bit 61	PLL_STBY	*PLL Standby.* 1 = turn off the PLL (reset value). 0 = turn on the PLL. If PCICR# is asserted and the CLKSEL[2:0] field in the PCI Control Register (Section 7.11.1) is 001, the controller automatically clears this bit at the end of reset to enable its internal PLL. After the PLL is

turned on, the system must give it time to lock up before clearing the PCI Warm Reset bit (bit 62, immediately below). The current PLL specification *(CB-C9 Multiplying APLL Data Sheet)* requires a t_{lock} of 100ms.

Bit 62 PCIWRST *PCI Warm Reset.*
1 = PCI warm reset.
0 = normal operation.
This bit functions differently, depending on the controller's configuration, as shown in table below.

PCICR# Signal	CPU Present	Function of PCIWRST Bit
asserted	always	PCIRST# signal is asserted while this bit is set. Resets to 1.
negated	yes	All PCI accesses to controller as target are retried while this bit is set. Resets to 1.
negated	no	All PCI accesses to controller as target are retried while this bit is set. Resets to 0.

Setting this bit allows the CPU to program the PCI Configuration Space Registers (Section 7.13) before making the controller visible as a PCI target.

Bit 63 PCICRST *PCI Cold Reset*.
1 = reset controller PCI logic and (if PCICR# is asserted) assert PCIRST#.
0 = normal operation (reset value).
When this bit is set, all PCI configuration registers take their reset values, and all data and pending operations in the PCI FIFOs are lost.

7.11.2 PCI Arbiter Register (PCIARB)

This register controls the operation of the PCI arbiter when the controller performs the PCI Central Resource functions (PCICR# asserted). Up to six devices can request access to the PCI Bus: five request via the REQ#[4:0] signals and the sixth is the controller itself as a PCI-Bus master (initiator).

The controller implements three-level rotating priority arbitration. Each of the six requestors can be in any (or several) of three groups. For any given access, one group will have the highest priority, as shown in Table 25. For any group, the longest-pending request has priority. If there is no requestor in that group, then the longest-pending request at the next-lowest priority group wins.

Table 25: Three-Level Rotating PCI-Bus Arbitration Priority

when Highest-Priority Level Is:	... Middle-Priority Level Is:	... And Lowest-Priority Level Is:
Group 0	Group 1	Group 2
Group 1	Group 2	Group 0
Group 2	Group 0	Group 1

Bit 5:0 GROUP0 *Requestors Allowed In Group 0*.

Bit	Requestor
0	REQ#[0] allowed
1	REQ#[1] allowed
2	REQ#[2] allowed
3	REQ#[3] allowed
4	REQ#[4] allowed
5	Controller allowed

Resets to 0x3F (all 1s), which allows all requestors in Group 0.

Bit 7:6 *reserved* Hardwired to 0.

Bit 13:8 GROUP1 *Requestors Allowed In Group 1*.
Same bit-values as the GROUP0 field. Resets to 0x00, which allows no requestors in Group 1.

Bit 15:14 *reserved* Hardwired to 0.

Bit 21-16 GROUP2 *Requestors Allowed In Group 2*.
Same bit-values as the GROUP0 field. Resets to 0x00, which allows no requestors in Group 2.

Bit 23:22 *reserved* Hardwired to 0.

Bit 27:24 CONS0 *Group 0 Highest-Priority Consecutive Accesses.*
The number of consecutive accesses for which Group 0 has highest priority. 1 to 15 means 1 to 15 consecutive accesses, 0 means 16 consecutive accesses. Resets to 0x1. (See the example following Bit 63, below.)

Bit 31:28 CONS0n *Group 0 Non-Highest-Priority Consecutive Accesses.*
The number of consecutive accesses for which Group 0 does not have highest priority. 1 to 15 means 1 to 15 consecutive accesses, 0 means 16 consecutive accesses. Resets to 0x1. (See the example following Bit 63, below.)

Bit 35:32 CONS1 *Group 1 Highest-Priority Consecutive Accesses.*
Of the Group 0 non-highest-priority accesses, the

number of consecutive accesses for which Group 1 has highest priority. 1 to 15 means 1 to 15 consecutive accesses, 0 means 16 consecutive accesses. Resets to 0x1. (See the example following Bit 63, below.)

Bit 39:36 CONS2 *Group 2 Highest-Priority Consecutive Accesses.*
Of the Group 0 non-highest-priority accesses, the number of consecutive accesses for which Group 2 has highest priority. 1 to 15 means 1 to 15 consecutive accesses, 0 means 16 consecutive accesses. Resets to 0x1. (See the example following Bit 63, below.)

Bit 43:40 PARK0 *Group 0 Park Count (in PCI clocks).*
If the current Group is 0, and the next-to-be-granted Group is also 0, and there are no Group 0 requests asserted, then instead of immediately granting access to a lower-priority-group requestor, arbitration is parked in Group 0 for this many PCI clocks. During this time, only Group 0 requestors are serviced. Resets to 0.

Bit 47:44 PARK1 *Group 1 Park Count (in PCI clocks).*
Same parameter-type as PARK0. Resets to 0.

Bit 51:48 PARK2 *Group 2 Park Count (in PCI clocks).*
Same parameter-type as PARK0. Resets to 0.

Bit 54:52 DEFGNT *Default Grant Device.*

DEFGNT Value	**Default device**
0	GNT#[0] device
1	GNT#[1] device
2	GNT#[2] device
3	GNT#[3] device
4	GNT#[4] device
5	Controller (reset value)
7:6	No default asserted

This field specifies the device that is granted the bus when there are no requests asserted on REQ#[4:0] and the Park Counter (PARK2:0) has expired. Resets to 0x5 (controller is default). Asserting the default grant does not modify the rotating priority group.

Bit 62:55 *reserved* Hardwired to 0.

Bit 63 ARBDISABLE *Arbitrator Disable.*
1 = disable.
0 = enable (reset value).
Disables the internal arbiter, even if PCICR# is

asserted. When this bit is set to 1 (disabled), the controller uses REQ#[0] and GNT#[0] to communicate with an external arbiter.

The *CONS0* through *CONS2* fields are used to specify the number of PCI accesses in which each arbitration group has priority. Group 0 is highest priority for *CONS0* consecutive accesses, followed by *CONS0n* consecutive accesses, split between Group 1 and Group 2. Considering just the *CONS0n* accesses, Group 1 is highest priority for *CONS1* consecutive accesses, followed by Group 2 highest for *CONS2* consecutive accesses.

Here is an example: assume *CONS0* = 5, *CONS0n* = 3, *CONS1* = 4, *CONS2* = 3. Then highest priority will be: 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 2, 2, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0, 1, 1, 2...

- Total Arbitration Events = *CONS0* + *CONS0n.*
- Fraction that Group 0 accesses gets PCI Bus = *CONS0* / Total Arbitration Events.
- Fraction that non-Group 0 accesses gets bus = *CONS0n* / Total Arbitration Events.
- Fraction of non-Group 0 accesses that Group 1 gets bus = *CONS1* / *(CONS1* + *CONS2).*
- Fraction of non-Group 0 accesses that Group 2 gets bus = *CONS2* / *(CONS1* + *CONS2).*

7.11.3 PCI Master (Initiator) Registers 0 and 1 (PCIINITn)

There are two PCI Master (Initiator) Registers, PCIINIT0 and PCIINIT1, one for each of the two PCI Address Windows specified by Physical Device Address Registers PCIW0 and PCIW1 (Section 5.4), respectively. These address windows can be accessed by the CPU, DMA, or Local-Bus devices with the controller acting as the PCI-Bus master. The PCIINIT0 and PCIINIT1 registers both have the same format:

Bit 0	*reserved*	Hardwired to 0.
Bit 3:1	TYPE	*PCI Command Type.* The upper three bits of the 4-bit PCI command type (Table 21) driven on C/BE#[3:0] at the beginning of the PCI access. Resets to b011 (Memory Read and Memory Write). The low bit of the command type is 0 for reads and 1 for writes. As PCI-Bus master, the controller can access any PCI space and perform any valid PCI command. The PCI I/O Space, for example, can be accessed by programming the TYPE field to b001; the PCI Configuration Space can be accessed by programming the TYPE field to b101. See Section 7.4.3 for more information.
Bit 4	ACCESS_32	*32-Bit Access.* 1 = 32-bit. 0 = PCI-Bus width as initialized (32- or 64-bit).

		Setting this bit forces a PCI access to be a 32-bit, even if the 64-bit bus extension is implemented (PCI64# asserted at reset). Normally, when a 64-bit PCI Bus is implemented, the controller attempts a 64-bit access and falls back to 32-bit only if the target requires it. This bit must be set when the controller, as PCI-Bus master, accesses the PCI I/O Space or Configuration Space. Resets to 0.
Bit 5	LOCK	*PCI LOCK#.* 1 = acquire or maintain Exclusive Access. 0 = no lock (reset value). See Section 3.6 of the *PCI Local Bus Specification* for details on Exclusive Access and the LOCK# signal.
Bit 6	COMBINING	*Burst-Combining*. 1 = combine bursts on memory writes. 0 = do not combine bursts (reset value). Burst-combining consists of combining a sequence of burst writes to sequential locations into a single PCI-Bus transaction. See Section 7.4.3 for details.
Bit 7	MERGING	*Byte-Merging*. 1 = merge bytes on memory writes. 0 = do not merge bytes (reset value). Byte-merging consists of merging a sequence of individual byte or word writes into a single dword PCI-Bus transaction. See Section 7.4.3 for details.
Bit 8	PREFETCHABLE	*Prefetch Enable.* 1 = enable prefetching on memory reads. 0 = disable (reset value). On reads, setting this bit enables the controller to prefetch additional data beyond that which is immediately requested by the CPU or DMA. See Section 7.4.4.2 for details.
Bit 9	CONFIGTYPE	*PCI Configuration-Space Access Type.* 1 = type 1 access. 0 = type 0 access (reset value). When the controller initiates accesses to the PCI Configuration Space, this bit indicates whether they are Type 0 or Type 1 accesses. Type 0 accesses (PCI_AD[1:0] = 00) select a device on the same PCI Bus that the cycle is being run. Type 1 accesses (PCI_AD[1:0] = 01) pass the configuration request on to another PCI Bus.
Bits 14:10	SINGLE_PFB	*Single-Dword Prefetchable.* For CPU-initiated single dword (non-block) Memory Read commands with the PREFETCHABLE bit set, this field specifies the number of 4-dword blocks to

		prefetch beyond the first block. See Section 7.4.4.2 for details. Resets to 0x01.
Bits 20:15	BLOCK_PFB	*Block Prefetchable.* For CPU-initiated block Memory Read commands with the PREFETCHABLE bit set, this field specifies the number of 4-dword blocks to prefetch beyond the first block. See Section 7.4.4.2 for details. Resets to 0x01.
Bits 35:21	PCIADD	*PCI Address (Lower).* The lower PCI physical address bits. These bits are to be masked by the MASK field of the PDAR (Section 5.4). Resets to 0x0. The CPU provides the lowest address bits—those from bit 0 up to the highest bit masked by the MASK field. See the PCI Address Decoding Example, Section 7.4.2.
Bits 63:36	PCIADD	*PCI Address (Upper).* The upper PCI physical address bits. Resets to 0x0. See the PCI Address Decoding Example, Section 7.4.2.

7.11.4 PCI Error Register (PCIERR)

This register captures the PCI address of last uncleared PCI error, if address capture is enabled by bits 36:32 of the PCI Control Register (Section 7.11.1). Writing anything to this register clears it to 0, and also clears the PCI Control Register ERRTYPE to 0. Due to clock synchronization requirements, it may take several CPU clocks for this register (and ERRTYPE) to be cleared after this register is written.

Bit 0	IS_CPU	*Initiator Is CPU.* 1 = on PCI master error, master was CPU. 0 = on PCI master error, master was not CPU (reset value). Five types of PCI master errors are reported by this bit. For PCI target errors (Discard Timer Expired) this bit is always 0. The type of master or target error is reported in the ERRTYPE field of the PCI Control Register (PCICTRL), Section 7.11.1.
Bit 1	*reserved*	Hardwired to 0.
Bit 63:2	ADDR	*PCI Error Address.* The PCI address where an error occurred. Resets to 0.

7.12 PCI Configuration Space Cycles

The controller supports a PCI Configuration Space, as defined in the *PCI Local Bus Specification*. Only the Main CPU in a PCI system should run PCI Configuration Space cycles. When it does so, the PCI Central Resource (Section 7.8) is responsible for generating the IDSEL inputs (Section 7.12.3) to PCI devices. These IDSEL inputs are the chip-selects during PCI Configuration Space accesses by the Main CPU. The PCI Central Resource itself does not have a PCI Configuration Space.

The registers that make up the PCI Configuration Space are described in Section 7.13. Setting the PCIWRST bit in the CPU Status Register (CPUSTAT), Section 5.5.1, allows the CPU to program the PCI Configuration Space registers before making the controller visible as a PCI target.

The concepts of PCI Configuration Space Cycles and Main Controller (Section 5.3.2) are unrelated. The controller can be a Main Controller for a given CPU, but that CPU might not be the Main CPU in the system, and the Main Controller for that CPU, or any other CPU in the system, might not provide the PCI Central Resource for the system. Only the Main CPU in a PCI system should run PCI Configuration Space Cycles.

7.12.1 As PCI-Bus Master and Target

As a PCI-Bus master, the controller accesses the PCI Configuration Space when the TYPE field in the PCI Master (Initiator) Registers (PCIINITn, Section 7.11.3) contains the value b101. When this is done, reads and writes on the PCI Bus are Configuration Reads and Configuration Writes. As a PCI master, the controller can generate any arbitrary PCI Configuration Address, whether Type 0 or Type 1.

As a PCI target, the controller responds only to PCI Configuration transactions (a) when its IDSEL input is asserted and (b) that are Type 0 (device on this bus) and Function Number 0. All PCI Configuration writes to the controller are delayed writes. The address and data for a configuration write is placed in the INFIFO, and the PCI transaction is terminated with Retry. When the delayed write has been performed into the Configuration Register, the transaction is allowed to complete. This mechanism is normally used to configure the controller when it is in PCI Stand-Alone Mode (i.e., when it is not the Central Resource), but it can also be used when the controller is the Central Resource; i.e. the controller can talk to itself in PCI Configuration Space.

7.12.2 Configuration Mechanisms

The controller does not use the PCI Configuration Mechanism #1 or #2, described in Section 3.7.4.1 and 3.7.4.2 of the *PCI Local Bus Specification*. Because the VR5000 CPU has more than 32 physical address bits, the entire Configuration Space can be memory-mapped into the normal CPU address space. Every 32-bit Configuration Address can be accessed directly by properly setting the PCIADD field in the PCIINITn register (Section 7.11.3). When this is done, the PCIADD field specifies the upper address bits and the CPU generates the lower address bits. Addresses are generated the same way for memory, I/O, and Configuration Space.

During Configuration Space accesses, the low two bits of the PCI address specify the type of access. The type is controlled by the CONFIGTYPE field of the PCIINITn register. For Type 0 (device on this bus) the low two address bits are b00. For Type 1 (device across a bridge) the low two bits are b01. The Type 0 and Type 1 accesses are illustrated in Figure 3-19 of the *PCI Local Bus Specification*.

Only 32-bit Configuration Space accesses are allowed by the *PCI Local Bus Specification;* 64-bit accesses should not be attempted. The ACCESS_32 bit should be set in the PCIINITn register. Combining (Section 7.4.3.1) and Merging (Section 7.4.3.2) may be used.

7.12.3 Generating IDSEL Inputs

The IDSEL (Initialization Device Select) input to a PCI device is used as the chip-select during PCI Configuration Space accesses. These IDSEL inputs should be generated by the PCI Central Resource. This can be done by resistive coupling to the PCI_AD[31:16] signals. The controller is designed so that the *Implementation Note: System Generation of IDSEL*, in Section 3.7.4 of the *PCI Local Bus Specification* can be followed.

The controller pre-drives addresses during Configuration Space cycles in order to provide additional time for the resistively coupled IDSEL signals to become valid before the controller asserts FRAME#. The address is driven 8 clocks before FRAME# is asserted.

7.13 PCI Configuration Space Registers

Table 26 summarizes the registers that make up the PCI Configuration Space. These registers are visible to the PCI Central Resource when the controller's IDSEL input is asserted. The Configuration Space registers are also visible in the controller's internal register address space (Table 8). The internal address is determined by adding an offset of 0x200 to the base address of the Controller Internal Registers and Devices (INTCS) PDAR, and then adding the offset of the Configuration Space register shown in Table 26. This provides two paths for accessing the same register.

Some register bits are writable only when accessed via the internal register space, and they appear read-only to PCI-Configuration-Space accesses. This allows flexibility in programming the controller and yet retains compatibility with the PCI specification.

Table 26: PCI Configuration Space Register Summary

Name	Symbol	Offset	R/W	Reset Value	Description
PCI Vendor ID	VID	0x01:0x00	R	0x1033	Vendor ID for NEC, assigned by PCI Special Interest Group.
PCI Device ID	DID	0x03:0x02	R	0x005A	Device ID for the controller, assigned by NEC.
PCI Command	PCICMD	0x05:0x04	R/W	0x0000 or 0x0006 [a]	Coarse control of PCI interface.
PCI Status	PCISTS	0x07:0x06	R/W	0x02A0	Status of PCI events.
PCI Revision ID	REVID	0x08	R	0x01, 0x02, or 0x03	Device revision.
PCI Class Code	CLASS	0x0B-0x09	R	0x06 0000	Device type.
PCI Cache Line Size	CLSIZ	0x0C	R/W	0x00	System cache-line size, in 32-bit words.
PCI Latency Timer	MLTIM	0x0D	R/W	0x00	Minimum guaranteed clocks for PCI Bus master.
PCI Header Type	HTYPE	0x0E	R	0x00	Configuration register layout.
BIST	*unimplemented*	0x0F	R	0x00	*Hardwired to 0.*
PCI Base Address Register Control	BARC	0x17:0x10	R/W	0x0000 0000 0000 0004	PCI base address of the controller's internal control registers and devices. This register corresponds to the INTCS Physical Device Address Register (Section 5.4).
PCI Base Address Register 0	BAR0	0x1F-0x18	R/W	0x0000 0000 0000 0000	PCI base address of RAM bank 0. This register corresponds to the SDRAM0 Physical Device Address Register (Section 5.4).

Table 26: PCI Configuration Space Register Summary (continued)

Name	Symbol	Offset	R/W	Reset Value	Description
PCI Base Address Register 1	BAR1	0x27:0x20	R/W	0x0000 0000 0000 0000	PCI base address of RAM bank 1. This register corresponds to the SDRAM1 Physical Device Address Register (Section 5.4).
PCI Cardbus CIS Pointer	*unimplemented*	0x2B-0x28	R	0x0000 0000	*Hardwired to 0.*
PCI Sub-System Vendor ID	SSVID	0x2D-0x2C	R(W) [b]	*depends on various conditions*	Read-only value set from Serial Mode EEPROM.
PCI Sub-System ID	SSID	0x2F-0x2E	R(W) [b]	*depends on various conditions*	Read-only value set from Serial Mode EEPROM.
Expansion ROM Base Address	*unimplemented*	0x33:0x30	R	0x0000 0000	*Hardwired to 0.*
reserved	—	0x3B-0x34	R	0x00	*Hardwired to 0.*
PCI Interrupt Line	INTLIN	0x3C	R/W	0xFF	Interrupt-signal routing information.
PCI Interrupt Pin	INTPIN	0x3D	R	0x01	The controller drives INTA#
PCI Min_Gnt	*unimplemented*	0x3E	R	0x00	*Hardwired to 0.*
PCI Max_Lat	*unimplemented*	0x3F	R	0x00	*Hardwired to 0.*
PCI Base Address Register 2	BAR2	0x47:0x40	R/W	0x0000 0000 0000 0000	PCI base address of device selected by DCS#[2]. This register corresponds to the DCS2 Physical Device Address Register (Section 5.4).
PCI Base Address Register 3	BAR3	0x4F-0x48	R/W	0x0000 0000 0000 0000	PCI base address of device selected by DCS#[3]. This register corresponds to the DCS3 Physical Device Address Register (Section 5.4).
PCI Base Address Register 4	BAR4	0x57:0x50	R/W	0x0000 0000 0000 0000	PCI base address of device selected by DCS#[4]. This register corresponds to the DCS4 Physical Device Address Register (Section 5.4).
PCI Base Address Register 5	BAR5	0x5F-0x58	R/W	0x0000 0000 0000 0000	PCI base address of device selected by DCS#[5]. This register corresponds to the DCS5 Physical Device Address Register (Section 5.4).
PCI Base Address Register 6	BAR6	0x67:0x60	R/W	0x0000 0000 0000 0000	PCI base address of device selected by DCS#[6]. This register corresponds to the DCS6 Physical Device Address Register (Section 5.4).
PCI Base Address Register 7	BAR7	0x6F-0x68	R/W	0x0000 0000 0000 0000	PCI base address of device selected by DCS#[7]. This register corresponds to the DCS7 Physical Device Address Register (Section 5.4).
PCI Base Address Register 8	BAR8	0x77:0x70	R/W	0x0000 0000 0000 0000	PCI base address of device selected by DCS#[8]. This register corresponds to the DCS8 Physical Device Address Register (Section 5.4).
PCI Base Address Register BOOT	BARB	0x7F-0x78	R/W	0x0000 0000 0000 0004	PCI base address of the Boot ROM. This register corresponds to the BOOTCS Physical Device Address Register (Section 5.4).
reserved	—	0xFF-0x80	R	0x00	*Hardwired to 0.*

[a] PCICMD resets to 0x0000 when PCICR# is negated, or to 0x0006 when PCICR# is asserted.

[b] Read-only from PCI Configuration Space, read-write from the controller’s internal register space.

7.13.1 PCI Vendor ID Register (VID)

Bit 15:0	VID	Hardwired to 0x1033 for NEC PCI devices. Assigned by PCI Special Interest Group (SIG).

7.13.2 PCI Device ID Register (DID)

Bit 15:0	DID	Hardwired to 0x005A for the controller. Assigned by NEC.

7.13.3 PCI Command Register (PCICMD)

Bit 0	IOEN	*PCI I/O Space Target Enable.* 1 = (not valid) 0 = disable. (hardwired to 0) As a PCI-Bus target, the controller responds only to PCI memory and configuration space accesses, not to PCI I/O space accesses (although it can perform accesses to PCI I/O space as a PCI-Bus master).
Bit 1	MEMEN	*PCI Memory Space Target Enable.* 1 = enable. (reset value when PCICR# asserted) 0 = disable. (reset value when PCICR# negated) Enables the controller to respond to PCI memory space accesses as a PCI-Bus target.
Bit 2	BMASEN	*PCI-Bus Master Enable.* 1 = enable. (reset value when PCICR# asserted) 0 = disable. (reset value when PCICR# negated) Enables the controller to act as a master on the PCI Bus.
Bit 3	SPCEN	*PCI Special Cycle Enable.* 1 = (not valid) 0 = disable. (hardwired to 0) The controller ignores Special Cycles.
Bit 4	MWIEN	*Memory Write and Invalidate Enable.* 1 = (not valid) 0 = disable. (hardwired to 0) In normal operation, the controller does not generate Memory Write and Invalidate accesses. However, for testing purposes the TYPE field in the PCI Master (Initiator) Register (PCIINITn, Section 7.11.3) can be programmed so as to generate any PCI command listed in Table 21.
Bit 5	VGA	*VGA Palette Snoop.* 1 = (not valid) 0 = disable. (hardwired to 0) The controller is not a VGA device.
Bit 6	PEREN	*Parity Error (PERR#) Enable.* 1 = respond to even-parity data error.

		0 = ignore such parity errors (reset value). See PCI-Master Parity Detection (Section 7.4.5) and PCI-Target Parity Detection (Section 7.5.4) for details on parity-error handling.
Bit 7	WCYC	*Wait-Cycle Control.* 1 = (not valid) 0 = no address or data stepping. (hardwired to 0) The controller does not do address or data stepping (although it does pre-drive addresses during PCI Configuration Space Cycles so that IDSEL will be valid when the controller asserts FRAME#, as described in Section 7.12).
Bit 8	SERREN	*System Error (SERR#) Enable.* 1 = assert SERR# signal on system error. 0 = disable SERR# assertion (reset value). See Section 7.4.5, Section 7.5.4 and Section 7.11.1 (bits 46:40) for details on parity-error handling.
Bit 9	FBBEN	*Fast Back-to-Back Enable.* 1 = enable fast back-to-back transactions. 0 = enable such transactions (reset value). This bit specifies whether the controller, as master, is allowed to perform fast back-to-back PCI-Bus transactions.
Bit 15:10	*reserved*	Hardwired to 0.

7.13.4

PCI Status Register (PCISTS)

These status bits are set to 1 when the indicated event occurs. Writing a 1 to a bit causes it to be cleared to 0.

Bit 4:0	*reserved*	Hardwired to 0.
Bit 5	66M	*66 MHz Capable.* 1 = enabled. (hardwired to 1) 0 = (not valid)
Bit 6	UDF	*User-Definable Features Supported.* 1 = (not valid) 0 = disable. (hardwired to 0) The controller does not support User Definable Features.
Bit 7	FBBC	*Fast Back-to-Back Capable.* 1 = capable of fast back-to-back. (hardwired to 1) 0 = (not valid) This bit specifies whether the controller, as target, is capable of accepting fast back-to-back PCI-Bus transactions.

Bit 8 DPR *Data-Parity Error Reported.*
1 = master or target asserted PERR#.
0 = parity error cleared (reset value).
The controller sets this bit if the controller initiated a PCI transaction and asserted PERR# on a read or detected asserted PERR# by the target on a write. This bit can only be set if the PEREN bit is set in the PCI Command Register (Section 7.13.3).

Bit 10:9 DEVSEL *DEVSEL# Timing.*
Hardwired to 01, to specify that the controller uses medium response time (2 clocks after the address phase) when driving the DEVSEL# output signal as a PCI target.

Bit 11 STA *Signaled Target-Abort.*
1 = (not valid)
0 = no Target-Abort generated. (hardwired to 0)
The controller never generates Target Abort.

Bit 12 RTA *Received Target-Abort.*
1 = controller, as master, received a Target-Abort.
0 = Target-Abort cleared (reset value).

Bit 13 RMA *Received Master-Abort.*
1 = controller, as master, received a Master-Abort.
0 = Master-Abort cleared (reset value).

Bit 14 SSE *Signaled System Error.*
1 = controller asserted SERR#.
0 = system error cleared (reset value).
The controller sets this bit if the controller asserted SERR# (i.e. detected an address even-parity error or other system error). This bit can only be set if the SERREN bit is set in the PCI Command Register (Section 7.13.3).

Bit 15 DPE *Detected Parity Error.*
1 = controller detected an even-parity error.
0 = parity error cleared (reset value).
This bit is set on any even-parity error (address or data, read or write) even if the PEREN bit is cleared in the PCI Command Register (Section 7.13.3).

7.13.5 PCI Revision ID Register (REVID)

Bit 7:0 REVID *Revision ID.*
Hardwired to indicate the version of the controller.

Tapeout	Revision ID
July 1997	0x01 (pre-production)
March 1998	0x02

7.13.6 PCI Class Code Register (CLASS)

Bit 7:0 PROGINT *Programming Interface Code.*
Hardwired to 0x00.

Bit 15:8 SUBCL *Sub-Class Code.*
Hardwired to 0x00 indicating a Host Bridge.

Bit 23:16 BASECL *Base Class Code*.
Hardwired to 0x06 indicating a Bridge Device.

7.13.7 PCI Cache-Line Size Register (CLSIZ)

Bits 7:0 CLSIZ *Cache-Line Size.*
The PCI cache-line size, in units of 32-bit words. The controller uses this value to determine how much data to prefetch during PCI target reads, and for command coercion on prefetchable PCI master reads. Valid values are 0, 1, 2, 4, 8, 16, 32, 64, 128 words. Writing anything else forces the value to 0. Resets to 0.

7.13.8 PCI Latency Timer Register (MLTIM)

Bit 7:0 MLTIM *Latency Timer.*
This register specifies, in PCI clocks, the minimum number of PCI clocks that the controller can hold the bus as a master after its GNT#[n] is negated. Resets to 0.

7.13.9 PCI Header Type Register (HTYPE)

Bit 6:0 HTYPE *Header Type.*
Hardwired to 0x00, indicating header type 0.

Bit 7 SINGLEFN *Single Function.*
Hardwired to 0. The controller is a single-function PCI device.

7.13.10 PCI Base Address Registers (BARn)

The controller has 11 Base Address Registers (BARs), corresponding to 11of the PDARs (Section 5.4 on page 45). The BARs are used to control PCI-Bus master access to controller resources. The two PCI Address Window PDARs do not have corresponding BARs, as explained in Section 7.5.1. Thus, the BARs include:

BAR	PDAR(s)
Base Address Register 0 (BAR0)	SDRAM0
Base Address Register 1 (BAR1)	SDRAM1
Base Address Register8:2 (BAR8:2)	DCS[8:2]
Base Address Register Boot (BARB)	BOOTCS
Base Address Register Control (BARC)	INTCS

BARs must be programmed with non-overlapping PCI addresses. If the Visible on PCI Bus (VISPCI) bit is cleared in the corresponding PDAR, all BAR bits for the device are forced to 0, the bits cannot be written, and no access to the corresponding resource is allowed from the PCI Bus.

Bit 0	SPACE	*Memory Space Indicator.* 1 = PCI I/O space. (not valid) 0 = PCI memory space. (hardwired to 0) As a PCI-Bus target, the controller responds only to PCI memory and configuration space accesses, not to PCI I/O space accesses (although it can perform accesses to PCI I/O space as a PCI-Bus master).
Bit 2:1	TYPE	*Type.* Hardwired to b10, indicating that the controller’s address space can be located anywhere in a 64-bit address space.
Bit 3	PREFETCHABLE	*Prefetchable.* 1 = enable prefetching on reads to this region. 0 = disable prefetching in this region (reset value). When set, this bit indicates that the device returns all bytes on reads, regardless of byte-enables, and that writes can be merged without causing errors. Read-only via PCI Configuration Space (offset shown in Table 26. Read-write when accessed as an internal register (offset shown in Table 26, plus 0x200).
Bit 63:4	BASEADDR	*Base Address.* The PCI starting address for this device. Bits 31:21 are forced to 0, if masked by the MASK field in the corresponding PDAR (Section 5.4). Bits 20:4 are hardwired to 0. Resets to 0.

7.13.11 PCI Sub-System Vendor ID (SSVID)

Bit 15:0	SSVID	*Sub-System Vendor ID.* This ID is issued to sub-system or add-in board vendors by the PCI Special Interest Group. It is intended to uniquely identify the board or sub-system where the PCI device resides. Reset value provided by Serial Mode EEPROM. Read-only via PCI Configuration Space (offset shown in Table 26). Read-write when accessed as an internal register (offset shown in Table 26, plus 0x200).

7.13.12 PCI Sub-System ID (SSID)

Bit 15:0	SSID	*Sub-System ID.* This ID is vendor-specific, and can be used to identify board revisions. Reset value provided by Serial Mode EEPROM. Read-only via PCI Configuration Space (offset shown in Table 26. Read-write when accessed as an internal register (offset shown in Table 26, plus 0x200).

7.13.13

PCI Interrupt Line Register (INTLIN)

Bit 7:0	INTLIN	*PCI Interrupt Line.* Holds the PCI interrupt-signal routing code for use by system software. See Section 2.2.6 of the *PCI Local Bus Specification* for an example. The controller ignores the contents of this register. Resets to 0xFF.

7.13.14

PCI Interrupt Pin Register (INTPIN)

Bit 7:0	INTPIN	*PCI Interrupt Pin.* Hardwired to 0x01, indicating that the controller uses INTA# to request a PCI interrupt.

8.0 Local-Bus Interface and Registers

The LOC_AD[31:0] and PCI_AD[63:32] signals, and a few other related signals, share the same pins on the controller package, so that when the controller's PCI interface is configured for 32-bit operation, a 32-bit Local Bus is available for I/O and memory devices (such as boot memory).

The Local-Bus interface consists of:

- ❑ LOC_CLK: a Local-Bus clock, which SysClock divided by 4 or 2.
- ❑ LOC_AD[31:0]: a 32-bit multiplexed address and data bus.
- ❑ LOC_A[4:0]: a 5-bit de-multiplexed low-address and byte-enable bus.
- ❑ LOC_ALE: Address latch enable.
- ❑ LOC_FR#: Frame indication.
- ❑ LOC_RD#, LOC_WR#: Read and write signals (or a single RD/WR# signal).
- ❑ LOC_RDY#: Ready (acknowledge).
- ❑ LOC_BR#, LOC_BG#, LOC_BGACK#: Bus arbitration (68000 or Intel mode).

Two additional signals control devices that can be located either on the Local Bus or the memory bus:

- ❑ BootCS#: Boot ROM chip-select.
- ❑ DCS#[8:2]: 7 programmable chip-selects.

The controller can be a Local-Bus master (on behalf of the CPU, DMA, or PCI-Bus masters) or a Local-Bus target for accesses by masters on the Local-Bus. A Local-Bus master obtains control of the Local Bus through arbitration (68000 or Intel mode). When the controller grants control, it tri-states all of its Local-Bus outputs except LOC_CLK and LOC-BG#, so that the Local-Bus master can access other Local-Bus devices directly, or access controller resources (memory, PCI-Bus targets, or the controller's internal registers). When a Local-Bus master accesses controller resources, the controller's DMA logic carries out the Local-Bus master's request. Local-Bus masters cannot access Local-Bus targets through the controller; instead, they must do so directly on the Local Bus, without the help of the controller.

The controller supports burst transfers on the Local Bus. See Section 8.3.2.2 and Section 8.4.1 for details.

8.1 Local-Bus Configuration and Monitoring

Software configures and monitors the Local-Bus interface using the following registers:

- ❑ Physical Device Address Registers (PDARs), Section 5.4 on page 45.
- ❑ Interrupt Control Register (INTCTRL), Section 5.5.2 on page 52.
- ❑ Interrupt Status Register 0 (INTSTAT0), Section 5.5.3 on page 55.
- ❑ Interrupt Status 1/CPU Interrupt Enable Register (INTSTAT1), Section 5.5.4 on page 55.
- ❑ Interrupt Clear Register (INTCLR), Section 5.5.5 on page 56.
- ❑ Local-Bus Registers, Section 8.6 on page 120.

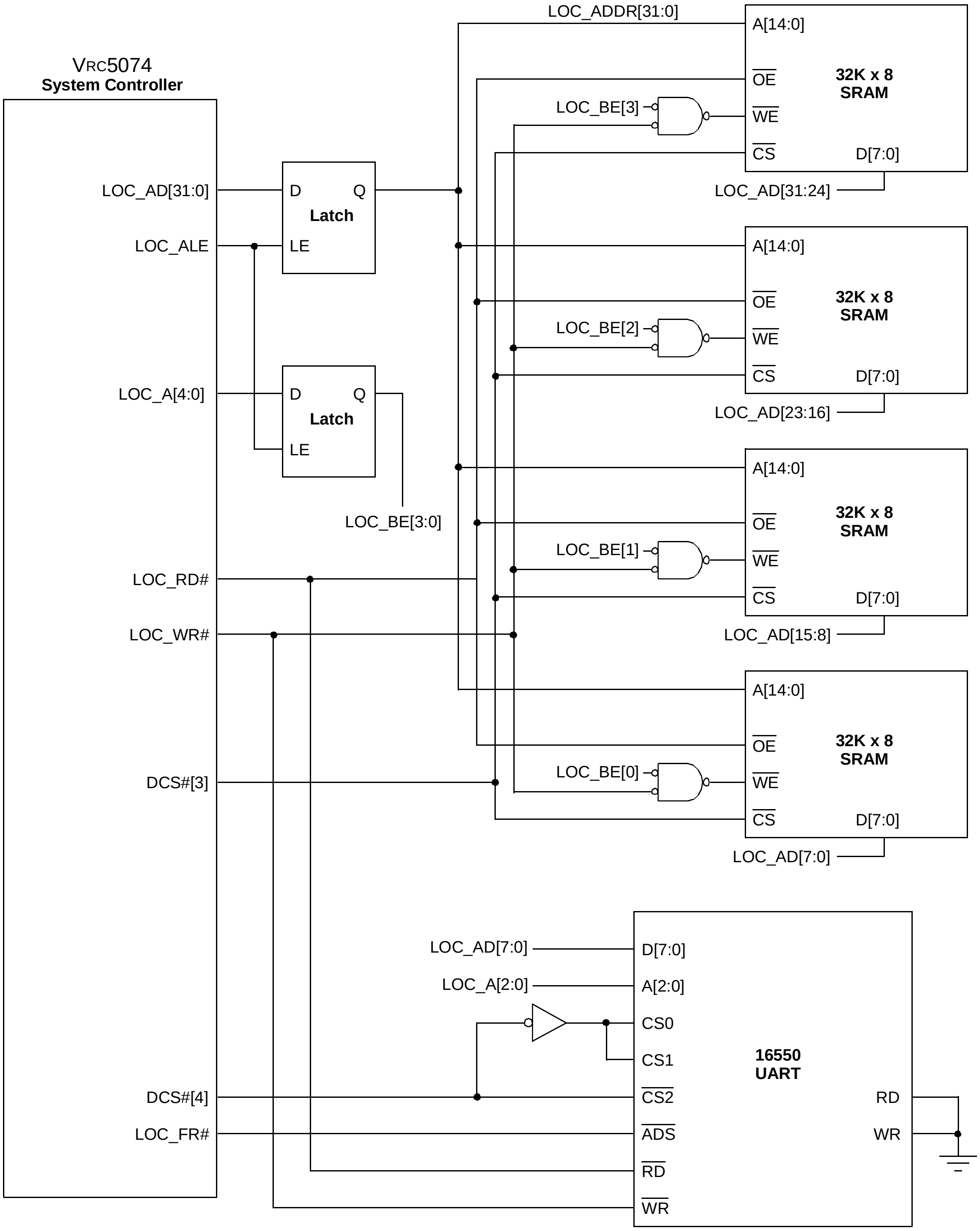
Figure 16 shows an example of a Local-Bus configuration that implements SRAM and an external UART (in addition to the controller's internal UART). The SRAMs respond to byte-enable signals, allowing single-byte granularity on writes. The connections to the external 16550 UART shows how the LOC_A[4:0] signals can be used directly for devices with small address spaces.

8.2 Device Chip-Select Configuration

The seven programmable DCS[8:2] chip-selects can be used to access devices on the Local Bus or the Memory Bus, as specified in the MEM/LOC bit in the Physical Device Address Registers (PDAR, Section 5.4) for each chip-select. The chip-selects have a flexible address map, which allows from 2MB to 4GB per chip-select. The Local Bus's control signals can be configured to customize the shape of a Local-Bus cycle in the Local-Bus Chip-Select Timing Register (LCSTn, Section 8.6.2). For example, the fields of the LCSTn register specify polarity of the chip-select and read/write (LOC_RD# or LOC_WR#) signals, the time from address-valid to chip-select asserted, the time from chip-select asserted to read/write asserted, the duration of read/write, read/write negated to chip-select negated, chip-select negated to address invalid, and bus idle time after chip-select negated.

Ready (LOC_RDY#) support is available, per chip-select, for Local-Bus devices that do not respond in a fixed amount of time (as specified in the LCSTn register). LOC_RDY# may be sampled directly off the Local Bus or after undergoing double synchronization by the controller. In LOC_RDY# mode, all bus signals are extended until LOC_RDY# is received from the target. The negation of the read/write command can be specified as relative to the assertion of LOC_RDY#, and the remaining bus signals can be specified as relative to the negation of read/write, as described above. A 12-bit programmable timer (up to 4K Local-Bus clocks) is available as a LOC_RDY# watchdog timer. This timer should be programmed to a value higher than the slowest device on the Local Bus. The timer begins counting down when a LOC_RDY#-response bus cycle begins. If a LOC_RDY# is not received before the timer reaches zero, the cycle terminates as though a LOC_RDY# were received, and an interrupt is generated, if enabled by the LBRTDEN bit of the Interrupt Control Register (INTCTRL, Section 5.5.2).

Figure 15: Example Local-Bus Configuration



8.3 Local-Bus Master Transactions (Controller-to-Local Bus)

The controller becomes the master of the Local Bus and initiates a Local Bus cycle by asserting LOC_FR#. When LOC_FR# is asserted, no other device may drive Local Bus signals, with the exceptions of providing response data to read requests and arbitrating for control of the bus by asserting LOC_BR#.

8.3.1 Timing

During the first clock of a Local-Bus cycle, the LOC_AD[31:0] bus contains the address of the request. LOC_ALE is asserted for the first half of this clock cycle to enable external latching of the Local-Bus address. The Local Bus supports byte-addressing in the local address space. Therefore, if a 16-bit device is used on the Local Bus, LOC_AD[1] would be the least-significant address bit wired to that device. Similarly, if a 32-bit device is used, LOC_AD[2] would be the least-significant address bit wired to that device.

Also during the first clock of a Local-Bus cycle, the LOC_A[3:0] signals carry active-low byte-enables—in effect, BE#[3:0] for the Local Bus—while LOC_AD[31:0] carries the address. The LOC_ALE signal may also be used to externally latch both the address and the byte-enables. For example, when a 16-bit resource is accessed, LOC_A[1:0] are the byte-enables during the first Local-Bus clock. When a 32-bit resource is accessed, LOC_A[3:0] are the byte-enables.

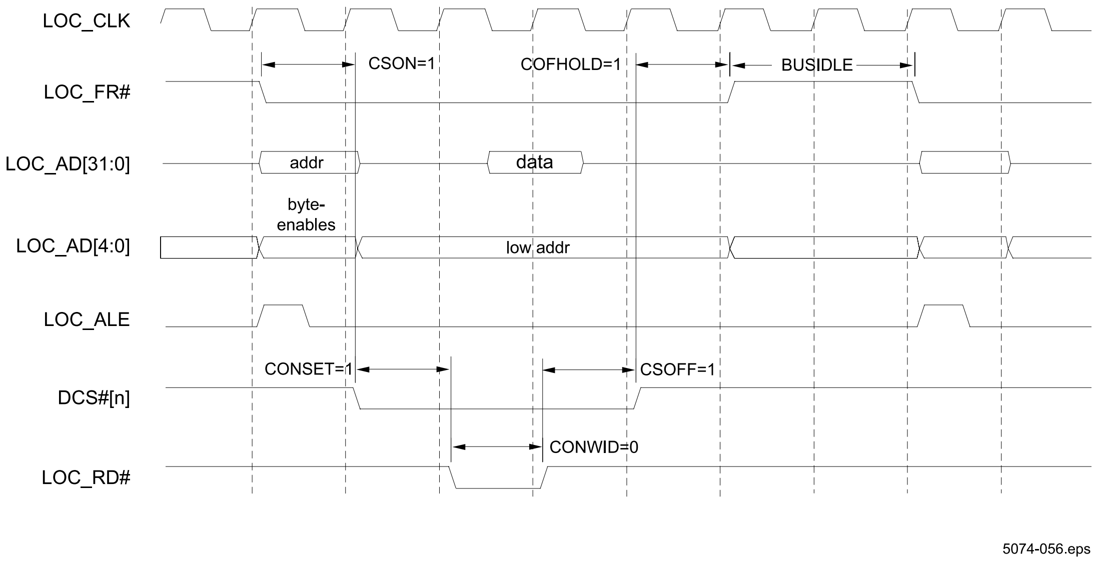
During the remainder of a non-block bus cycle, i.e. from the second Local-Bus clock through the end of the bus cycle, LOC_A[4:0] carries the five low-address bits (the same bits that were carried on LOC_AD[4:0] bits when LOC_ALE was active) while LOC_AD[31:0] carries the data. If all address spaces for Local-Bus devices are less than or equal to 32 bytes, the LOC_A[4:0] bits may be used in place of externally latching the address from the LOC_AD[4:0] bus.

During the remainder of the bus cycle, the appropriate DCS[8:2] chip-select and either the LOC_RD# or LOC_WR# signal are asserted, according to the polarity specified by the CON_POL bit of the device's Local-Bus Chip-Select Timing Register (LCSTn, Section 8.6.2). When LOC_RD# or LOC_WR# is negated, which is either a specified duration or after the detection of the LOC_RDY# signal (as specified in the CONWID and SUBSCWID fields of the LCSTn register), the cycle ends with the negation of LOC_FR#. A new cycle may begin, as indicated by the assertion of LOC_FR#, after the bus-idle time specified by the BUSIDLE field of the LCSTn register.

Figure 16 shows an example read access on the Local Bus. The CSON, CSOFF, COFHOLD, CONSET, CONWID, and BUSIDLE values are software-configuration fields in the Local Bus Chip-Select Timing Registers (LCSTn), Section 8.6.2.

If you have only 32-bit devices on an external board, none of which use burst transfers, you can connect the LOC_AD[31:0] bus to the external board without connecting the LOC_A[4:0] bus.

Figure 16: Local-Bus Read



8.3.2 Dword vs. Block Requests

Requests to resources with data sizes larger than the width of the device cause multiple bus cycles on the Local Bus. For example, a word request to a 16-bit device results in two Local-Bus cycles. However, multiple Local-Bus cycles resulting from a dword (or less) request look different than multiple Local-Bus cycles resulting from a Block request.

8.3.2.1 Dword Requests

Multiple Local-Bus cycles resulting from a dword (or less) request look like successive separate requests on the Local Bus. Each bus cycle has its own address, data (for writes) and control signals asserted. These requests differ from random requests in that they cannot be interrupted by requests from other internal controller masters (CPU or DMA) or from external Local-Bus masters.

The number of local cycles resulting from a dword request varies, depending on the size of the resource and the size of the request. The controller's Local-Bus interface performs only as many bus requests as necessary to complete the request. For example, a tri-byte write by the CPU to a byte-wide Local-Bus device results in only three Local-Bus write cycles. The only additional delay for these types of accesses is one SysClock per byte, half-word, or word of data (depending on the size of the resource being addressed) for which no byte-enables are asserted. In the case of the tri-byte example, there would be five SysClock delays around and/or between Local-Bus cycles in which the controller inspects the byte-enables from the requester to determine if a bus cycle must be performed.

8.3.2.2 Block Requests

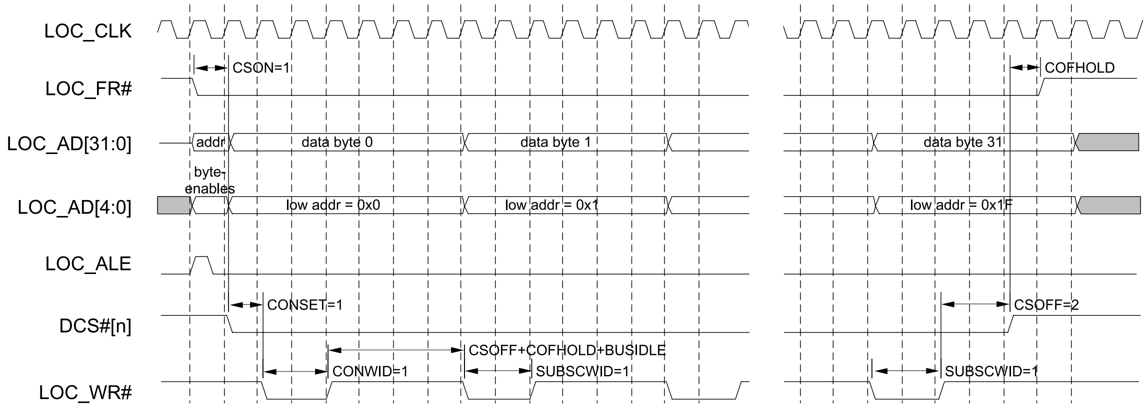
Block requests (32-byte cache line requests) to Local-Bus devices always result in multiple requests on the Local Bus. A block request to a byte-wide device results in 32 reads or writes. Block requests to half-word devices result in 16 reads or writes, and block requests to word devices result in 8 reads or writes.

These are not separate Local-Bus cycles, as described above for the case of multiple cycles resulting from a dword request. Rather, these look like one, long Local-Bus cycle with multiple assertions of LOC_RD# or LOC_WR#. There is one LOC_FR#, one LOC_ALE during the first half of the bus clock, and the address and byte-enables are only on the LOC_AD[31:0] and LOC_A[4:0] buses during the first bus clock, as shown in Figure 17. The low-order address bits are driven on LOC_A[4:0] to indicate which part of the block is being transferred. The byte-enables must all be asserted, since a full port-size unit of data is to be read or written on each assertion of LOC_RD# or LOC_WR#.

The LOC_A[4:0] bits must be connected to devices responding to block requests; during block requests, these are the only address bits that increment after each unit of data is read or written. As described above, these bits contain byte addresses. Therefore, LOC_A[0] is the least-significant address bit wired to a byte device, LOC_A[1] is the least-significant address bit wired to a half-word device, and LOC_A[2] is the least-significant bit wired to a word device.

Figure 17 shows an example block write to a byte-wide device on the Local Bus. The CSON, CSOFF, COFHOLD, CONSET, CONWID, BUSIDLE, and SUBSCWID values are software-configuration fields in the Local Bus Chip-Select Timing Registers (LCSTn), Section 8.6.2.

Figure 17: Local-Bus Block Write To Byte-Wide Device



5074-057.eps

When the ARBEN bit is set in the Local Bus Configuration Register (LCNFG, Section 8.6.1), external Local-Bus devices are allowed to arbitrate for and gain control of the Local Bus.

8.4 Arbitration for Local-Bus Control

The ARBMODE bit in the Local Bus Configuration Register (LCNFG, Section 8.6.1) specifies one of two bus-arbitration modes. Based on this selection, the LOC_BR#, LOC_BG#, and LOC_BGACK# signals are configured to function as:

- ❑ *68000 Mode:*
  - LOC_BR# = bus request (BR#)
  - LOC_BG# = bus grant (BG#)
  - LOC_BGACK# = bus-grant acknowledge (BGACK#)
- ❑ *Intel Mode:*
  - LOC_BR# = bus hold (HOLD)
  - LOC_BG# = bus-hold acknowledge (HLDA)

When a Local-Bus master gains control of the bus, the controller tri-states all of its Local-Bus outputs except LOC_CLK and LOC-BG#. If multiple masters are implemented on the Local Bus, external logic must arbitrate among those masters. If multiple masters with different arbitration modes are implemented, external logic must arbitrate among those masters and present a single arbitration mode to the controller.

8.4.1 Signal Redefinition for Local-Bus Masters

When a Local-Bus master gains control of the Local Bus, the definitions of the LOC_A[4:0] signals change, as follows:

- ❑ *LOC_A[4]:* Determines where the Local-Bus master's request is targeted:
  - LOC_A[4] = 1 requests a *controller (non-Local-Bus)* target.
  - LOC_A[4] = 0 requests a *Local-Bus* target.
- ❑ *LOC_A[3:0]:*
  - For *controller targets:* The controller floats its LOC_A[3:0] signals and these bits become address bits [35:32] inside the controller. These bits are concatenated with the 30-bit address latched from the LOC_AD[31:2] bus to form a 36-bit physical address. LOC_AD[1:0] are assumed to be 0; all accesses to controller resources by Local-Bus masters are assumed to be 32-bit accesses, because there are no byte-enable signals. This mechanism allows external Local-Bus masters to access the entire controller address space. LOC_AD[1:0] specify the length of the access. If LOC_AD[1:0] = 00, a single 32-bit word is transferred. If LOC_AD[1:0] = 01, a block of eight 32-bit words is transferred as a burst. On reads, the controller asserts LOC_RDY# to indicate when data is valid.
  - For *Local-Bus targets:* LOC_A[3:0] carry implementation-dependent information.

Thus, the LOC_A[4] bit distinguishes two, separate address spaces, one for non-Local-Bus controller resources (memory, PCI-Bus devices, or controller registers) and another for Local-Bus targets. Accesses by the Local-Bus master to controller resources are implemented by the DMA logic, as described in the next section. Accesses by the Local-Bus master to Local-Bus targets are implemented directly between the two devices, in a separate address space and without the assistance of the controller (except for LOC_CLK, which the controller continues to drive).

8.4.2 Local-Bus Target Transactions (Local Bus-to-Controller)

When a Local-Bus master requests a controller resource, the controller's DMA logic carries out that request. The DMA logic must be in a receptive state before the controller grants the Local-Bus master control of the Local Bus. The controller does this, regardless of whether the request is targeted to an internal controller resource or to a Local-Bus device, because the target of the cycle is not known until the Local-Bus master's bus cycle begins. The effect on non-Local-Bus DMA activity depends on whether the Local-Bus master is targeting a controller resource or a Local-Bus device:

- ❑ *Controller-Resource Target, LOC_A[4] = 1:* If the Local-Bus master is targeting a controller resource, non-Local-Bus DMA activity to/from that resource is delayed until the Local-Bus master's request has completed. However, DMA activity to/from other resources can continue in parallel with the Local-Bus master's bus request.
- ❑ *Local-Bus Target, LOC_A[4] = 0:* If the Local-Bus master is targeting a Local-Bus device, DMA activity is free to resume immediately because the DMA logic is not involved with the Local-Bus activity.

The following types of accesses by Local-Bus masters are not supported:

- ❑ *Loop-Back Requests via the Controller*. Requests to Local-Bus targets through the controller's internal logic (i.e., when LOC_A[4] = 1) are not allowed, because these requests will cause a deadlock. As described above, a Local-Bus master's request for controller resources is carried out via the DMA logic. When the DMA logic requests the Local Bus, it is held off until the Local Bus is free. But the Local Bus will not be free until the DMA completes its request. The result is deadlock. If such an access is attempted, the controller discards write data and terminates read requests by returning all 0s as data.
- ❑ *Loop-Back Requests via a PCI-Bus Device*. The same deadlock scenario described above would occur for loop-back requests via the PCI Bus. Such requests must not be attempted.
- ❑ *Non-Word Accesses:* All requests by Local-Bus masters to controller resources are assumed to be 32-bit word width, because there are no byte-enable signals in this direction. Size information in the request is ignored. The controller always performs 32-bit operations.

8.5 Local Bus vs. 64-bit PCI Bus

When PCI64# is asserted, the controller implements a 64-bit PCI Bus. In this case, the controller reconfigures the functions of several Local-Bus signals, as follows:

- ❑ *LOC_AD[31:0]:* becomes PCI_AD[63:32]
- ❑ *LOC_ALE:* becomes REQ64#, request 64-bit transfer on PCI Bus.
- ❑ *LOC_CLK:* becomes ACK64#, acknowledge 64-bit transfer on PCI Bus.
- ❑ *LOC_A[3:0]:* become C/BE#[7:4], the byte-enables for PCI_AD[63:32].
- ❑ *LOC_A[4]:* becomes PAR64, the even-parity bit for PCI_AD[63:32] and C/BE#[7:4].

Table 4 on page 27 lists all of these signal reconfigurations. If any of the seven Local-Bus DCS#[8:2] chip-selects are to be used in this configuration, they must be accessed on the memory bus by setting the MEM/LOC bit in the Physical Device Address Registers (PDAR, Section 5.4). The location of the BOOTCS is automatically moved to the memory bus when PCI64# is asserted.

8.6 Local-Bus Registers

Local-Bus masters, if enabled to arbitrate for control of Local Bus by the ARBEN field of this register, may access the controller's resources (memory, PCI-Bus devices, DMA, and the controller's internal registers) or other Local-Bus devices.

Table 27: Local-Bus Registers

Register	Symbol	Offset	R/W	Reset Value	Description
Local Bus Configuration	LCNFG	0x0100	R/W	0x0 0000 0000	Local Bus configuration
reserved	—	0x0108	R	0x0 0000 0000	—
Local Bus Chip-Select Timing 2 [a]	LCST2	0x0110	R/W	0x0 0000 0000	Local Bus cycle timing for DCS#[2] signal.
Local Bus Chip-Select Timing 3 [a]	LCST3	0x0118	R/W	0x0 0000 0000	Local Bus cycle timing for DCS#[3] signal.
Local Bus Chip-Select Timing 4 [a]	LCST4	0x0120	R/W	0x0 0000 0000	Local Bus cycle timing for DCS#[4] signal.
Local Bus Chip-Select Timing 5 [a]	LCST5	0x0128	R/W	0x0 0000 0000	Local Bus cycle timing for DCS#[5] signal.
Local Bus Chip-Select Timing 6 [a]	LCST6	0x0130	R/W	0x0 0000 0000	Local Bus cycle timing for DCS#[6] signal.
Local Bus Chip-Select Timing 7 [a]	LCST7	0x0138	R/W	0x0 0000 0000	Local Bus cycle timing for DCS#[7] signal.
Local Bus Chip-Select Timing 8 [a]	LCST8	0x0140	R/W	0x0 0000 0000	Local Bus cycle timing for DCS#[8] signal.
reserved	—	0x0148	R	0x0 0000 0000	—
Device Chip-Select Muxing and Output Enables	DCSFN	0x0150	R/W	0x0 0000 0000	Device CS source muxing and output-enables
Device Chip-Selects As I/O Bits	DCSIO	0x0158	R/W	0x0 0000 0000	Device chip-select signals as I/O signals.
reserved	—	0x0160	R	0x0 0000 0000	—
reserved	—	0x0168	R	0x0 0000 0000	—
reserved	—	0x0170	R	0x0 0000 0000	—
Local Boot Chip-Select Timing [a]	BCST	0x0178	R/W	0x0 003F 8E3F	Local-Bus cycle timing for BootCS# signal.

[a] When the controller is configured for 32-bit PCI operation (PCI64# negated), the boot memory and the seven DCS devices can be individually configured by the MEM/LOC bit in the PDAR (Section 5.4) to appear on the memory bus or the Local Bus. When the controller is configured for 64-bit PCI operation (PCI64# asserted), these devices always appear on the memory bus.

8.6.1 Local Bus Configuration Register (LCNFG)

Bit 0 ARBMODE *Local-Bus Arbitration Mode.*
1 = 68000 mode (i.e. BR, BG, BGACK).
0 = Intel mode (i.e. HOLD, HLDA).
For 68000 mode, the BR#, BG#, and BGACK# signals serve as the 68000 BR, BG, and BGACK signals, respectively. For Intel mode, the BR# and BGACK# signals serve as the Intel HOLD and HLDA signals, respectively.

Bit 1 ARBEN *Local-Bus Arbitration Enable.*
1 = enable Local-Bus masters to arbitrate for control of the Local Bus.
0 = disable Local-Bus masters from controlling the Local Bus.
Clearing this bit prevents access to controller resources. Setting the bit allows Local-Bus devices to arbitrate for control of the Local Bus, using the arbitration mode specified by the ARBMODE bit.

Bits 3:2 *reserved* Hardwired to 0.

Bits 4 FLCLCLK *Fast Clock.*
1 = LOC_CLK runs at SysClock divided by 2.
0 = LOC_CLK runs at SysClock divided by 4.
This bit resets to 0.

Bits 15:5	*reserved*	Hardwired to 0.
Bits 19:16	DMAHOG	*Minimum Number of Accesses by DMA.* The minimum number of consecutive Local-Bus cycles the DMA may perform before the DMA is forced to allow the CPU or a PCI-Bus master to take control of the Local Bus. 0x0 means 1 access, 0xF means 16 consecutive accesses. Resets to 0x0. The limit is only enforced when another resource requests the Local Bus. The Local-Bus DMAHOG, PCIHOG and CPUHOG fields are the software interface to the Programmable 3-Way Arbiter, shown in Figure 1 on page 12.
Bits 23:20	PCIHOG	*Minimum Number of Accesses by PCI.* The minimum number of consecutive PCI-interface accesses to Local-Bus resources before the PCI interface is forced to allow another controller resource to take control of the Local Bus. 0x0 means 1 access, 0xF means 16 consecutive accesses. Resets to 0x0. The limit is only enforced once another resource requests the Local Bus.
Bits 27:24	CPUHOG	*Minimum Number of Accesses by CPU.* The minimum number of consecutive CPU accesses to Local-Bus resources before the CPU is forced to allow another controller resource to take control of the Local Bus. 0x0 means 1 access, 0xF means 16 consecutive accesses. Resets to 0x0. The limit is only enforced once another resource requests the Local Bus.
Bit 63:28	*reserved*	Hardwired to 0.

8.6.2 **Local Bus Chip-Select Timing Registers (LCSTn)**

The eight identical LCSTn registers configure bus-cycle timing characteristics on the Local Bus. The seven LCST8:2 registers correspond to the DCS#[8:2] device chip-select signals, which themselves are configured in their PDARs (Section 5.4). One more register, BCST, corresponds to the BootCS# signal, which is also configured by its PDAR.

Bit 0	CSON	*Chip-Select On (Asserted).* 1 = assert DCS#[n] one clock after valid address. 0 = assert DCS#[n] with valid address. The valid address referred to is on the LOC_AD[31:0] bus. Asserting DCS#[n] with the valid address means in the clock that LOC_FR# is asserted. Be careful when using LOC_A[4:0] with CSON cleared to 0, because DCS#[n] will assert while LOC_A[4:0] drives byte-enables, one clock before LOC_A[31:0] drives the address.

Bits 2:1 CONSET *Command-On Set.*

The number of clocks, after the assertion of DCS#[n], that the LOC_RD# or LOC_WR# signal is asserted. When zero, the command (LOC_RD# or LOC_WR#) is asserted coincident with DCS#[n].

Bits 8:3 CONWID *Command-On Width (or Local-Bus Ready Timer).*

When the RDYMODE bit (bit 22) is cleared, this field specifies the duration of LOC_RD# or LOC_WR# signal assertion. The CONWID value can range from 1 to 64 LOC_CLKs. The duration of assertion is the CONWID value, plus 1. For example, a CONWID value of 000000b specifies a 1-clock duration of the read or write command. A CONWID value of 000111b specifies an 8-clock assertion, and so on.

When the RDYMODE bit (bit 22) is set, indicating the LOC_RDY# signal is being used, the CONWID field is concatenated with the SUBSCWID field to form a 12-bit Local-Bus Ready Timer for LOC_RDY#, with CONWID being the lower 6 bits. Interrupts based on this timer are enabled by the LBRTDEN bit of the Interrupt Control Register (INTCTRL, Section 5.5.2).

Bits 14:9 SUBSCWID *Subsequent Command-On Width (or Local-Bus Ready Timer).*

When the RDYMODE bit (bit 22) is cleared, this field specifies the duration of LOC_RD# or LOC_WR# signal (command) assertion for subsequent portions of a block-transfer cycle. The CONWID value can range from 1 to 64 LOC_CLKs. The duration of assertion is the CONWID value, plus one.

When the RDYMODE bit (bit 22) is set, indicating the LOC_RDY# signal is being used, the SUBSCWID field is concatenated with the CONWID field to form a 12-bit Local-Bus Ready Timer for LOC_RDY#, with SUBSCWID being the upper 6 bits. If used, this timer should be programmed to a value higher than the slowest device on the Local Bus. The timer begins counting down when a LOC_RDY#-response bus cycle begins. If a LOC_RDY# is not received before the timer reaches zero, the cycle terminates as though a LOC_RDY# were received, and an interrupt is generated, if enabled by the LBRTDEN bit of the Interrupt Control Register (INTCTRL, Section 5.5.2).

Bits 16:15 CSOFF *Chip-Select Off.*

This field specifies the number of LOC_CLKs, after LOC_RD# or LOC_WR# is negated, that DCS#[n] is

		negated. When zero, DCS#[n] is negated coincident with the read or write signal. When non-zero, DCS#[n] is negated that number of clocks after the read or write signal is negated.
Bits 18:17	COFHOLD	*Command-Frame Hold.* This field specifies the number of LOC_CLKs, after DCS#[n] is negated, that the command-frame is extended (LOC_FR# held asserted). When zero, LOC_FR# is negated coincident with the negation of DCS#[n]. When non-zero, LOC_FR# is negated that number of clocks after the negation of DCS#[n].
Bits 21:19	BUSIDLE	*Bus Idle.* This field specifies the minimum number of LOC_CLKs between the negation and re-assertion of LOC_FR# for a subsequent cycle. There is a two-clock minimum imposed by the control logic. The idle time increases if the subsequent cycle is less than a dword and one or more of the least-significant byte-enables from the master is negated (this delay is caused by logic that searches through the requestor’s byte-enables so that only necessary Local-Bus cycles are performed).
Bit 22	RDYMODE	*Ready Mode.* 1 = LOC_RDY# determines access duration. 0 = fixed timing for accesses, per bits 14:3. If RDYMODE is set, the CONWID and SUBSCWID fields (bits 14:3) become a Local-Bus Ready Timer for LOC_RDY#. This time-out timer works for both reads and writes on the Local Bus.
Bit 23	RDYSYN	*LOC_RDY# Synchronize.* 1 = synchronize LOC_RDY# to SysClock. 0 = do not synchronize LOC_RDY# to SysClock. If this bit is set, the LOC_RDY# signal is assumed to be asynchronous to SysClock, and the controller will synchronize it to SysClock. This imposes a 2-clock delay at the end of the access.
Bits 25:24	CONOFF	*Command Off.* This field specifies the number of LOC_CLKs, after LOC_RDY# is asserted, that the LOC_RD# or LOC_WR# signal (command) is negated. When zero, the command is negated coincident with the assertion of LOC_RDY# (or two clocks later if LOC_RDY# requires synchronization). When non-zero, the command is negated that number of clocks after LOC_RDY# is asserted.

Bit 26 CS_POL *Chip-Select Polarity.*
1 = DCS#[n] is active-High.
0 = DCS#[n] is active-Low.

Bit 27 CON_POL *Command Polarity.*
1 = LOC_RD# and LOC_WR# are active-High.
0 = LOC_RD# and LOC_WR# are active-Low.

Bits 63:28 *reserved* Hardwired to 0.

8.6.3 Device Chip-Select Function Register (DCSFN)

This register specifies the functionality of the DCS#[8:2] signals. These signals can be used as device chip-selects, whose operation is controlled by the corresponding PDAR (Section 5.4) and Local-Bus Chip Select Timing Register (LCSTn, Section 8.6.2). Alternatively, they can be used for general-purpose I/O bits, additional UART modem control functions, or DMA hardware handshaking.

The DCSFN register and the DCS[8:2] PDARs must be programmed before accessing devices selected by the DCS#[8:2] signals.

Bits 2:0 DCSFN2 *DCS#[2] Signal Function*.

Binary Value	Signal Function
b000	General-purpose input whose value can be read in the DCSL2IN field of the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4. Reset value.
b001	The controller’s memory interface or Local-Bus interface drives the signal, depending on the MEM/LOC bit in Physical Device Address Register DCS2, Section 5.4.
b011	General-purpose output whose value is programmed by bit 8 of the DCSLOUT field in the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4.
b101	The UART_RTS# output signal is enabled on the DCS#[2] pin.
All other values	*reserved*

Bit 3 *reserved* Hardwired to 0.

Bits 6:4 DCSFN3 *DCS#[3] Signal Function*.

Binary Value	Signal Function
b000	General-purpose input whose value can be read in the DCSL3IN field of the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4. Reset value.
b001	The controller’s memory interface or Local-Bus interface drives the signal, depending on the MEM/LOC bit in Physical Device Address Register DCS3, Section 5.4.
b011	General-purpose output whose value is programmed by bit 9 of the DCSLOUT field in the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4.
b110	The UART_CTS# output signal is enabled on the DCS#[3] pin.
All other values	*reserved*

Bit 7 *reserved* Hardwired to 0.

Bits 10:8 DCSFN4 *DCS#[4] Signal Function*.

Binary Value	Signal Function
b000	General-purpose input whose value can be read in the DCSL4IN field of the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4. Reset value.
b001	The controller’s memory interface or Local-Bus interface drives the signal, depending on the MEM/LOC bit in Physical Device Address Register DCS4, Section 5.4.
b011	General-purpose output whose value is programmed by bit 10 of the DCSLOUT field in the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4.
b110	The UART_DCD# output signal is enabled on the DCS#[4] pin.
All other values	*reserved*

Bit 11 *reserved* Hardwired to 0.

Bits 14:12 DCSFN5 *DCS#[5] Signal Function*.

Binary Value	Signal Function
b000	General-purpose input whose value can be read in the DCSL5IN field of the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4. Reset value.
b001	The controller’s memory interface or Local-Bus interface drives the signal, depending on the MEM/LOC bit in Physical Device Address Register DCS5, Section 5.4.

Binary Value	Signal Function
b011	General-purpose output whose value is programmed by bit 11 of the DCSLOUT field in the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4.
b110	The UART_XIN output signal is enabled on the DCS#[5] pin.
All other values	*reserved*

Bit 15 *reserved* Hardwired to 0.

Bits 18:16 DCSFN6 *DCS#[6] Signal Function*.

Binary Value	Signal Function
b000	General-purpose input whose value can be read in the DCSL6IN field of the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4. Reset value.
b001	The controller’s memory interface or Local-Bus interface drives the signal, depending on the MEM/LOC bit in Physical Device Address Register DCS6, Section 5.4.
b011	General-purpose output whose value is programmed by bit 12 of the DCSLOUT field in the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4.
b101	The DMA hardware handshaking DMA_ACK# output signal is enabled on the DCS#[6] pin. See Section 9.4.
All other values	*reserved*

Bit 19 *reserved* Hardwired to 0.

Bits 22:20 DCSFN7 *DCS#[7] Signal Function*.

Binary Value	Signal Function
b000	General-purpose input whose value can be read in the DCSL7IN field of the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4. Reset value.
b001	The controller’s memory interface or Local-Bus interface drives the signal, depending on the MEM/LOC bit in Physical Device Address Register DCS7, Section 5.4.
b011	General-purpose output whose value is programmed by bit 13 of the DCSLOUT field in the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4.
b110	The DMA hardware handshaking DMA_REQ# input signal is enabled on the DCS#[7] pin. See Section 9.4.
All other values	*reserved*

Bit 23 *reserved* Hardwired to 0.

Bits 26:24 DCSFN8 *DCS#[8] Signal Function.*

Binary Value	Signal Function
b000	General-purpose input whose value can be read in the DCSL8IN field of the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4. Reset value.
b001	The controller's memory interface or Local-Bus interface drives the signal, depending on the MEM/LOC bit in Physical Device Address Register DCS8, Section 5.4.
b011	General-purpose output whose value is programmed by bit 14 of the DCSLOUT field in the Device Chip-Selects as I/O Bits Register (DCSIO), Section 8.6.4.
b110	The DMA hardware handshaking DMA_EOT# input signal is enabled on the DCS#[8] pin. See Section 9.4.
All other values	*reserved*

Bits 63:27 *reserved* Hardwired to 0.

8.6.4 Device Chip-Selects as I/O Bits Register (DCSIO)

The DCSIO register is used to read input and write output when any of the DCS#[8:2] signals are specified to be used for general-purpose I/O in the Device Chip-Select Function Register (DCSFN, Section 8.6.3). Bits 6:0 of the DCSIO register represent inputs on the DCS[8:2] signals. Bits 14:8 represent outputs on the DCS[8:2] signals. The input bits are synchronized internally by the controller to SysClock.

Bit 0 DCSL2IN *DCS#[2] Input Synchronized.*
The input value on DCS#[2], synchronized internally to SysClock.

Bit 1 DCSL3IN *DCS#[3] Input Synchronized.*
The input value on DCS#[3], synchronized internally to SysClock.

Bit 2 DCSL4IN *DCS#[4] Input Synchronized.*
The input value on DCS#[4], synchronized internally to SysClock.

Bit 3 DCSL5IN *DCS#[5] Input Synchronized.*
The input value on DCS#[5], synchronized internally to SysClock.

Bit 4 DCSL6IN *DCS#[6] Input Synchronized.*
The input value on DCS#[6], synchronized internally to SysClock.

Bit 5 DCSL7IN *DCS#[7] Input Synchronized.*
The input value on DCS#[7], synchronized internally to SysClock.

Bit 6	DCSL8IN	*DCS#[8] Input Synchronized.* The input value on DCS#[8], synchronized internally to SysClock.
Bit 7	*reserved*	Hardwired to 0.
Bits 14:8	DCSLOUT	*DCS#[8:2] Value.* The value to drive on DCS#[8:2] if these signals are enabled by the DCSOE[8:2] bits in the DCSFN register (Section 8.6.3).
Bits 63:15	*reserved*	Hardwired to 0.

8.6.5 Local Boot Chip-Select Timing Register (BCST)

This register has the same format at the LCST8:2 registers (Section 8.6.2). The reset value for the BCST register is 0x0 003F 8E3F. So, at reset the timing parameters for the Boot Chip-Select (BOOTCS) Physical Device Address Register (Section 5.4) are:

		Reset Value
Bit 0	CSON	1
Bits 2:1	CONSET	3
Bits 8:3	CONWID	7
Bits 14:9	SUBSCWID	7
Bits 16:15	CSOFF	3
Bits 18:17	COFHOLD	3
Bits 21:19	BUSIDLE	7
Bit 22	RDYMODE	0
Bit 23	RDYSYN	0
Bits 25:24	CONOFF	0
Bit 26	CS_POL	0
Bit 27	CON_POL	0
Bits 63:28	*reserved*	0

This configures BOOTCS for the slowest possible boot ROM. After boot, you may configure this register to allow faster access, depending on the timing requirements for your boot ROM.

9.0 DMA Controller and Registers

The controller supports DMA transfers, from any physical address to any physical address, on two chainable channels. Thus, DMA transfers can occur:

- ❑ From:
  - Memory,
  - PCI-Bus device,
  - Local-Bus device, or
  - Controller’s internal registers
- ❑ To:
  - Memory,
  - PCI-Bus device,
  - Local-Bus device, or
  - Controller’s internal registers

The DMA logic includes a 32-entry x 8-byte (256-byte) DMA FIFO that is used to buffer transfers. The controller is capable of performing unaligned read and write transfers from and to main memory at a maximum rate of 640Mb/s. The controller is also capable of transferring from and to the PCI Bus at the maximum PCI transfer rate of 533 MB/sec (64-bit, 66 MHz), 266 MB/sec (64-bit, 33 MHz or 32-bit, 66 MHz), or 133 MB/sec (32-bit, 33 MHz).

9.1 DMA Configuration and Monitoring

Software configures and monitors the DMA logic using the following registers:

- ❑ Interrupt Control Register (INTCTRL), Section 5.5.2 on page 52.
- ❑ Interrupt Status Register 0 (INTSTAT0), Section 5.5.3 on page 55.
- ❑ Interrupt Clear Register (INTCLR), Section 5.5.5 on page 56.
- ❑ DMA Registers, Section 9.5 on page 133.
- ❑ Device Chip-Select Function Register (DCSFN), Section 8.6.3 on page 125.

9.2 DMA Transfer Mechanism

The DMA transfer mechanism is configured by software and operates autonomously thereafter. If both DMA channels are configured for transfers, the second channel will automatically begin transferring when the first channel completes. This is called *chaining*.

9.2.1 Configuration and Enabling

The controller contains two sets of DMA registers, for Channel 0 and Channel 1 (Section 9.5). Each register set controls a separate DMA transfer. One set of registers may be written or read while the other set is controlling a transfer. Active registers can be read, but writing of the active registers is limited to the writing of only the DMA Reset (DRST) and Suspend DMA (SU) bits in the DMA Control Register (Section 9.5.1). Transfers on the two channels can be chained (linked), so that the completion of a transfer on one channel causes the second channel to begin transferring.

The controller’s DMA registers can be configured by any master attached to any of controller interfaces. Typically this is the CPU, but it may be any device on the PCI or Local Bus. To begin a DMA transfer, software specifies the source address, destination address, length of transfer, end-of-transfer interrupt, and transfer enable (the GO bit) in the DMA Control Register for that channel.

9.2.2 Operation

When the transfer has been enabled, the controller begins by acquiring access to the resource that is the source of the data transfer. When access to the source is granted, the controller begins reading data at the highest rate supported by the source and placing the data in its 32 x 8-byte DMA FIFO. When the FIFO reaches its high-water mark, the controller requests access to the destination of the transfer. When access to the destination is granted, the controller begins writing the data at the highest rate possible supported by the destination.

If, during a transfer, the DMA FIFO becomes full, the controller releases control of the data source until the FIFO is emptied to its low-water mark. The controller then reacquires the data source and continues filling the FIFO. If the FIFO becomes empty, the controller releases the data destination until the FIFO has been filled to its high-water mark. The controller then reacquires the data destination and continues emptying the FIFO.

When the correct number of bytes has been read from the source, the controller stops filling the FIFO but continues emptying the FIFO until the last transfer completes. Then the controller issues a DMA-complete interrupt to the CPU, if enabled as described in the next section, below.

9.2.3 Completion

When a transfer finishes, the controller generates an interrupt to the CPU, if the interrupt is enabled by the DMAEN bit in the Interrupt Control Register (INTCTRL, Section 5.5.2) and the IE field in the DMA Control Register for that DMA channel (DMACTRL, Section 9.5.1). The controller checks the status of the other set of DMA control registers to determine if another transfer is configured (chained); if so, the next DMA transfer begins automatically.

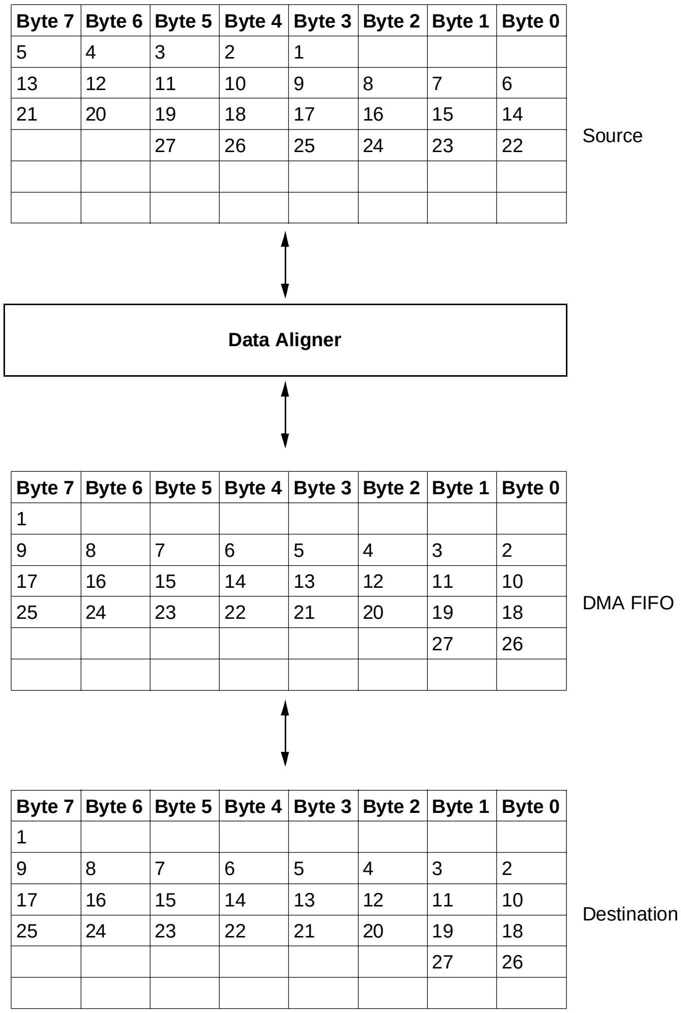
If any error occurs, the controller stops the current DMA transfer, sets one of the Stopped On An Error (bits 34:32) in the DMA Control Register for that DMA channel (DMACTRL, Section 9.5.1), and generates an interrupt, if enabled.

9.3 Data Aligner

The controller automatically handles unaligned DMA transfers. The aligner supports block reads and writes even when both the source and destination addresses are not aligned on dword boundaries.

The aligner packs data into the DMA FIFO in the alignment required by the destination address. Figure 18 shows the operation of the aligner for a DMA transaction starting from source address 0003 to starting destination address 0007.

Figure 18: Unaligned DMA Transfer Example



Source

Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
5	4	3	2	1			
13	12	11	10	9	8	7	6
21	20	19	18	17	16	15	14
		27	26	25	24	23	22

Data Aligner

DMA FIFO

Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
1							
9	8	7	6	5	4	3	2
17	16	15	14	13	12	11	10
25	24	23	22	21	20	19	18
						27	26

Destination

Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
1							
9	8	7	6	5	4	3	2
17	16	15	14	13	12	11	10
25	24	23	22	21	20	19	18
						27	26

9.4

DMA Hardware Handshaking

DMA transfers can be initiated either entirely in software, or in software accompanied by hardware handshaking. When the Hardware Handshake Enable (HHSEN) bit is set to 1 in the DMA Control Register (DMACTRLn, Section 9.5.1), and the DCSFNn and DCSOEn fields in the Device Chip-Select Function Register (DCSFN, Section 8.6.3) contains the appropriate values, the controller implements hardware handshaking by reconfiguring the functions of the DCS#[8:6] signals, as follows:

- ❑ DCS#[6] becomes DMA_ACK# (output).
- ❑ DCS#[7] becomes DMA_REQ# (input).
- ❑ DCS#[8] becomes DMA_EOT# (input).

9.4.1 External DMA Requests

An external device can use the DMA_REQ# input by itself to request a transfer. If the device uses the DMA_ACK# output, the device must also use the DMA_REQ# input. Using the DMA_REQ# input by itself, without the DMA_ACK# output, may cause difficulties in determining when it is safe to negate DMA_REQ#. The Hardware Handshake Destination (HHSDEST) bit in the DMA Control Register specifies whether the source or destination does the hardware handshake with the controller.

An external device requests a DMA transfer by asserting DMA_REQ#. The controller responds by asserting DMA_ACK# and begins reading data from the source and writing it to the destination. When the external device negates DMA_REQ#, the controller negates DMA_ACK#. For each of these assertion/negation cycles, one block of data (32 bytes) is read until the DMA transfer count goes to zero.

Blocks are aligned on 32-byte boundaries. The first and last transfers may be less than 32 bytes, because they may only transfer part of a block. If the requesting device can source or sink another block, the device can re-assert DMA_REQ# one clock after negating it.

9.4.2 End Of Transfer

The external device can use the DMA_EOT# input to abort a DMA transfer, but only if the DMA source is doing the handshaking (HHSDEST=0 in DMACTRLn). DMA errors may be reported by assertion of the DMA_EOT# input, if this function is enabled by the HHSEOT bit in the DMA Control Register. When such an error is reported, the controller aborts the associated DMA transfer, treating it as if the DMA transfer count had gone to zero. This is the DMA_EOT# input's only function.

9.5 DMA Registers

The controller contains two sets of DMA registers, for Channel 0 and Channel 1, each of which controls a separate DMA transfer. One set of registers may be written or read while the other set is active (controlling a transfer). Active registers can be read, but writing of the active registers is limited to the DMA Reset (DRST) and Suspend DMA (SU) bits in the DMA Control Register (Section 9.5).

Table 28: DMA Control Registers

Register	Symbol	Offset	R/W	Reset Value	Description
DMA Control 0	DMACTRL0	0x0180	R/W	0x0000 0000 0000 0000	DMA control set 0.
DMA Source Address 0	DMASRCA0	0x0188	R/W	0x0000 0000 0000 0000	DMA source address set 0.
DMA Destination Address 0	DMADESA0	0x0190	R/W	0x0000 0000 0000 0000	DMA destination address set 0.
DMA Control 1	DMACTRL1	0x0198	R/W	0x0000 0000 0000 0000	DMA control set 1.
DMA Source Address 1	DMASRCA1	0x01A0	R/W	0x0000 0000 0000 0000	DMA source address set 1.
DMA Destination Address 1	DMADESA1	0x01A8	R/W	0x0000 0000 0000 0000	DMA destination address set 1.
reserved	—	0x01B0	R	0x0000 0000 0000 0000	—
reserved	—	0x01B8	R	0x0000 0000 0000 0000	—

9.5.1 DMA Control Registers 0 and 1 (DMACTRLn)

Bits 19:0	BLKSIZE	*Block Size.* The number of bytes (up to 1 MB) to be transferred. 0 = 1 MB.
Bits 21:20	*reserved*	Hardwired to 0.

Bit 22	HHSDEST	*Hardware Handshake By Source or Destination.* 1 = destination handshakes with controller. 0 = source handshakes with controller. When the DCSFN[8:6] fields in the Device Chip-Select Muxing and Output-Enables Register (DCSFN, Section 8.6.3) contain the value 0x2, the DCS#[8:6] signals become the DMA_EOT#, DMA_REQ#, and DMA_ACK# signals, respectively. These signals are used for DMA handshaking with the controller. The HHSDEST bit specifies whether the source or destination of the DMA transfer does the hardware handshake. The HHSDEST bit is valid only if handshaking is enabled by the HHSEN bit.
Bit 23	HHSEN	*Hardware Handshake Enable.* 1 = enable. 0 = disable. This bit is valid only when at least one of the DCSFN[8:6] fields in the Device Chip-Select Muxing and Output-Enables Register (DCSFN, Section 8.6.3) contains the value 0x2 and the corresponding DCSOEn bit is set in the DCSFN register.
Bit 24	DRST	*DMA Reset.* 1 = reset. 0 = no reset. When this bit is set to 1, any DMA in process is terminated and reset after completion of the bus cycle in process. This bit automatically clears to 0 after the DMA channel has been successfully reset. This bit takes precedence over all other bits in the DMA Control Registers; values written to other bits in the register are disregarded when the DRST bit is set.
Bit 25	SRCINC	*Source-Address Incrementing.* 1 = increment source address. 0 = do not increment source address. Setting this bit causes the controller to increment the DMA source address for each read.
Bit 26	DESINC	*Destination-Address Incrementing.* 1 = increment destination address. 0 = do not increment destination address. Setting this bit causes the controller to increment the DMA destination address for each write.
Bit 27	SU	*Suspend DMA.* 1 = suspend current transfer. 0 = restart suspended transfer. This bit suspends the current DMA transfer after completion of current cycle. All register values are preserved. The suspended transfer may be restarted

		by clearing the SU bit. This bit may be set and cleared without consideration of the other bits in this register, except the DMA Reset (DRST) bit; changing bits other than SU and DRST have no effect after a DMA transfer has started.
Bit 28	GO	*Start Transfer.* 1 = start DMA transfer. 0 = (no effect). When set to 1, this bit causes DMA to begin with the parameters specified in the DMA Control Registers. The bit automatically resets after the transfer completes. Clearing the bit in software has no effect; the DMA transfer will continue.
Bit 29	IVLD	*Interrupt Valid.* 1 = this DMA channel generated an interrupt on completion of it’s last transfer. 0 = clear interrupt. This bit automatically resets when the GO bit is set for a new transfer.
Bit 30	IE	*Interrupt Enable.* 1 = enables interrupt on completion of the transfer specified by this channel. 0 = disable such interrupts. The DMAEN field in the Interrupt Control Register (INTCTRL, Section 5.5.2) is a global enable for the IE fields in the two DMA Control Registers. If the transfer in either DMA channel completes, and DMAEN is set, the controller generates an interrupt to the CPU. The IVLD bit, above, specifies which channel caused the interrupt.
Bit 31	BZ	*Busy.* (read only) 1 = a DMA controlled by this register is currently in process. 0 = DMA not in process. This bit may be polled.
Bit 32	MRDERR	*Memory Read Error.* 1 = transfer stopped on a memory read error. 0 = no such error.
Bit 33	PRDERR	*PCI Read Error.* 1 = transfer stopped on a PCI-Bus read error. 0 = no such error.
Bit 34	UDRDERR	*Undecodable Read Error.* 1 = transfer stopped on an undecodable read error. 0 = no such error.

Bit 35	HHSEOT	*Hardware Handshake Error.* 1 = DMA stopped on DMA_EOT assertion. 0 = no such error.
Bits 63:36	*reserved*	Hardwired to 0.

9.5.2

DMA Source Address Register 0 and 1 (DMASRCAn)

Bits 35:0	DMASRCA	*DMA Source Starting Address.* The starting source (read) address for this transfer. This register remains static throughout the DMA transfer, although setting the SRCINC bit of the DMA Control Register (Section 9.5.1) causes source (read) addresses to be incremented.
Bits 63:36	*reserved*	Hardwired to 0.

9.5.3

DMA Destination Address Register 0 and 1 (DMADESAn)

Bits 35:0	DMADESA	*DMA Destination Starting Address.* The starting destination (write) address for this transfer. This register remains static throughout the DMA transfer, although setting the DESINC bit of the DMA Control Register (Section 9.5.1) causes destination (write) addresses to be incremented.
Bits 63:36	*reserved*	Hardwired to 0.

10.0 Serial Port and Registers

The controller implements one serial port with the NEC NY16550L UART Mega Function. This UART is functionally identical to the National Semiconductor NS16550D. Details of its function can be found in the *CB-C8VX/VM ASIC Family 0.5 micron Standard Cell User's Manual, Mega Function NY16550L UART, Preliminary, 4 October 1996*.

10.1 Serial-Port Configuration and Monitoring

Software configures and monitors the serial-port logic using the following registers:

- ❑ Interrupt Control Register (INTCTRL), Section 5.5.2 on page 52.
- ❑ Interrupt Status Register 0 (INTSTAT0), Section 5.5.3 on page 55.
- ❑ Interrupt Clear Register (INTCLR), Section 5.5.5 on page 56.
- ❑ Serial-Port Registers, Section 10.4 on page 139.
- ❑ Device Chip-Select Function Register (DCSFN), Section 8.6.3 on page 125.

At reset, the UART_DTR# and UART_TxDRDY# signals define the controller's ID in a multi-controller configuration, as described in Section 12.0. However, this configuration does not affect the operation of the UART itself.

10.2 Additional UART Signals

The controller pinouts always carry the UART_DSR#, UART_DTR#, UART_RxDRDY#, UART_TxDRDY# signals. After reset, however, the DCS#[5:2] signals can be reconfigured to provide the following additional UART modem-control signals:

- ❑ DCS#[2] becomes UART_RTS# (output).
- ❑ DCS#[3] becomes UART_CTS# (input).
- ❑ DCS#[4] becomes UART_DCD# (output).
- ❑ DCS#[5] becomes UART_XIN (input).

These additional signals are implemented when software writes the appropriate values to the DCSFNn and DCSOEn fields in the Device Chip-Select Function Register (DCSFN, Section 8.6.3).

10.3

UART Clocking

The UART can be clocked from either an external input or by an internally-generated clock. The internal clock has a frequency of SysClock divided by 12. To achieve a desired baud rate, the UART Divisor Latch (see Section 10.4.4 and Section 10.4.5) must be properly programmed. The relationship between UART clock frequency, baud rate, and divisor value is:

baud_rate = UART_clock_frequency/(divisor_value * 16)

where for internally-generated clock:

UART_clock_frequency = SYS_CLK_freq / 12

Table 29 gives divisor values for several different input clock frequencies. The actual baud rate may vary significantly from the desired baud rate.

Table 29: UART Clock-Rate Divisor Values

	Internal Clock Source		**External Clock Source**					
	UART_XIN = 1.8432MHz		**SysClock = 99.5328MHz**		**SysClock = 88.4736MHz**		**SysClock = 73.728MHz**	
Baud Rate	**Divisor**	**Percent Error**	**Divisor**	**Percent Error**	**Divisor**	**Percent Error**	**Divisor**	**Percent Error**
50	2304		10368		9216		7680	
75	1536		6912		6144		5120	
110	1047	0.026%	4713	0.006%	4189	0.002%	3491	0.003%
134.5	857	0.058%	3854	0.007%	3426	0.001%	2855	0.001%
150	768		3456		3072		2560	
300	384		1728		1536		1280	
600	192		864		768		640	
1200	96		432		384		320	
1800	64		288		256		213	0.156%
2000	58	0.690%	259	0.077%	230	0.174%	192	
2400	48		216		192		160	
3600	32		144		128		107	0.312%
4800	24		108		96		80	
7200	16		72		64		53	0.629%
9600	12		54		48		40	
19200	6		27		24		20	
38400	3		14	3.571%	12		10	
57600	2		9		8		7	4.762%

10.4

Serial-Port Registers

Table 30: Serial-Port Register Summary

Register	Symbol	Offset	R/W	Reset Value	Description
UART Receiver Data Buffer	UARTRBR	0x0300	R	0x0000 0000 0000 00XX	UART receiver data DLAB [a] = 0
UART Transmitter Data Holding	UARTTHR	0x0300	W	0x0000 0000 0000 00XX	UART transmit data DLAB [a] = 0
UART Interrupt Enable	UARTIER	0x0308	R/W	0x0000 0000 0000 0000	UART interrupt enable DLAB [a] = 0
UART Divisor Latch LSB	UARTDLL	0x0300	R/W	0x0000 0000 0000 00XX	UART divisor latch LSB DLAB [a] = 1
UART Divisor Latch MSB	UARTDLM	0x0308	R/W	0x0000 0000 0000 00XX	UART divisor latch MSB DLAB [a] = 1
UART Interrupt ID	UARTIIR	0x0310	R	0x0000 0000 0000 0001	UART interrupt ID
UART FIFO Control	UARTFCR	0x0310	W	0x0000 0000 0000 0000	UART FIFO control
UART Line Control	UARTLCR	0x0318	R/W	0x0000 0000 0000 0000	UART line control
UART Modem Control	UARTMCR	0x0320	R/W	0x0000 0000 0000 0000	UART modem control
UART Line Status	UARTLSR	0x0328	R/W	0x0000 0000 0000 0060	UART line status
UART Modem Status	UARTMSR	0x0330	R/W	0x0000 0000 0000 0000	UART modem status
UART Scratch	UARTSCR	0x0338	R/W	0x0000 0000 0000 00XX	UART scratch

[a] Divisor Latch Access Bit (DLAB) in the UART Line Control Register (Section 10.4.8)

10.4.1

UART Receiver Data Buffer Register (UARTRBR)

This register holds receive data. It is only accessed when the Divisor Latch Access Bit (DLAB) is cleared to 0 in the UART Line Control Register (UARTLCR), Section 10.4.8.

Bits 7:0	UDATA	*UART Receive Data*. (read-only)
Bits 63:8	*reserved*	Hardwired to 0.

10.4.2

UART Transmitter Data Holding Register (UARTTHR)

This register holds transmit data. It is only accessed when the Divisor Latch Access Bit (DLAB) is cleared to 0 in the UART Line Control Register (UARTLCR), Section 10.4.8.

Bits 7:0	UDATA	*UART Transmit Data*. (write-only)
Bits 63:8	*reserved*	Hardwired to 0.

10.4.3

UART Interrupt Enable Register (UARTIER)

This register is used to enable UART interrupts. It is only accessed when the Divisor Latch Access Bit (DLAB) is set to 1 in the UART Line Control Register (UARTLCR), Section 10.4.8. The UARTEN field in the Interrupt Control Register (INTCTRL, Section 5.5.2) is a global enable for interrupt sources enabled by this register.

Bit 0	ERBFI	*Enable Receive-Buffer-Full Interrupt.* 1 = enable receive-data-available interrupt. 0 = disable such interrupt. The receive-buffer-full state is reported in the UART Line Status Register (UARTLSR, Section 10.4.10).

Bit 1	ETBEI	*Enable Transmitter-Buffer-Empty Interrupt*. 1 = enable transmit-buffer-empty interrupt. 0 = disable such interrupt. The transmit-buffer-empty state is reported in the UART Line Status Register (UARTLSR, Section 10.4.10).
Bit 2	ELSI	*Enable Line-Status Interrupts.* 1 = enable line-status-error interrupt. 0 = disable such interrupts. Line status errors are reported in the UART Line Status Register (UARTLSR, Section 10.4.10).
Bit 3	EDSSI	*Enable Modem-Status Interrupts.* 1 = enable modem-status-change interrupt. 0 = disable such interrupts. Modem status changes are reported in bits 3:0 of the UART Modem Status Register (UARTMSR, Section 10.4.11).
Bits 63:4	*reserved*	Hardwired to 0.

10.4.4 UART Divisor Latch LSB Register (UARTDLL)

This register is only accessed when the Divisor Latch Access Bit (DLAB) is set to 1 in the UART Line Control Register (UARTLCR), Section 10.4.8.

Bits 7:0	DIVLSB	*UART Divisor Latch Least-Significant Byte (LSB).* See the 16550 data sheet for details on the relation between divisor values and baud rate.
Bits 63:8	*reserved*	Hardwired to 0.

10.4.5 UART Divisor Latch MSB Register (UARTDLM)

This register is only accessed when the Divisor Latch Access Bit (DLAB) is set to 1 in the UART Line Control Register (UARTLCR), Section 10.4.8.

Bits 7:0	DIVMSB	*UART Divisor Latch Most-Significant Byte (MSB).* See the 16550 data sheet for details on the relation between divisor values and baud rate.
Bits 63:8	*reserved*	Hardwired to 0.

10.4.6 UART Interrupt ID Register (UARTIIR)

Bit 0	INTPENDL	*UART Interrupt Pending*. (read-only) 1 = no interrupt pending. 0 = interrupt pending.

Bits 3:1 UIID *UART Interrupt ID*. (read-only)

Interrupt ID#	Priority	Source of Interrupt
0x3	Highest	*Receiver Line Status:* Overrun Error, Parity, Framing Error, or Break Interrupt. The interrupt is cleared when the UART Line Status Register (UARTLSR) is read.
0x2	Second	*Received Data Available:* Receiver Data Available or Trigger Level Reached. The interrupt is cleared when the UART Receiver Data Buffer Register (UARTRBR) is read.
0x6	Second	*Character Time-Out Indication:* No change in receiver FIFO during the last four character times and FIFO is not empty. The interrupt is cleared when the UART Receiver Data Buffer Register (UARTRBR) is read.
0x1	Third	*Transmitter Holding Register Empty:* The interrupt is cleared when the UART Transmitter Data Holding Register (UARTTHR) is written or this UART Interrupt ID Register (UARTIIR) is read.
0x0	Fourth	*Modem Status:* CTS#, DSR#, or DCD#. The interrupt is cleared when the UART Modem Status Register (UARTMSR) is read.

Bits 5:4 *reserved* Hardwired to 0.

Bits 7:6 UFIFOEN *UART FIFO Enabled*. (read-only)
Both of these bits are set to 1 when the transmit/ receive FIFO is enabled in the UFIFOEN0 bit is set in the UART FIFO Control Register (UARTFCR, Section 10.4.7).

Bits 63:8 *reserved* Hardwired to 0.

10.4.7 UART FIFO Control Register (UARTFCR)

Bit 0 UFIFOEN0 *UART FIFO Enable*. (write-only)
1 = enable receive and transmit FIFOs.
0 = disable and clear receive and transmit FIFOs.

Bit 1 URFRST *UART Receiver FIFO Reset*. (write-only)
1 = clear receive FIFO and reset counter.
0 = no clear.

Bit 2 UTFRST *UART Transmitter FIFO Reset*. (write-only)
1 = clear transmit FIFO and reset counter.
0 = no clear.

Bits 5:3 *reserved* Hardwired to 0.

Bits 7:6 URTR *UART Receive FIFO Trigger Level.*

Receive Trigger Level	Number of Bytes in Receiver FIFO
0x0	01
0x1	04
0x2	08
0x3	14

When the trigger level is reached, a Receive-Buffer-Full interrupt is generated, if enabled by the ERBFI bit in the UART Interrupt Enable Register (UARTIER, Section 10.4.3).

Bits 63:8 *reserved* Hardwired to 0.

10.4.8

UART Line Control Register (UARTLCR)

Bits 1:0 WLS *Word Length Select.*
11 = 8 bits.
10 = 7 bits.
01 = 6 bits.
00 = 5 bits.

Bit 2 STB *Stop Bits.*
1 = 2 bits (except 1.5 stop bits for 5-bit words).
0 = 1 bit.

Bit 3 PEN *Parity Enable.*
1 = generate parity on writes, check it on reads.
0 = no parity generation or checking.
For the UART, even or odd parity can be generated or checked, as specified in Bit 4 (EPS). This is unlike parity on the CPU, memory and PCI Bus interfaces, which is always *even* parity.

Bit 4 EPS *Even-Parity Select.*
1 = even parity.
0 = odd parity.

Bit 5 USP *Stick Parity.*
1 = force generated and checked parity to EPS.
0 = normal parity generation and checking.
This bit is only valid when parity is enabled (PEN bit set).

Bit 6 USB *Set Break.*
1 = force UART_TxDRDY# signal output Low (0).
0 = normal operation of UART_TxDRDY# signal output.

Bit 7 DLAB *Divisor Latch Access Bit.*
1 = access baud-rate divisor at offset 0x0300:308.
0 = access TxD/RxD and IE at offset 0x0300:308
When this bit is set, the UART accesses the UART

Divisor Latch LSB Register (UARTDLL, Section 10.4.4) at offset 0x0300, and the UART Divisor Latch MSB Register (UARTDLM, Section 10.4.5) at offset 0x308. When the bit is cleared, the UART accesses the UART Receiver Data Buffer Register (UARTRBR, Section 10.4.1) on reads at offset 0x0300, the UART Transmitter Data Holding Register (UARTTHR, Section 10.4.2) on writes at offset 0x0300, and the UART Interrupt Enable Register (UARTIER, Section 10.4.3) on any access at offset 0x0308.

Bits 63:8	*reserved*	Hardwired to 0.

10.4.9

UART Modem Control Register (UARTMCR)

This register controls the state of external UART_DTR# and UART_RTS# modem-control signals and of the loop-back test.

Bit 0	DTR	*Data Terminal Ready.* 1 = negate UART_DTR# signal. 0 = assert UART_DTR# signal.
Bit 1	RTS	*Request To Send.* 1 = negate UART_RTS# signal. 0 = assert UART_RTS# signal. This bit has an effect only if the DCS#[2] pin has been programmed, after reset, to carry the UART_RTS# signal. See Section 10.2.
Bit 2	OUT1	*Out 1.* 1 = OUT1# state active. 0 = OUT1# state inactive (reset value). This is a user-defined bit that has no associated external signal. Software can write to the bit, but this has no effect.
Bit 3	OUT2	*Out 2.* 1 = OUT2# state active. 0 = OUT2# state inactive (reset value). This is a user-defined bit that has no associated external signal. Software can write to the bit, but this has no effect.
Bit 4	LOOP	*Loop-Back Test*. 1 = loop-back. 0 = normal operation. This is an NEC internal test function.
Bits 63:5	*reserved*	Hardwired to 0.

10.4.10 UART Line Status Register (UARTLSR)

This register reports the current state of the transmitter and receiver logic.

Bit 0	DR	*Receive-Data Ready.* 1 = receive data buffer full. 0 = receive data buffer not full. Receive data is stored in the UART Receiver Data Buffer Register (UARTRBR, Section 10.4.1).
Bit 1	OE	*Receive-Data Overrun Error.* 1 = overrun error on receive data. 0 = no such error.
Bit 2	PE	*Receive-Data Parity Error.* 1 = parity error on receive data. 0 = no such error.
Bit 3	FE	*Receive-Data Framing Error.* 1 = framing error on receive data. 0 = no such error.
Bit 4	BI	Break Interrupt. 1 = break received on UART_RxDRDY# signal. 0 = no break.
Bit 5	THRE	*Transmitter Holding Register Empty.* 1 = transmitter holding register empty. 0 = transmitter holding register not empty. Transmit data is stored in the UART Transmitter Data Holding Register (UARTTHR, Section 10.4.2).
Bit 6	TEMT	Transmitter Empty. 1 = transmitter holding and shift registers empty. 0 = transmitter holding or shift register not empty.
Bit 7	RFERR	*Receiver FIFO Error.* 1 = parity, framing, or break error in receiver buffer. 0 = no such error.
Bits 63:8	*reserved*	Hardwired to 0.

10.4.11 UART Modem Status Register (UARTMSR)

This register reports the current state of and changes in various control signals.

Bit 0	DCTS	*Delta Clear To Send.* 1 = UART_CTS# state changed since this register was last read. 0 = no such change.
Bit 1	DDSR	*Delta Data Set Ready.* 1 = UART_DSR# input signal changed since this register was last read. 0 = no such change.

Bit 2	TERI	*Trailing Edge Ring Indicator.* 1 = RI# state changed since this register last read. 0 = no such change. RI# is not implemented as an external signal, so this bit is never set by the controller.
Bit 3	DDCD	*Delta Data Carrier Detect.* 1 = UART_DCD# state changed since this register was last read. 0 = no such change.
Bit 4	CTS	*Clear To Send.* 1 =UART_CTS# state active. 0 = UART_CTS# state inactive. This bit is the complement of the UART_CTS# input signal. If the LOOP bit in the UART Modem Control Register (UARTMCR), Section 10.4.9, is set to 1, the CTS bit is equivalent to the RTS bit in the UARTMCR.
Bit 5	DSR	Data Set Ready. 1 = UART_DSR# state active. 0 = UART_DSR# state inactive. This bit is the complement of the UART_DSR# input signal. If the LOOP bit in the UART Modem Control Register (UARTMCR), Section 10.4.9, is set to 1, the DSR bit is equivalent to the DTR bit in the UARTMCR.
Bit 6	RI	Ring Indicator. 1 = not valid. 0 = always reads 0. This bit has no associated external signal.
Bit 7	DCD	Data Carrier Detect. 1 =UART_DCD# state active. 0 = UART_DCD# state inactive. This bit is the complement of the UART_DCD# input signal. If the LOOP bit in the UART Modem Control Register (UARTMCR), Section 10.4.9, is set to 1, the DCD bit is equivalent to the OUT2 bit in the UARTMCR.
Bits 63:8	*reserved*	Hardwired to 0.

10.4.12 UART Scratch Register (UARTSCR)

This register contains a UART reset bit plus 8 bits of space for any software use.

Bits 7:0	USCR	*UART Scratch Register.* Available to software for any purpose.

Bit 8	URESET	*UART Reset.* 1 = reset UART. 0 = no reset. This bit always reads 0.
Bits 63:9	*reserved*	Hardwired to 0.

11.0 Interrupts

The controller supports interrupts to the CPU on its Int# or NMI# inputs from a variety of causes, and it supports re-routing of interrupts to a PCI host CPU. The following registers are used to configure, report status, and clear interrupts:

- ❑ Interrupt Control Register (INTCTRL), Section 5.5.2 on page 52
- ❑ Interrupt Status Register 0 (INTSTAT0), Section 5.5.3 on page 55
- ❑ Interrupt Status 1/CPU Interrupt Enable Register (INTSTAT1), Section 5.5.4 on page 55
- ❑ Interrupt Clear Register (INTCLR), Section 5.5.5 on page 56
- ❑ PCI Interrupt Control Register (INTPPES), Section 5.5.6 on page 57
- ❑ Watchdog Timer Control Register (T3CTRL), Section 5.6.7 on page 61
- ❑ General-Purpose Timer Control Register (T2CTRL), Section 5.6.5 on page 60
- ❑ Memory Control Register (MEMCTRL), Section 6.6.1 on page 72
- ❑ Memory Check Error Status Register (CHKERR), Section 6.6.3 on page 74
- ❑ PCI Control Register (PCICTRL), Section 7.11.1 on page 91
- ❑ PCI Error Register (PCIERR), Section 7.11.4 on page 103
- ❑ PCI Command Register (PCICMD), Section 7.13.3 on page 107
- ❑ PCI Status Register (PCISTS), Section 7.13.4 on page 108
- ❑ PCI Interrupt Line Register (INTLIN), Section 7.13.13 on page 112
- ❑ PCI Interrupt Pin Register (INTPIN), Section 7.13.14 on page 112
- ❑ PCI Control Register (PCICTRL), Section 7.11.1 on page 91
- ❑ Local Bus Chip-Select Timing Registers (LCSTn), Section 8.6.2 on page 122
- ❑ DMA Control Registers 0 and 1 (DMACTRLn), Section 9.5.1 on page 133
- ❑ UART Interrupt Enable Register (UARTIER), Section 10.4.3 on page 139
- ❑ UART Interrupt ID Register (UARTIIR), Section 10.4.6 on page 140
- ❑ UART Line Status Register (UARTLSR), Section 10.4.10 on page 144
- ❑ UART Modem Status Register (UARTMSR), Section 10.4.11 on page 144

For details on wiring PCI interrupts, see Section 2.2.6 of the *PCI Local Bus Specification*.

On reset, all interrupts are enabled onto Int#0 by default. After reset, each interrupt can be separately enabled and programmed to interrupt the CPU on any of its seven interrupts, Int#[5:0] and NMI#. Most of the interrupt configuration is done in the Interrupt Control Register (INTCTRL), although other registers must also be configured for some of the interrupts. Each of the seven CPU interrupts are separately enabled.

Each CPU interrupt has a 16-bit status field in the Interrupt Status Register 0 (INTSTAT0) or Interrupt Status 1/CPU Interrupt Enable Register (INTSTAT1). The status field shows which interrupt source or sources are requesting service for a particular

CPU interrupt level. A clear bit is available for each interrupt source, although these bits only function for edge-triggered interrupts.

When the controller is the PCI Central Resource (PCICR# asserted at reset), the controller's INTA# signal is bidirectional, rather than an output, so that the controller can accept up to five PCI interrupts on INTA# through INTE#. It forwards these interrupts to the CPU, as specified in Interrupt Control Register (INTCTRL), Section 5.5.2. When the controller is not the PCI Central Resource (PCICR# negated at reset), interrupts may be serviced by a PCI host CPU. CPU Interrupt Level 0 (Int#[0]) may be driven onto PCI interrupt signal INTA#, and CPU Interrupt Level 1 (Int#[1]) may be driven onto the PCI system error signal, SERR#.

Table 31 summarizes the registers used to configure and monitor the causes of these interrupts. For details, see the register descriptions referenced in this table.

Table 31: Interrupt Configuration and Reporting Registers

Interrupt Type	Interrupts Configured In:	Interrupt Status Reported In:	Interrupts Cleared In:
CPU Parity Errors	INTCTRL (Section 5.5.2)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4)	INTCLR (Section 5.5.5)
CPU No-Target Decode	INTCTRL (Section 5.5.2)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4)	INTCLR (Section 5.5.5)
Memory Errors (parity or ECC)	INTCTRL (Section 5.5.2) MEMCTRL (Section 6.6.1)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4) CHKERR (Section 6.6.3)	INTCLR (Section 5.5.5)
DMA Events	INTCTRL (Section 5.5.2) DMACTRLn (Section 9.5.1)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4) DMACTRLn (Section 9.5.1)	INTCLR (Section 5.5.5)
UART Events	INTCTRL (Section 5.5.2) UARTIER (Section 10.4.3)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4) UARTIIR (Section 10.4.6) UARTLSR (Section 10.4.10) UARTMSR (Section 10.4.11)	INTCLR (Section 5.5.5)
Watchdog Timer	INTCTRL (Section 5.5.2) T3CTRL (Section 5.6.7)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4)	INTCLR (Section 5.5.5)
General-Purpose Timer	INTCTRL (Section 5.5.2) T2CTRL (Section 5.6.5)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4)	INTCLR (Section 5.5.5)
Local-Bus Ready Timer	INTCTRL (Section 5.5.2) LCSTn (Section 8.6.5)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4)	INTCLR (Section 5.5.5)
PCI Interrupts (INTE# through INTA#)	INTCTRL (Section 5.5.2) INTPPES (Section 5.5.6) PCICTRL (Section 7.11.1) PCICMD (Section 7.13.3) INTLIN (Section 7.13.13) INTPIN (Section 7.13.14)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4) PCIERR (Section 7.11.4) PCISTS (Section 7.13.4)	INTCLR (Section 5.5.5)
PCI SERR# (System Error) [a]	INTCTRL (Section 5.5.2) PCICTRL (Section 7.11.1)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4) PCIERR (Section 7.11.4) PCISTS (Section 7.13.4)	INTCLR (Section 5.5.5)
PCI Internal Error [b]	INTCTRL (Section 5.5.2) INTPPES (Section 5.5.6) PCICTRL (Section 7.11.1)	INTSTST0 (Section 5.5.3) INTSTST1 (Section 5.5.4) PCIERR (Section 7.11.4) PCISTS (Section 7.13.4)	INTCLR (Section 5.5.5)

a. A *PCI System Error* is an address- or data-parity error on a PCI Special Cycle, or any other serious system error.

b. A *PCI Internal Error* indicates that something bad happened during a PCI transaction; the fault could lie either with the PCI device or the controller.

12.0 Reset and Initialization

At reset, the controller begins the CPU initialization from Serial Mode EEPROM or by self-initialization. Immediately after this initialization, the controller's Physical Device Address Registers (PDARs) and Boot ROM are located at the addresses described in Section 5.4.1. Only the address ranges for Boot ROM (BOOTCS) and the controller's internal registers (INTCS) are accessible at reset. The address spaces of the two SDRAM banks, SDRAM0 and SDRAM1, and the address spaces of the device chip-selects, DCS#[8:2], power up in the disabled state so that main memory and the devices associated with DCS#[8:2] are not accessible. After the boot sequence, software may configure the PDARs to support memory accesses, as described in Section 5.4.

12.1 Types of Reset

The controller supports the following types of reset:

- ❑ *Power-Up Reset:* When the VccOk input from external circuitry transitions between negated and asserted, the controller:
  - resets its internal registers and state.
  - asserts ColdReset# and Reset# to the CPU.
  - asserts PCIRST# on the PCI Bus, if PCICR# is asserted to the controller.
  - samples the configuration signals described in Section 12.2, below.
  - runs the CPU initialization procedure described in Section 12.3, below.
- ❑ *Cold Reset:* When the CLDRST bit is set in the CPU Status Register (CPUSTAT), Section 5.5.1, the controller performs the same actions as for Power-Up Reset, above.
- ❑ *Warm Reset:* When the WARMRST bit is set in the CPU Status Register (CPUSTAT), Section 5.5.1, the controller:
  - asserts Reset# to the CPU.
- ❑ *PCI Cold Reset:* When the PCICRST bit is set in the PCI Control Register (PCICTRL), Section 7.11.1, the controller:
  - resets its PCI logic, including resetting all PCI configuration registers to their reset values (all data and pending operations in the PCI FIFOs are lost).
  - if PCICR# is asserted, asserts PCIRST# on the PCI Bus.
- ❑ *PCI Warm Reset:* When the PCIWRST bit is set in the PCI Control Register (PCICTRL), Section 7.11.1, the controller functions differently, depending on the controller's configuration:
  - if PCICR# is asserted, the controller asserts PCIRST# on the PCI Bus.
  - if PCICR# is negated, all PCI accesses to the controller as PCI target are retried.

The remaining parts of this chapter relate only to the first two types of reset—Power-Up and Cold Reset.

12.2 Power-Up and Cold Reset Configuration Signals

Several signals are sampled at Power-Up and Cold Reset to determine the following properties of the controller's operation:

- *Endian Mode:* The CPU interface can operate in little-endian or big-endian mode. (The memory, PCI-Bus, and Local-Bus interfaces always operate in little-endian mode).
- *PCI-Bus and Local-Bus Width:* Either a 64-bit PCI Bus and no Local Bus, or a 32-bit PCI Bus and a 32-bit Local Bus.
- *PCI Central Resource Function:* The controller can operate either as the PCI Central Resource or in the PCI Stand-Alone Mode.
- *Base Address of Controller Registers and Boot ROM:* The default base addresses for controller internal registers and Boot ROM are 0x0 1FA0 0000 and 0x0 1FC0 0000, respectively. However, these base addresses are different if multiple controllers are used in a system.
- *Controller ID in Multi-Controller Configurations:* When multiple controllers are used in a system, each controller has its own ID number.

The controller drives the CPU's Reset# and ColdReset# signals. Alternatively, software can cause a CPU cold and warm reset by writing to the CLDRST or WARMRST bit in the CPU Interface Registers (Section 5.5 on page 50).

Table 32 and Table 33 show the signals that the controller samples on reset. The controller samples the two UART signals for this purpose only on the rising edge of ColdReset#. The other signals (BigEndian, PCI64# and PCICR#) must be static at all times.

Table 32: Endian and PCI Reset Configuration Signals

Signal Sampled at Reset	When Negated At Reset	When Asserted At Reset
BigEndian [a]	The controller implements a Little-Endian CPU interface.	The controller implements a Big-Endian CPU interface.
PCI64#	The controller implements a 32-bit PCI Bus: • REQ64# becomes LOC_ALE. • ACK64# becomes LOC_CLK. • C/BE#[7:4] becomes LOC_A[3:0]. • PAR64 becomes LOC_A[4]. • PCI_AD[63:32] becomes LOC_AD[31:0]. • The controller's default location for Boot ROM is the Local Bus.	The controller implements a 64-bit PCI Bus: • LOC_ALE becomes REQ64#. • LOC_CLK becomes ACK64#. • LOC_A[3:0] becomes C/BE#[7:4]. • LOC_A[4] becomes PAR64. • LOC_AD[31:0] becomes PCI_AD[63:32]. • The controller's default location for Boot ROM is the memory bus.
PCICR#	The controller is not the PCI Central Resource: • PCLK[0] is an input and PCLK[4:1] are floated. • REQ#[0] is an output and REQ#[4:1] are unused inputs. • GNT#[0] is an input and GNT#[4:1] are floated. • INTA# is an output. • PCIRST# is an input.	The controller is the PCI Central Resource: • PCLK[4:0] are all outputs, and the controller uses PCLK[0] as its PCI-Bus clock. • REQ#[4:0] are all inputs. • GNT#[4:0] are all outputs. • INTA# is bidirectional. • PCIRST# is an output. • The controller configures 64-bit PCI operation with its REQ64# output. • The controller generates PCI Configuration Space cycles.

[a] The BigEndian signal is ORed with Endian Bit (EB) of the Serial Mode EEPROM initialization sequence to determine the CPU's endian mode.

Table 33: Base-Address and ID Reset Configuration Signals

Signal Sampled at Reset		**Controller ID Number**	**Base Address Of Controller's Internal Registers After Reset (PDAR = INTCS)**	**Base Address Of Boot ROM After Reset (PDAR = BOOTCS)**
UART_DTR#	**UART_TxDRDY#**			
0	0	00 (Main Controller)	0x0 1FA0_0000 [a]	0x0 1FC0 0000
0	1	01	0x0 1F80_0000	*disabled*
1	0	10	0x0 1F60_0000	*disabled*
1	1	11	0x0 1F40_0000	*disabled*

[a] This is the base address for all single-controller configurations, and for the Main Controller in a multi-controller configuration.

12.3 PCI Reset Sequencing

When PCICR# is asserted, the controller is the PCI Central Resource and drives PCIRST#. The PCI Bus is held in reset until the CPU clears the PCIWRST bit in the PCI Control Register (PCICTRL, Section 7.11.1).

When PCICR# is negated, and there is no CPU attached to the controller, the controller holds all of its logic in reset while the PCIRST# input is asserted. After PCIRST# is negated, the controller comes out of reset. All PCI accesses to the controller are retried until the controller completes reading the Serial Mode EEPROM.

When PCICR# is negated, and a CPU is attached to the controller, all controller logic and the CPU are held in reset while the PCIRST# input is asserted. After PCIRST# is negated, the controller and the CPU are brought out of reset. All PCI accesses to the controller are retried until the controller completes reading the Serial Mode EEPROM, and the CPU has cleared the PCIRST bit in the PCI Control Register.

12.4 CPU and Controller Initialization

On both power-up and cold resets, the controller drives the CPU's ColdReset# signal. The controller also controls the CPU's serial mode-initialization sequence immediately after the CPU wakes up from power-up or cold reset. From the CPU's point of view at reset, the controller pretends to be the CPU's Serial Mode EEPROM (Section 12.4.2). If the system has no Serial Mode EEPROM, the controller generates a default data stream. If the system has a Serial Mode EEPROM, the controller is connected between it and the CPU and monitors the passing through of serial mode data, making any required corrections.

12.4.1 Reset Signal Control

The controller needs external analog circuitry to provide the VccOk signal, as specified in the *VR5000 Bus Interface User's Manual*. The controller's internal PLL is held in reset as long as VccOk is negated. Between the time VccOk is asserted to the controller, and the controller negates ColdReset# to the CPU, the internal controller PLL is locking up. The skew-controlled clock from the PLL is only used inside the controller after ColdReset# is negated. Before then, all active logic is running off an unbuffered raw clock.

The controller internally synchronizes VccOk to SysClk. If the assertion of VccOk meets setup and hold times, VccOk needs be asserted for a minimum of only one SysClk.

When the external circuit asserts VccOk, the controller continues to asset ColdReset# to the CPU and begins to read the initialization sequence (Section 12.4.2) while still holding the CPU in reset. After the controller reads a byte of mode information, it asserts CntrVccOk to the CPU and counts 64K SysClocks before synchronously negating ColdReset#. 64 SysClocks later, the controller negates Reset#. When a cold software reset occurs, the same sequence takes place, although the reset indication originates internally.

When a warm software reset occurs, the controller synchronously asserts Reset# for 64 SysClocks. The ColdReset# signal remains negated throughout a warm software reset and the controller's other operations are unaffected.

12.4.2 Initialization Sequence

The CPU needs a serial stream of initialization data, as defined in Sections 5.2 and 5.3 of the *VR5000 Bus Interface User's Manual*. This data stream may come from a Serial Mode EEPROM or from the controller itself (self-initialization).

If the Serial Mode EEPROM alternative is chosen, the SGS-Thomson M93C46-W or the Microchip Technology 93AA46 Serial EEPROM, or equivalent, must be used. The EEPROM must support the following features:

- ❑ 64 x 16 configuration.
- ❑ Sequential read operation.
- ❑ 3.3V supply voltage.
- ❑ Microwire bus interface.
- ❑ Clock frequency of 800 kHz (SysClock/128).

If the self-initialization alternative is chosen, the PROM_SD signal must be pulled up. If an alternate source is used, care must be taken to provide the CPU with controller-compatible initialization data.

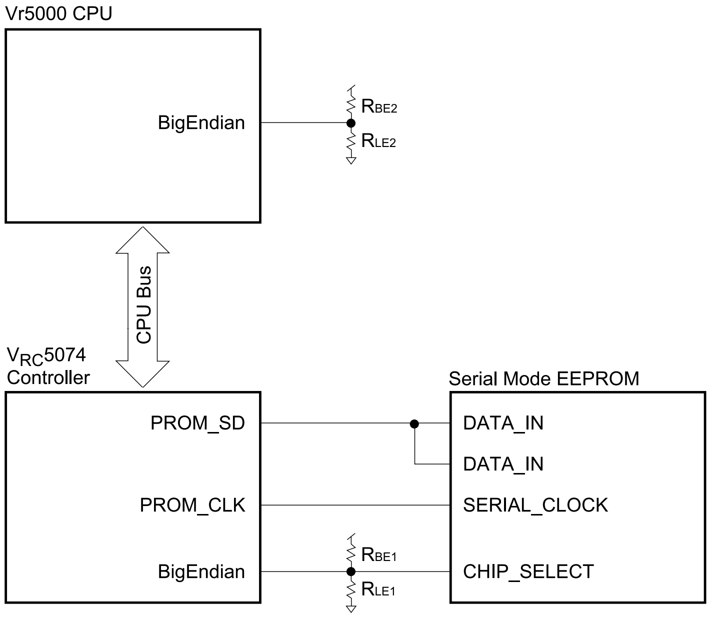
12.4.2.1 Connecting the Serial Mode EEPROM

The M93C46-W Serial EEPROM (or equivalent) has separate DATA-IN and DATA-OUT signals, but the controller has only one bidirectional signal, PROM_SD, to which the EEPROM data signals should connect and on which both address and data appear.

Figure 19 illustrates the connections. The EEPROM's DATA-IN and DATA-OUT signals should be tied together and then connected to the controller's PROM_SD signal. (For details on how these signals should be tied together, see the SGS-Thomson Application Note AN394 *Microwire EEPROM Common I/O Operation*.) The controller's PROM_CLK output should be connected to EEPROM's SERIAL_CLOCK input. The controller's BigEndian signal should be connected to the EEPROM's CHIP_SELECT signal and tied to either Vcc or GND through a resistor. The controller can then drive BigEndian to select the EEPROM and read its initialization data. After initialization, the controller uses BigEndian as an input to indicate the endian mode for the controller.

Resistors R_{BE1} and R_{BE2} *or* R_{LE1} and R_{LE2} must be used to select the endian mode for the controller and the CPU; four resistors are is shown in Figure 19 but only two should be used.

Figure 19: Serial Mode EEPROM Signal Connections



4373-092.eps

12.4.2.2 Initialization Data

Upon power-up or cold reset in the Serial Mode EEPROM initialization alternative, the controller sends a read command and an address (location 0) to the Serial Mode EEPROM to obtain a byte of mode information. The EEPROM will drive a 0 on its Data Out pin during the clock in which the controller is sending its final address bit. Since the controller is sending an address of 0, there is no bus conflict, and the EEPROM Data In and Data Out pins can be tied together.

Then, the controller asserts the CntrVccOk signal to the CPU and monitors the CPU's ModeClock output. When ModeClock goes Low, the controller shifts the first initialization byte out of a holding register (corrected if necessary), drives it onto the ModeOut signal (to the CPU's ModeIn), and reads the next byte from the Serial Mode EEPROM. This process continues until all mode information is read from the Serial Mode EEPROM and provided to the CPU. Since ModeClock runs at SysClock divided by 256, and PROM_CLK runs at SysClock divided by 128, the controller can read mode data and provide a continuous uninterrupted stream to the CPU.

The controller passes 256 bits of configuration data to the CPU, beginning with bit 0 of the serial data stream from the EEPROM. In addition, the EEPROM contains 37 bits of controller-specific configuration data. Table 34 shows the complete set of CPU and controller initialization data.

Table 34: Serial Initialization Data Stream

Bit	Function	Default Value Generated By Controller When No Serial Mode EEPROM Is Present	Restrictions Enforced By Controller [a]	Description
0	*reserved*, Must be 0.	0	*none*	First bit shifted out of Serial Mode EEPROM.
4:1	XmitDatPat	0 (DDDD)	Bits 4:3 forced to 0	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
7:5	SysCkRatio	0 (multiply by 2)	*none*	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
8	EndBit	0 (little-endian)	*none*	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
10:9	Non-Block Write	2 (pipelined writes)	Forced to 2	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
11	TmrIntEn	0 (timer interrupt enabled)	*none*	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
12	Secondary Cache Enable	0 (secondary cache disabled)	*none*	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
14:13	DrvOut	2 (100%)	*none*	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
15	*reserved*, Must be 0.	0	*none*	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
17:16	Secondary Cache Size	0 (512Kbyte)	*none*	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
255:18	*reserved*, Must be 0.	0	*none*	See Section 5.3 of the *V_R5000 Bus Interface User's Manual,*
256	Controller Boot ROM Location	0 if PCI64# negated 1 if PCI64# asserted	*none*	0 = Local Bus 1 = Memory Bus See description of MEM/LOC bit in PDAR (Section 5.4).
258:257	Controller Boot ROM Size	0 (8 bits)	*none*	0 = 8 bits 1 = 16 bits 2 = 32 bits 3 = 64 bits See description of WIDTH field in PDAR (Section 5.4).
260:259	Controller PCI Clock Speed	0	*none*	See description of CLKSEL field in PCICTRL (Section 7.11.1).
276:261	Controller PCI SSVID	0	*none*	See description of SSVID register (Section 7.13.11).
292:277	Controller PCI SSID	0	*none*	See description of SSID register (Section 7.13.12).

[a] If a Serial Mode EEPROM is present, the controller monitors and if necessary corrects the values of certain parameters. This function ensures that the CPU-controller interface operates as intended.

12.4.3 In-Circuit Programming of the Serial Mode EEPROM

The Serial Mode EEPROM cannot be written under CPU control. However, with appropriate external circuitry, in-circuit programming of the EEPROM can be accomplished when the VccOk signal is Low. At that time, the controller is in reset with its PROM_SD bidirectional signal tri-stated and its PROM_CLK output driven Low. To program the

EEPROM in-circuit, hold VccOk Low, OR the external EEPROM clock with PROM_CLK from the controller, and drive the data into the EEPROM. Drive the external EEPROM clock Low upon completion, and tri-state the external data source. Alternatively, jumpers can be used to connect PROM_SD, PROM_CLK, and the BigEndian chip-select for this in-circuit programming configuration.

When the controller is the Main Controller in a multi-controller configuration (Section 5.3.1), it is the only controller that can drive PROM_CLK; the other controllers tri-state. Therefore, a third technique of in-circuit programming is to hold VccOk Low to this Main Controller and temporarily drive its UART_TxDRDY# or UART_DTR# input High. By doing this, the controller thinks it is not the Main Controller, and both PROM_CLK and PROM_SD are tri-state. External circuitry can then drive these signals such that in-circuit EEPROM programming can proceed.

13.0 Endian-Mode Software Issues

13.1 Overview

The native endian mode for MIPS processors, like Motorola and IBM 370 processors, is *big-endian*. However, the native mode for Intel (which developed the PCI standard) and VAX processors is *little-endian*. For PCI-compatibility reasons, most PCI peripheral chips, including the VRC5074 controller, operate natively in *little-endian* mode.

While the VRC5074 controller is natively little-endian, it supports either big- or little-endian mode on the CPU interface. The state of the BigEndian signal at reset or the Big Endian (BE) bit of the Serial Mode EEPROM initialization sequence (Section 12.0) determines this endian mode. However, there are important considerations when using the controller in a mixed-endian design. The most important aspect of the endian issue is which byte lanes of the SysAD bus are activated for a particular address.

If the big-endian mode is implemented for the CPU interface, the controller swaps bytes within words and halfwords that are coming in and going out on the SysAD bus. All of the controller's other interfaces operate in little-endian mode. There are a number of implications associated with this:

- ❑ Data in memory is always ordered in little-endian mode, even with a big-endian CPU interface.
- ❑ Little-endian bit-fields and other data structures that span two or more bytes (such as bit-fields within registers or FIFOs) are fragmented when the CPU interface is big-endian. The contents of these data structures are byte-swizzled, so that the bits are arranged [7:0], [15:8], [23:16], [31:24], [39:32], [47:40], [55:48], [63:56], rather than [63:0].
- ❑ Big-endian devices on the PCI Local Bus or the I/O Local Bus must be byte-swapped external to the controller.

The sections below view the endian issue from a programmer's perspective. They describe how to implement mixed-endian designs and how to make code endian-independent.

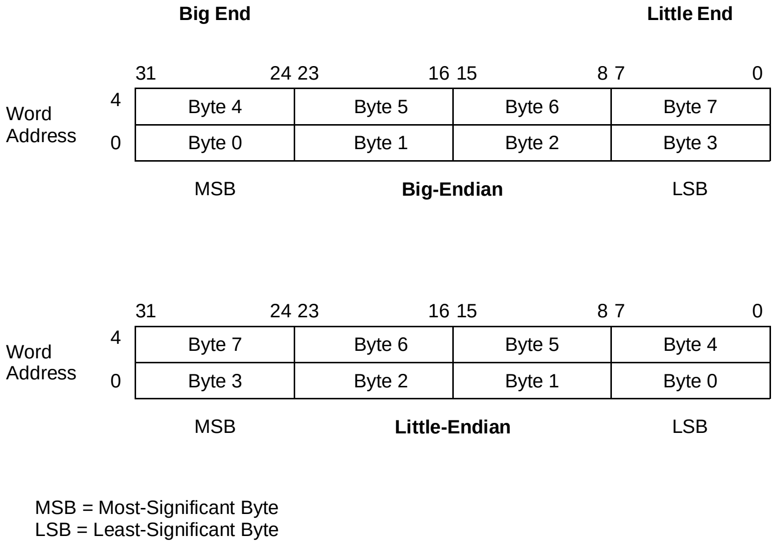
13.2 Endian Modes

The endian mode of a device refers to its word-addressing method and byte order:

- ❑ *Big-Endian* devices address data items at the *big end* (most-significant bit number). The most-significant byte (MSB) in an addressed data item is at the *lowest* address.
- ❑ *Little-Endian* devices address data items at the *little end* (least-significant bit number). The most-significant byte (MSB) in an addressed data item is at the *highest* address.

Figure 20 shows the bit and byte order of the two endian modes, as it applies to bytes within word-sized data items. The *bit order* within bytes is the same for both modes. The big (most-significant) bit is on the left side, and the little (least-significant) bit is on the right side. Only the *bit order* of sub-items is reversed within a larger addressable data item (halfword, word, doubleword, quadword) when crossing between the two endian modes. The sub-items' *order of significance* within the larger data item remains the same. For example, the least-significant halfword (LSHW) in a word is always to the right and the most-significant halfword (MSHW) is to the left.

Figure 20: Bit and Byte Order of Endian Modes



Big End **Little End**

Big-Endian

Word Address	31 24	23 16	15 8	7 0
4	Byte 4	Byte 5	Byte 6	Byte 7
0	Byte 0	Byte 1	Byte 2	Byte 3
	MSB			LSB

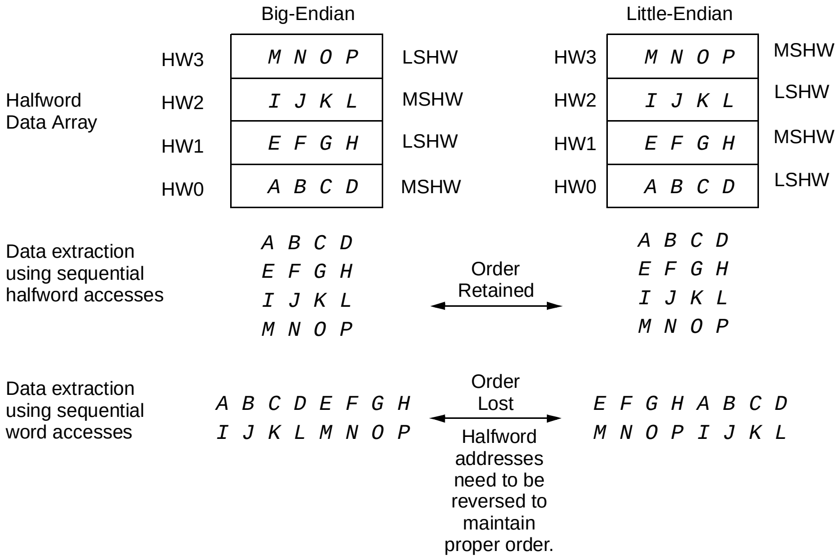
Little-Endian

Word Address	31 24	23 16	15 8	7 0
4	Byte 7	Byte 6	Byte 5	Byte 4
0	Byte 3	Byte 2	Byte 1	Byte 0
	MSB			LSB

MSB = Most-Significant Byte
LSB = Least-Significant Byte

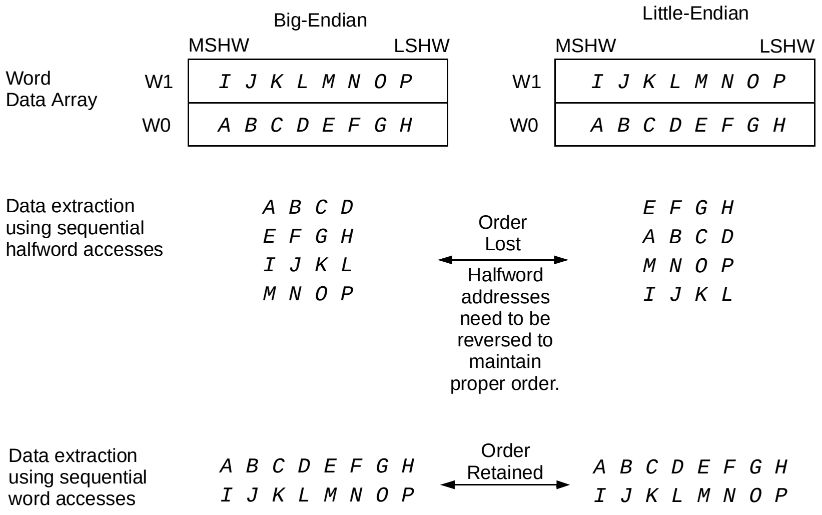
If the access type matches the data-item type, no swapping of data sub-items is necessary. Thus, when making halfword accesses into a data array consisting of halfword data (Figure 21), no byte-swapping takes place. In this case, data-item *bit order* is retained between the two endian modes. The code that sequentially accesses the halfword data array would be identical regardless of the endian protocol of its CPU. The code would be endian-independent.

Figure 21: Halfword Data-Array Example



However, when making halfword accesses into a data array consisting of word data (Figure 22), access to the *more-significant* halfword requires the address corresponding to the *less-significant* halfword (and vice versa). Such code is not endian-independent. A supergroup access (e.g. accessing two halfwords simultaneously as a word from a halfword data array) causes the same problem. Such problems also arise when a halfword access is made into a 32-bit I/O register, whereas a word access into a 32-bit register creates no problem.

Figure 22: Word Data-Array Example



13.3 LAN Controller Example

The AMD AM79C791 LAN controller is one example of how a PCI-Bus device that is natively little-endian adapts to mixed-endian environments. This LAN controller provides limited support for big-endian system interfaces. Its designers assumed only two data types: a 32-bit word corresponding to the width of I/O registers, and an 8-bit byte corresponding to the width of the Ethernet DMA FIFO.

13.3.1 DMA Accesses from Ethernet FIFO

Ethernet data packets consist of bytes. To maximize bus bandwidth, these bytes are transferred via 32-bit word DMA accesses into memory. This access-data mismatch corresponds to the supergroup scenario shown at the bottom of Figure 21. The mismatch means that a byte-swap must be performed to allow the little-endian LAN controller to access the big-endian memory. The LAN controller provides its own internal hardware for this byte-swap.

13.3.2 Word Accesses to I/O Registers

The LAN controller’s designers assumed that the 32-bit internal I/O registers would be accessed by 32-bit word transfers. In that case, the access type and data type match, and no swapping of bytes or halfwords is needed because order of significance is the same for both endian modes. For such word transfers, the I/O register model is endian-independent, and the LAN controller's designers did not provide internal swapping hardware for non-word accesses into the I/O registers.

Word accesses offer the advantage that the register address values documented in the AM79C971 Technical Manual can be used without change (although offsets for individual register fields such as the PCI Latency Timer must be ignored). The position of individual register fields as well as byte position within these fields would also remain the same as documented in the Technical Manual.

13.3.3
Byte or Halfword Accesses to I/O Registers

Word accesses can cause some inconvenience (e.g. shadow registers) when modifying only one or two fields within a 32-bit PCI register. In this case, byte or halfword access to the 32-bit register may be simpler. This type of transfer is analogous to the halfword access into a data array consisting of word data types, shown in Figure 22. Such accesses are mismatched to the defined data type and must be *cross-addressed* to get the byte or halfword of interest. The AM79C791 LAN controller does not provide big-endian hardware support to deal with byte or halfword transfers into the I/O registers. Code written to perform byte or halfword accesses into the 32-bit I/O registers will not be endian-independent.

The I/O register-field addresses documented in the AM79C971 Technical Manual are based on a register model derived from a little-endian perspective. The number order of these addresses progresses from right (least-significant) to left. However, a big-endian system will respond to all addresses as if the number order progresses from left (most-significant) to right. To access the desired byte or halfword, the address order documented in the Technical Manual must be reversed.

The fields of the PCI Status Register and PCI Command Register are two examples of frequently used I/O register fields. The address offsets documented in the Technical Manual are 0x06 and 0x04, respectively. The PCI Command Register field is located in the less-significant halfword of the 32-bit I/O register that is also located at offset 0x04. The PCI Command Register field shares the same offset with its 32-bit register because of the little-endian number order. In a big-endian system, the more-significant halfword (i.e. PCI Status Register field) would share the same offset value with its 32-bit register. So, if the offset 0x04 is used to access the PCI Command Register field, a big-endian system would actually access the PCI Status Register field. To access the proper halfword, the offsets must be exchanged between the two 16-bit register fields. In other words there must be a reversal (or swapping) of number order, relative to the information documented in the Technical Manual.

These special addressing considerations are completely independent of the operand pointers associated with the CPU register used as source or destination. The source or destination within the CPU's register file can be at any location, size, or alignment without altering the transfer results. A common error is to byte-swap CPU register data when transferring a halfword to or from a 32-bit register. The order of significance is the same for both endian modes, so no byte-swap is needed. This is purely an addressing problem.

Table 35 and Table 36 show how the offsets in the AM79C971 Technical Manual are swapped with the other offsets to produce the proper cross-addressed offset required by big-endian systems. The determining factors for the swap are the values of the two least-significant bits of the offsets. According to the AM79C971 Technical Manual, the

PCI Command Register field has the offset 0x04. Table 36 shows that the offset 0x06 is needed to access the PCI Command Register field. The two least-significant bits of 0x04 are b00, which convert to b10 to give the result of 0x06h.

Table 35: Cross-Addressing for Byte Accesses Into a 32-bit I/O Register

Least-Significant Bits of Offset From AM79C971 Technical Manual	Least-Significant Bits of Offset Required by Big-Endian System
b00	b11
b01	b10
b10	b01
b11	b00

Table 36: Cross-Addressing for Halfword Accesses into a 32-bit I/O Register

Least-Significant Bits of Offset From AM79C971 Technical Manual	Least-Significant Bits of Offset Required by Big-Endian System
b00	b10
b10	b00

13.4 GUI Controller Example

The Cirrus Logic CL-GD5465 GUI controller is another example of a PCI-Bus device that offers some mixed-endian support. The designers of this GUI controller assumed three data types: 32-bit word, 16-bit halfword, and 8-bit byte. Unlike the LAN controller which could make certain assumptions as to data type (for I/O register or DMA FIFO accesses), the GUI hardware cannot determine what data type will be used during any particular data transfer; any data type might be involved in any I/O register or RDRAM access.

The data type must be known for a given bus transfer so that the appropriate byte or halfword swap can be performed. The data types may change from bus cycle to the next; one software task may be operating in parallel with and independently of another software task. One of the easiest methods to accommodate such an environment, without semaphores and such, is to provide address apertures into the memory space.

The aperture scheme calls for GUI hardware resources to be mirrored into three address ranges. Depending on which address range selected, a specific data type and data swap is used. Chapter 13 of the CL-GD5465 Technical Manual gives details of these three apertures.

13.4.1 Word Accesses to I/O Registers

The GUI controller’s internal 32-bit I/O registers can be accessed with 32-bit word transfers. In this case, the access type and data type match; no swapping of bytes or halfwords is required because the order of significance is the same for both endian modes. With such word transfers, the I/O register model is endian-independent, so the first address aperture described in the CL-GD5465 Technical Manual is used.

Word accesses have the advantage that the register address values documented in the Technical Manual can be used without change (although offsets for individual register fields such as the PCI Latency Timer must be ignored). The position of individual register fields as well as byte position within these fields also remains the same as shown in the Technical Manual.

13.4.2 Byte or Halfword Accesses to I/O Registers

As in the LAN-controller example, byte or halfword access may be simpler than word accesses when modifying only one or two fields within a 32-bit I/O register. This type of transfer is analogous to the halfword access into a data array consisting of word data types, shown in Figure 22. Such accesses are mismatched to the defined data type and must be swapped to get the byte or halfword of interest. Code written to perform byte or halfword accesses into the 32-bit word I/O registers will not be endian-independent.

There are two methods to perform byte or halfword accesses into the GUI controller. The first method is the use of the apertures for halfword-swap (second aperture) and byte-swap (third aperture). This method has the advantage that the little-endian addresses documented in the Technical Manual are the same as those used by big-endian code, except for the addition of the offset required to select the appropriate aperture. (As of this printing, the second aperture remains unverified and has generated some confusion resulting from poor documentation or improper implementation.)

The second method of performing byte or halfword accesses is to *cross-address* the transfer. Care must be taken, however, when referencing the CL-GD5465 Technical Manual. The I/O register field addresses documented in the Technical Manual are based on a little-endian register model. The number order of these addresses progress from right (least-significant) to left. However, big-endian systems respond to addresses as if the number order progresses from left (most-significant) to right. To access the desired byte or halfword, the address order documented in the Technical Manual must be reversed.

13.4.3 Accesses to RDRAM

The CL-GD5465 GUI controller’s internal pixel and video engines constrain the RDRAM to be little-endian. Here again, big-endian systems have a few problems accessing data subgroups, such as a single byte access into a 32-bit data type. Sub-item accesses are also a factor for RDRAM and the cross-addressing and address apertures solutions are the same as those described in Section 13.4.2. Supergroup access are also encountered with RDRAM. This situation is mentioned in Section 13.2 and shown in Figure 22. A specific GUI-oriented example of this would be an 8-bit data type, such as a pixel, which is transferred four-at-a-time to maximize PCI-Bus bandwidth.

There are two methods for dealing with supergroup transfers. First is the address-aperture method, used in the sub-item scenario of Section 13.4.2. The third aperture, byte-swap, is used to provide the proper data swap for the four 8-bit pixel case. The second aperture, halfword-swap, is used to transfer such things as two 16-bit pixels simultaneously.

The second aperture method requires that the data order in the CPU register be swapped prior to an RDRAM write access, or immediately after an RDRAM read access. To continue with the previous four-pixel transfer example, the byte number-order of the four pixels in the CPU register would be reversed. Now the pixel number-order increases, starting from the right side of the register (first pixel originally on left, now on right). Then, the four pixels are written into the RDRAM with a standard 32-bit word transfer (first aperture). The case of two 16-bit pixels requires the two halfwords to be swapped, but not the order of the two bytes inside the halfwords. This second method is probably more time-consuming and is not recommended.

14.0 Timing Diagrams

This section shows timing diagrams for the controller's various operations on the memory bus and PCI Bus. The following notation is used:

A or An means Address or sequential Address number

D or Dn means Data or sequential Data-item number

14.1 CPU Accesses to Local Memory

Figure 23 through Figure 32 show the timing for CPU accesses to the controller's local memory, including:

- ❑ CPU Single-Byte Memory Read (Figure 23)
- ❑ CPU Single-Byte Memory Write (Figure 24)
- ❑ CPU Eight-Byte Memory Read (Figure 25)
- ❑ CPU Eight-Byte Memory Write (Figure 26)
- ❑ CPU Block (32-Byte) Memory Read (Figure 27)
- ❑ CPU Block (32-Byte) Memory Write (Figure 28)
- ❑ CPU Back-To-Back Eight-Byte Memory Read (Figure 29)
- ❑ CPU Back-To-Back Eight-Byte Memory Write (Figure 30)
- ❑ CPU Back-To-Back Block (32-Byte) Memory Read (Figure 31)
- ❑ CPU Back-To-Back Block (32-Byte) Memory Write (Figure 32)

All SDRAM accesses are full-dword (64 bit) accesses. The controller internally implements partial-dword (less than 64-bit) write requests as read-merge-writes: it first reads from the write address, then merges the partial-dword write data into the read data, then writes the full dword to memory. Because of this, partial-dword writes take longer than full-dword writes.

Figure 23: Single-Byte Memory Read

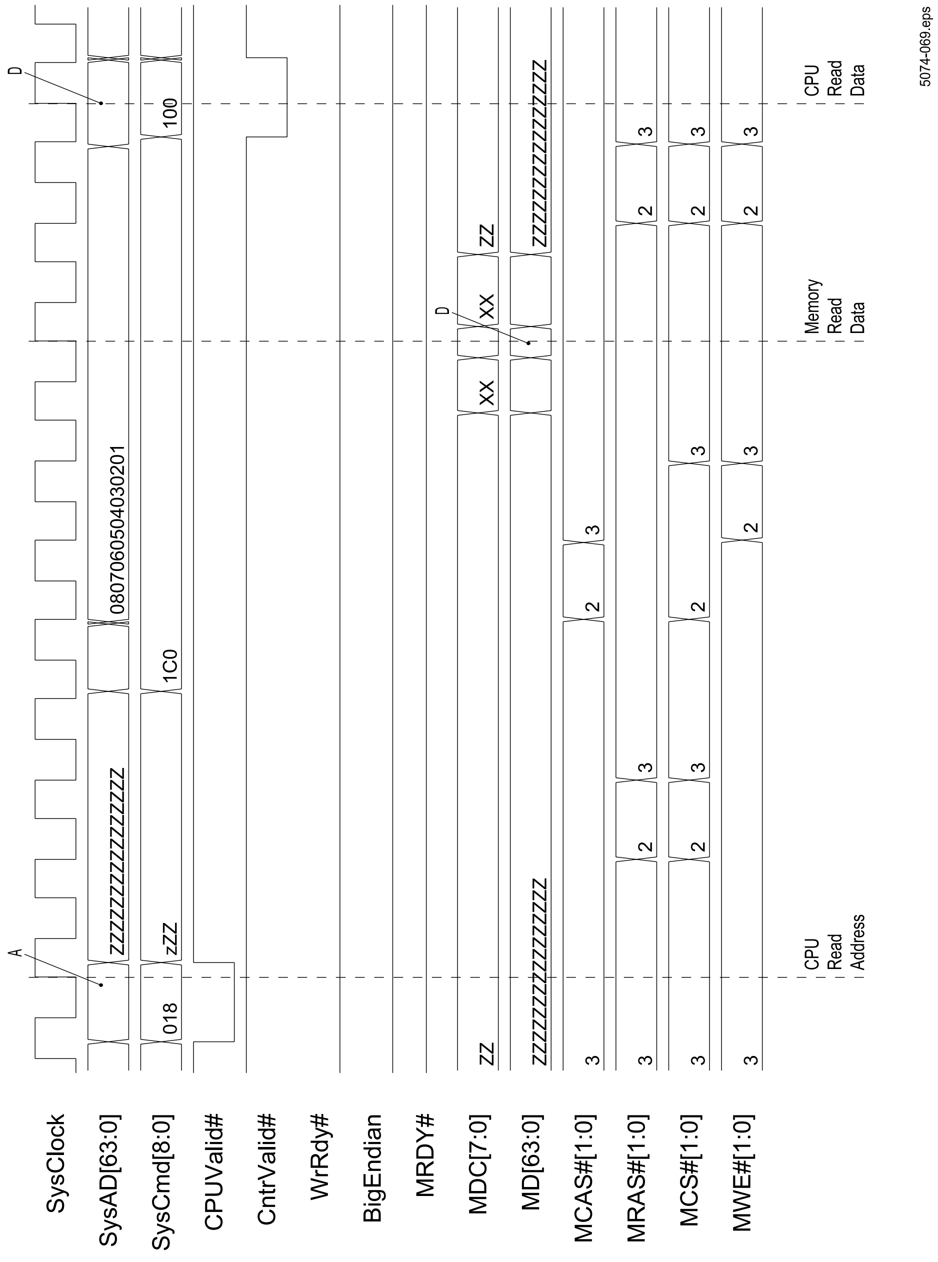
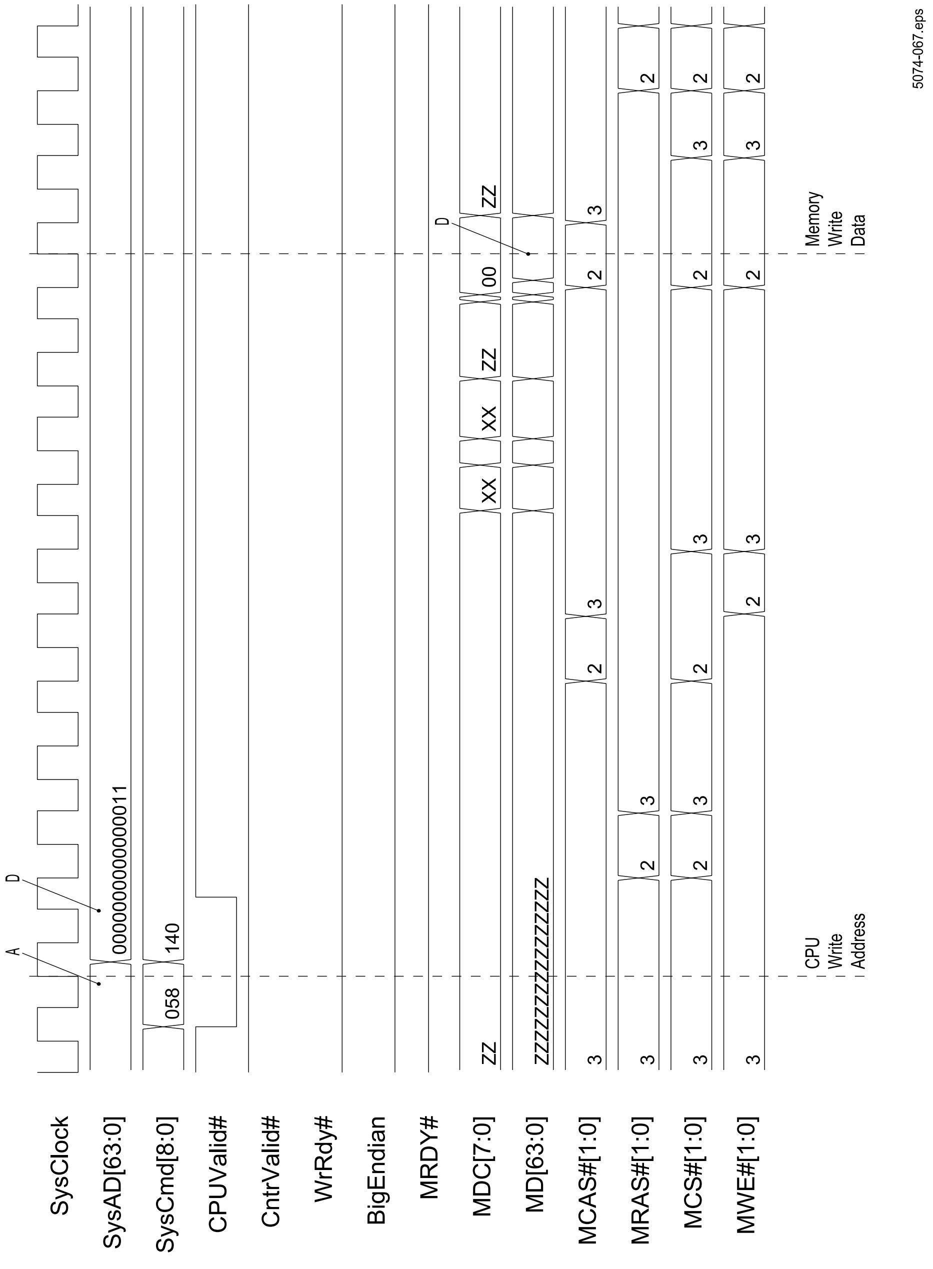


Figure 24: Single-Byte Memory Write



NEC

Figure 25:   Eight-Byte Memory Read

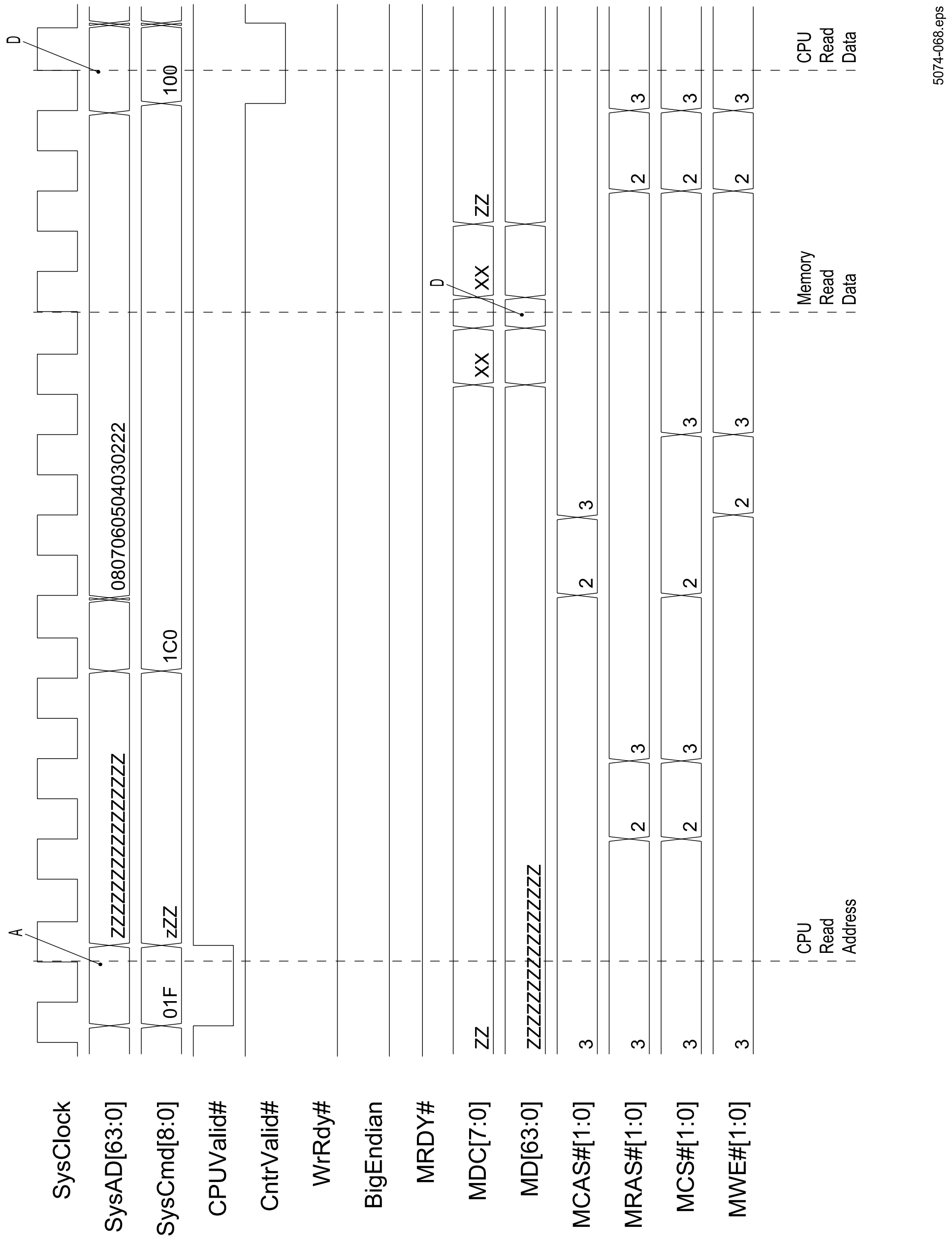


Figure 26: Eight-Byte Memory Write

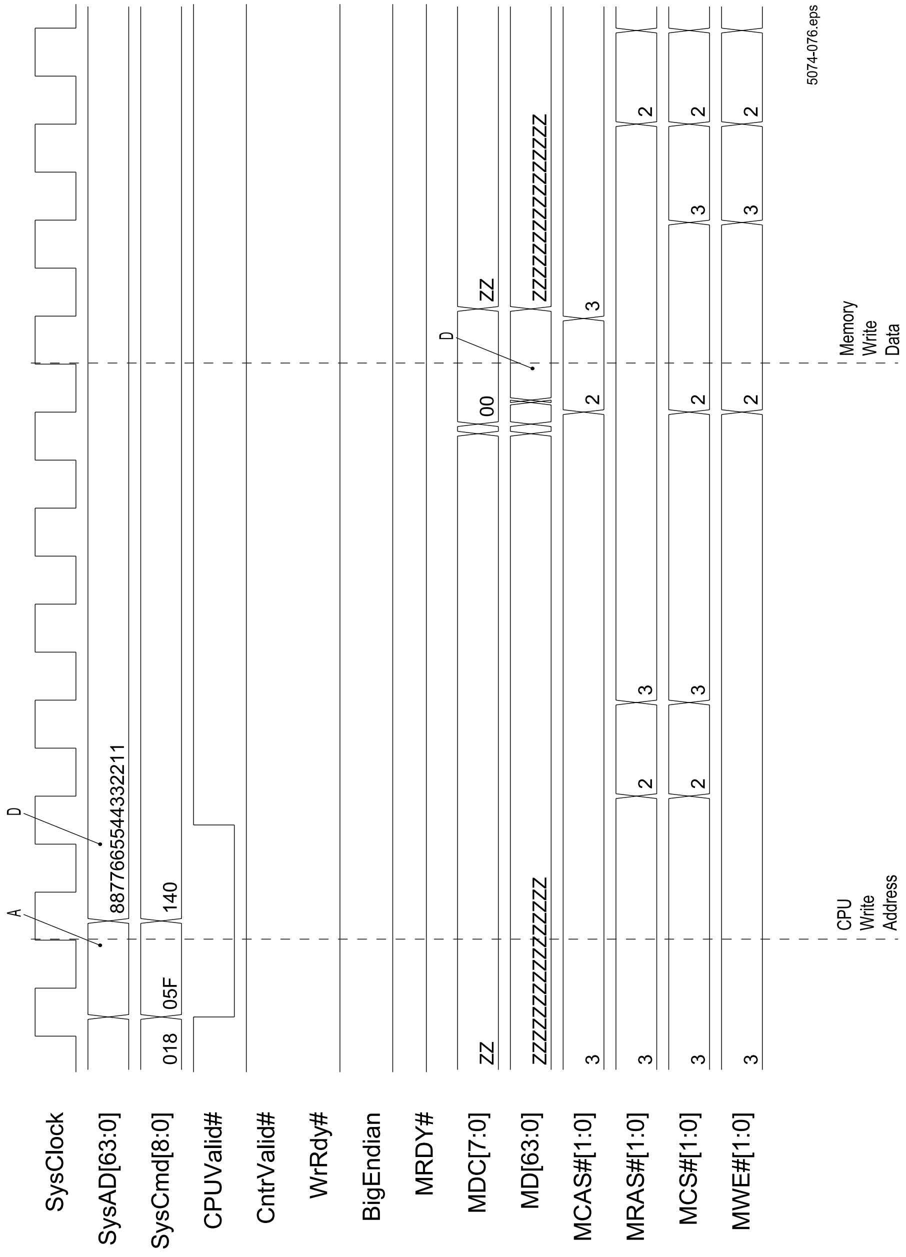


Figure 27: Block Memory Read

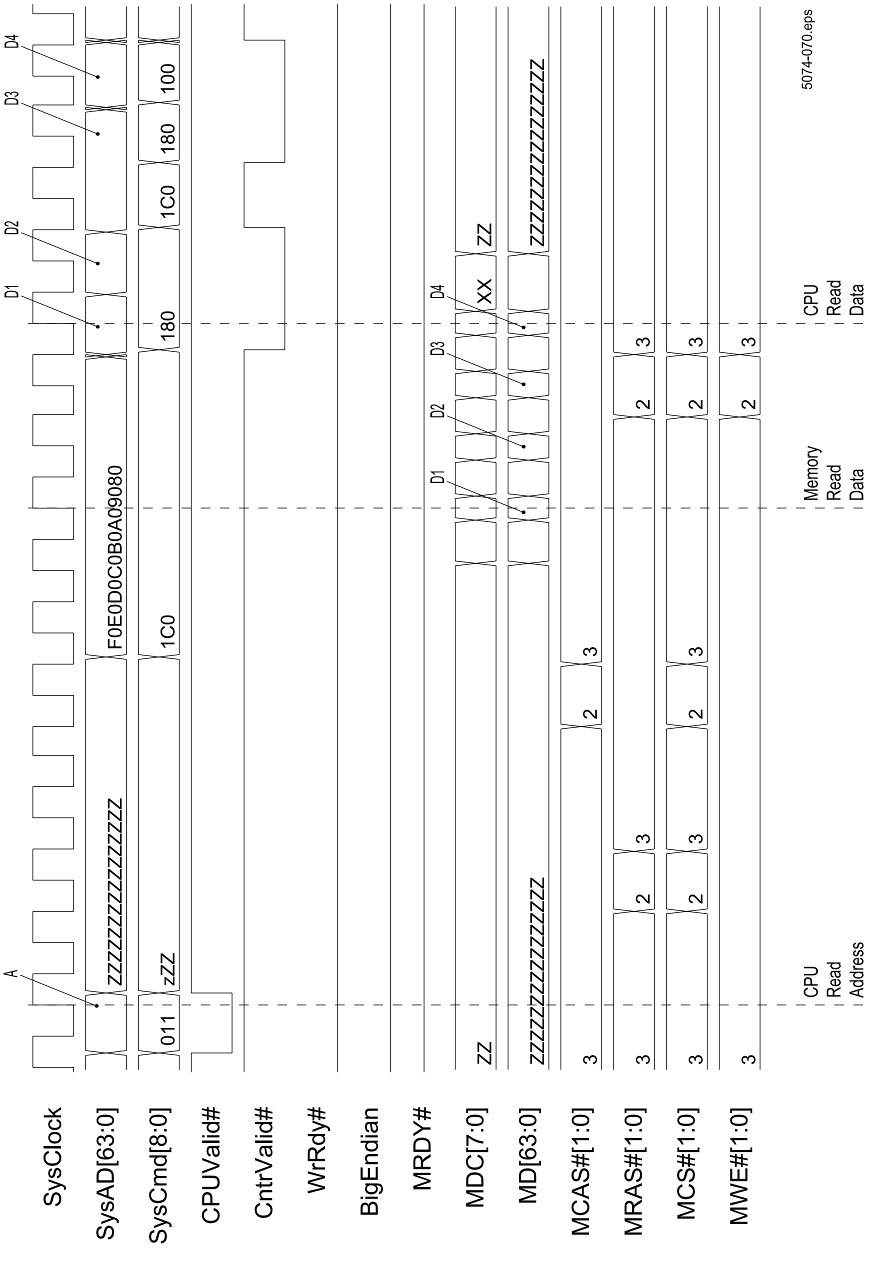


Figure 28: Block Memory Write

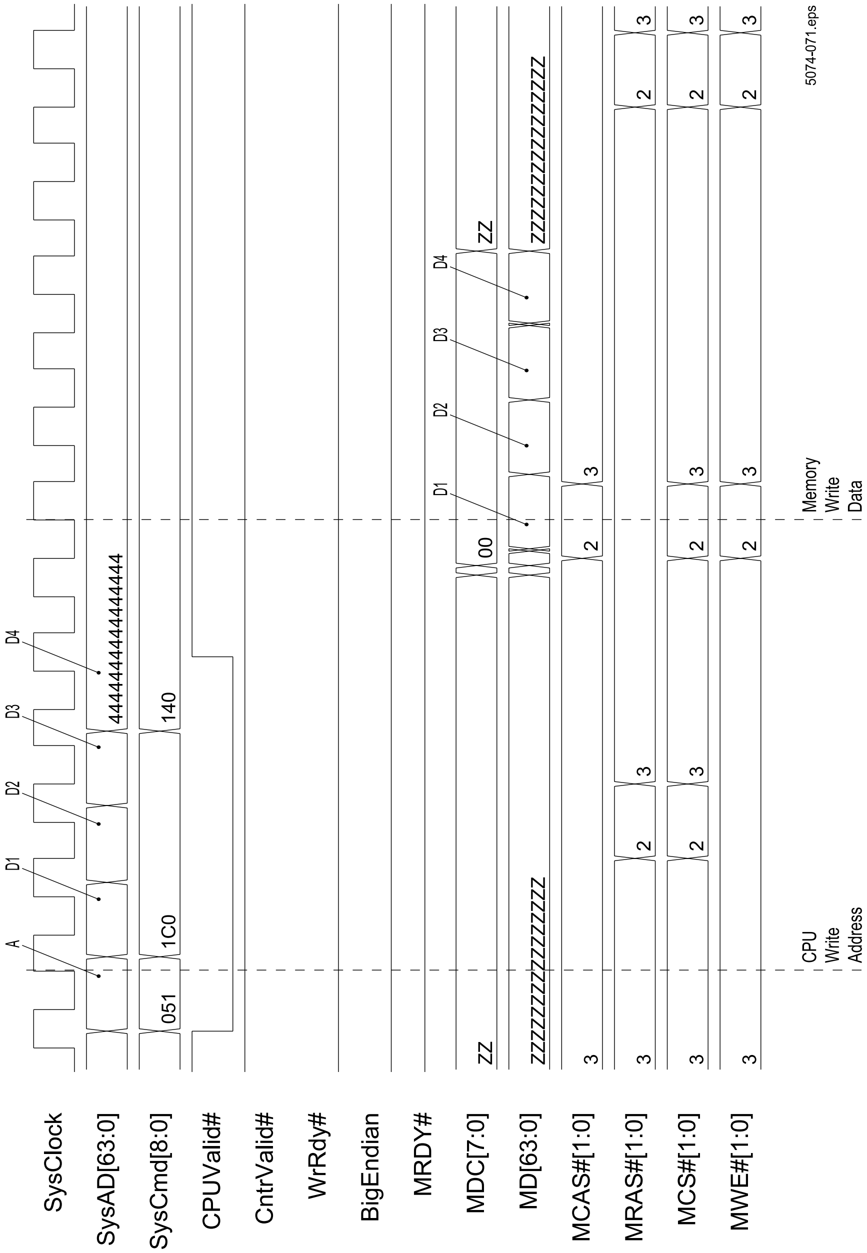


Figure 29: Back-To-Back Eight-Byte Memory Reads

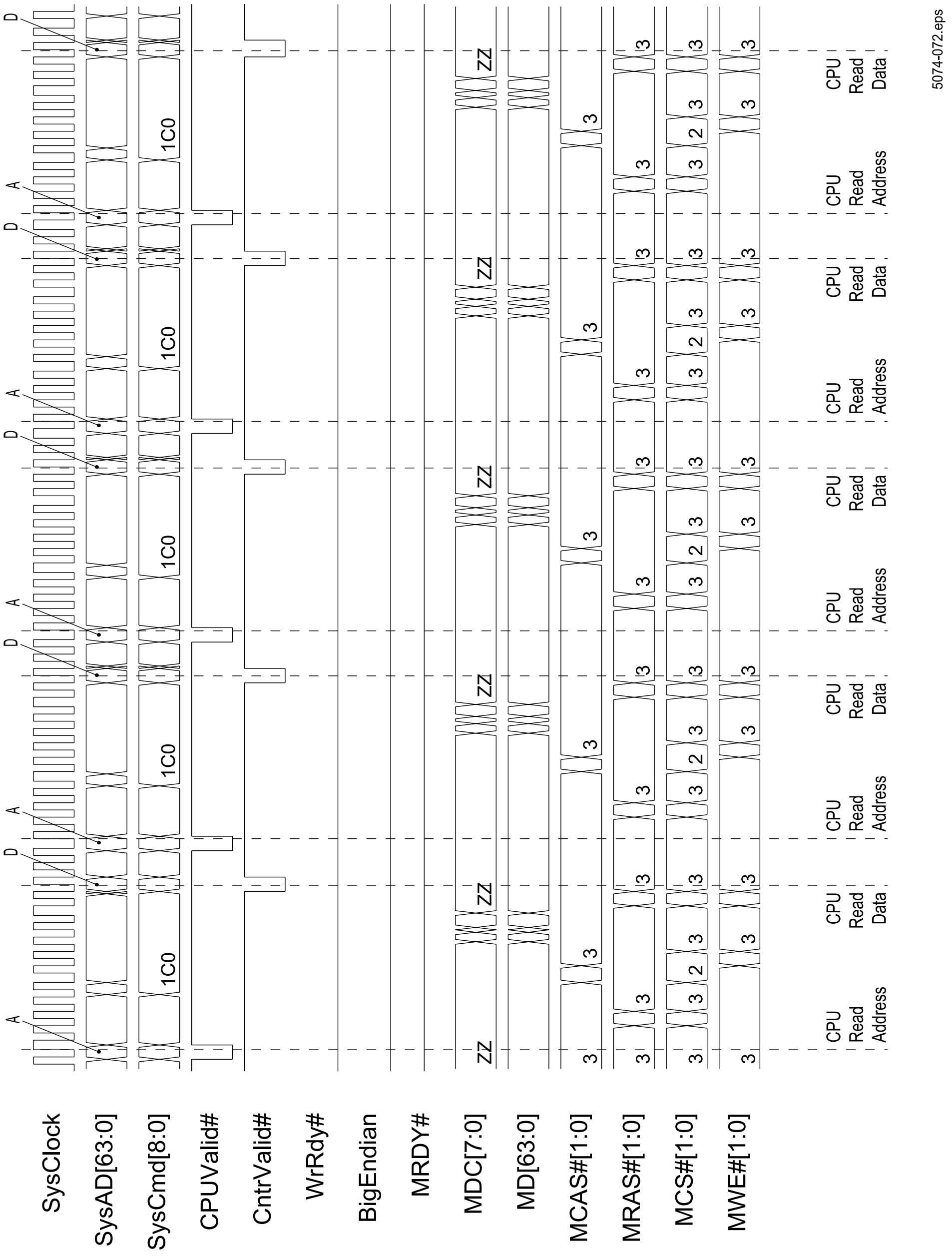


Figure 30: Back-To-Back Eight-Byte Memory Writes

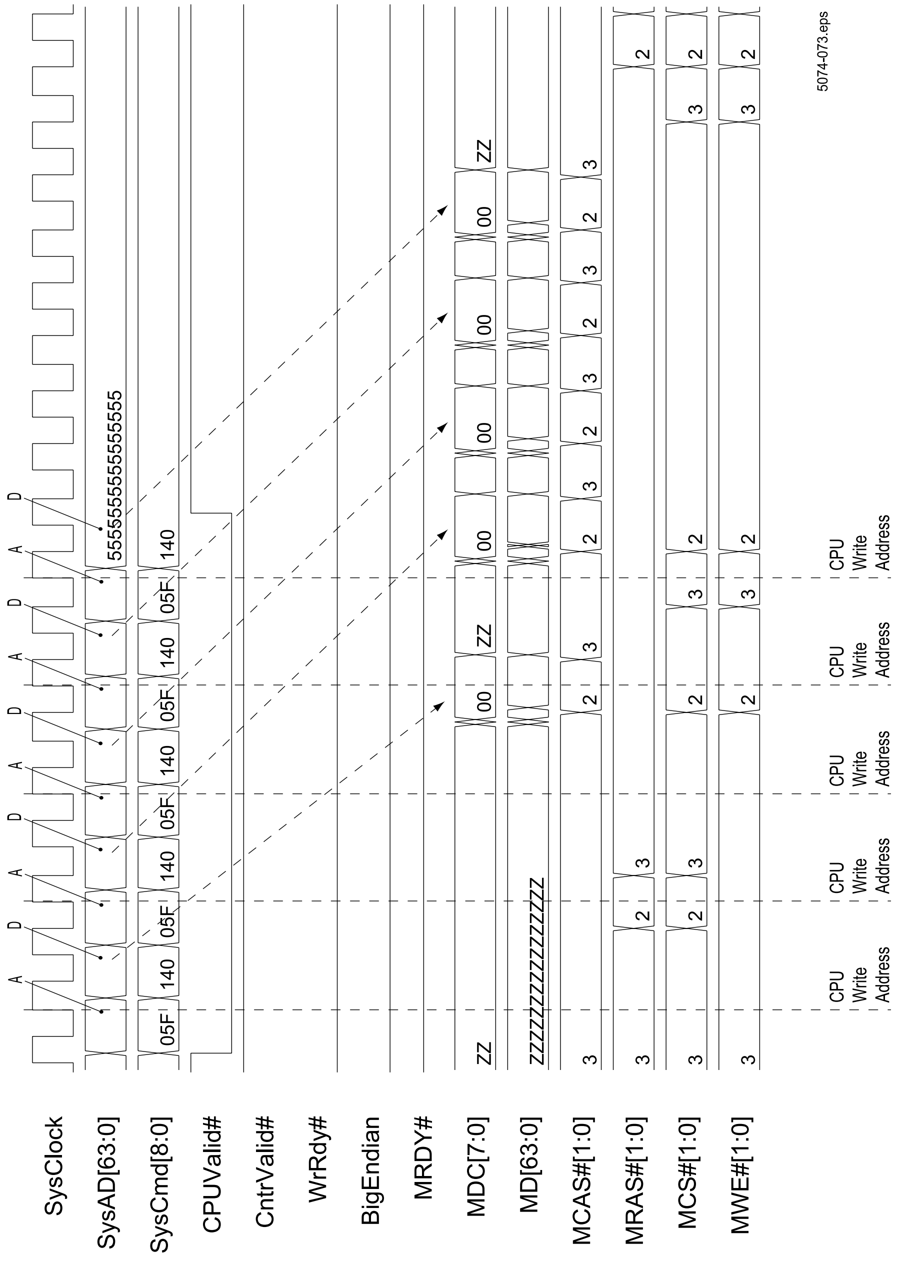


Figure 31: Back-To-Back Block Memory Reads

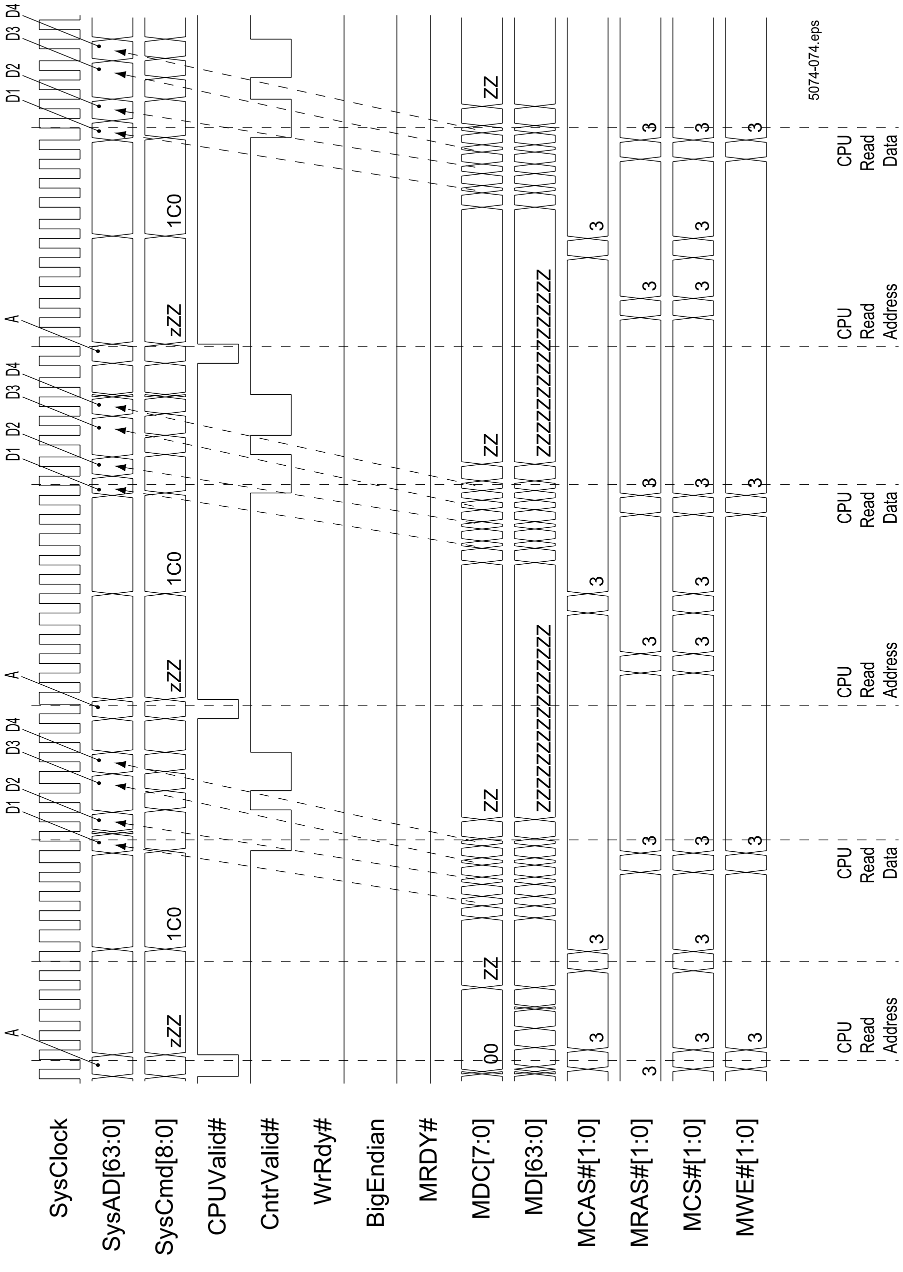
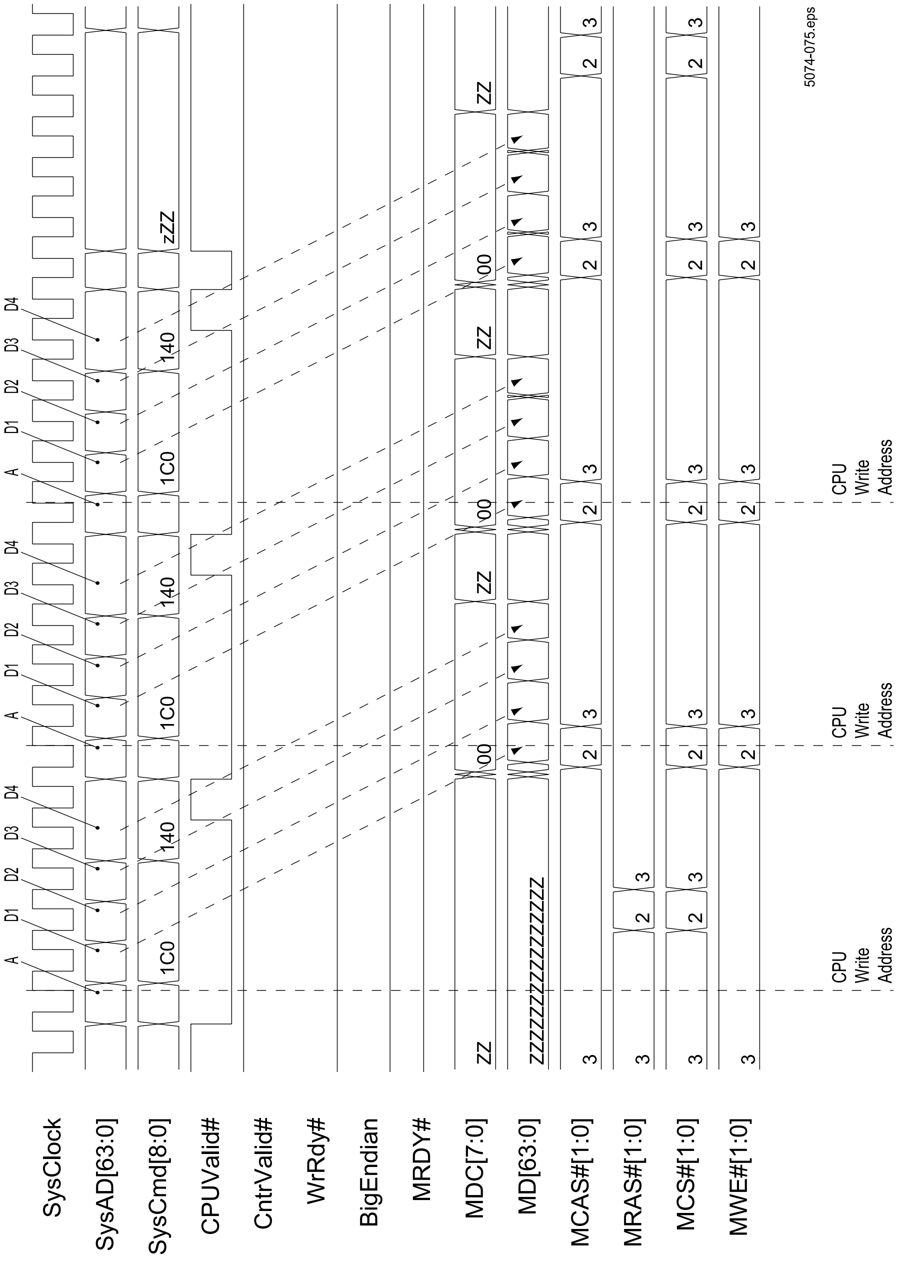


Figure 32: Back-To-Back Block Memory Writes



14.2

PCI-Bus Accesses

Figure 33 through Figure 39 show the timing for various transactions on the PCI Bus, including:

- ❑ Controller as PCI-Bus Master
  - PCI Memory Write/Read (Figure 33)
  - PCI Memory Byte Writes, With Byte-Merging (Figure 34)
  - PCI Memory Byte Read, With Prefetching (Figure 35)
  - PCI Memory Eight-Byte Writes, With Combining (Figure 36)
  - PCI Memory Dual Address Cycle (DAC) Write/Read (Figure 37)
- ❑ Controller as PCI-Bus Target
  - PCI-Bus Master Read/Write to Controller Memory (Figure 38)
  - PCI-Bus Master Read/Write to Controller’s Internal Registers (Figure 39)

All timing examples use a 33 MHz PCI-Bus clock and Medium target DEVSEL.

Figure 33: PCI Memory Write/Read

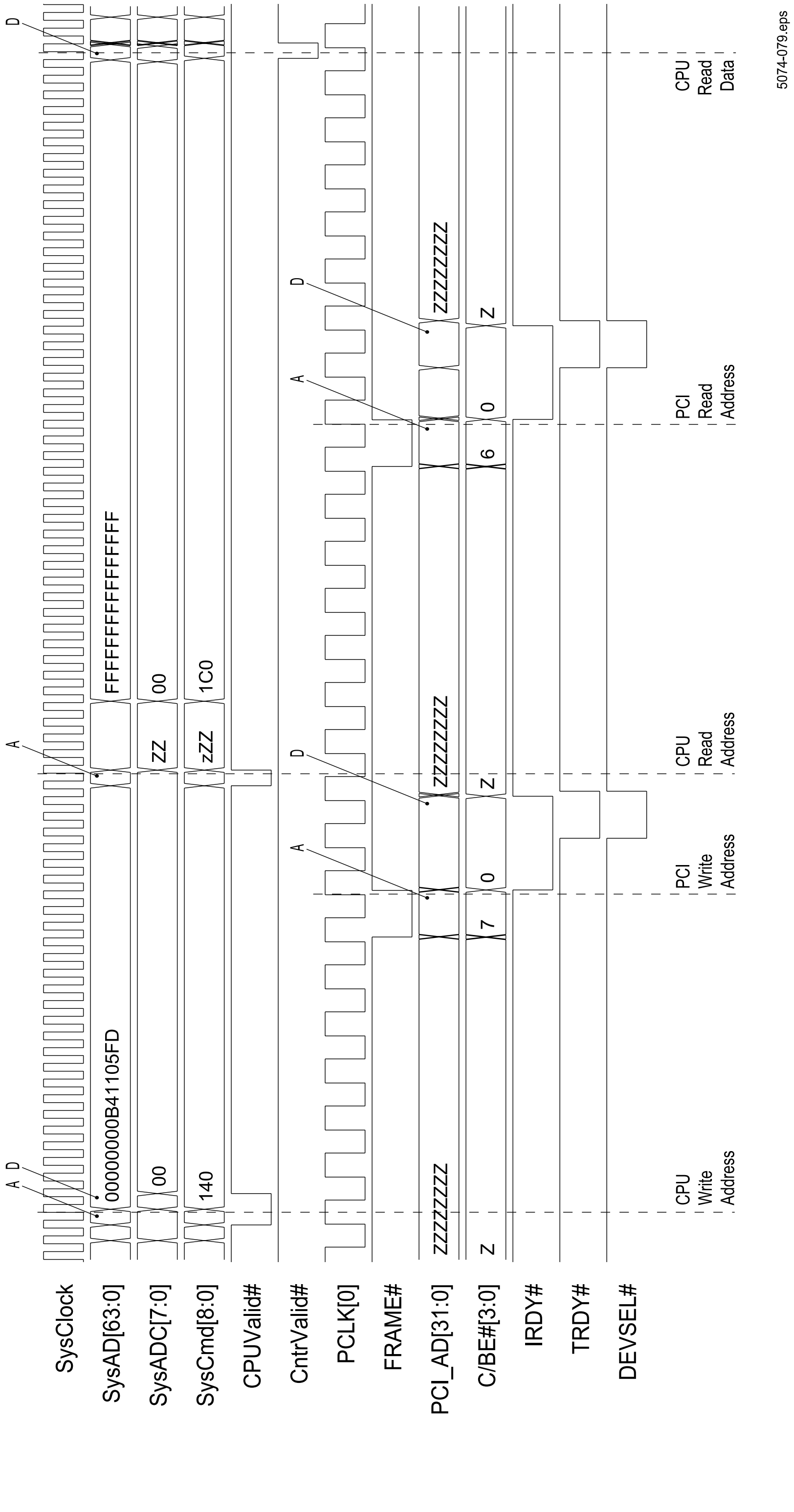


Figure 34: PCI Memory Byte Writes, With Byte-Merging

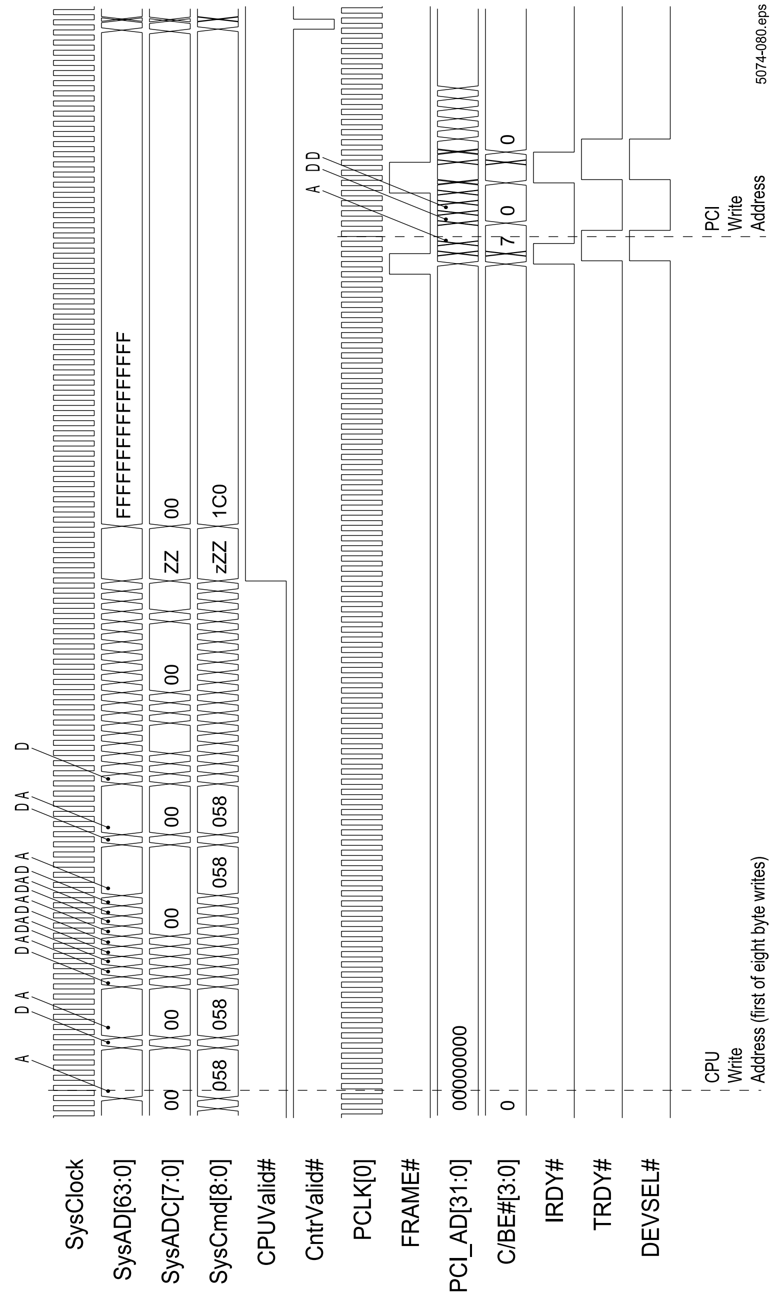


Figure 35: PCI Memory Byte Read, With Prefetching

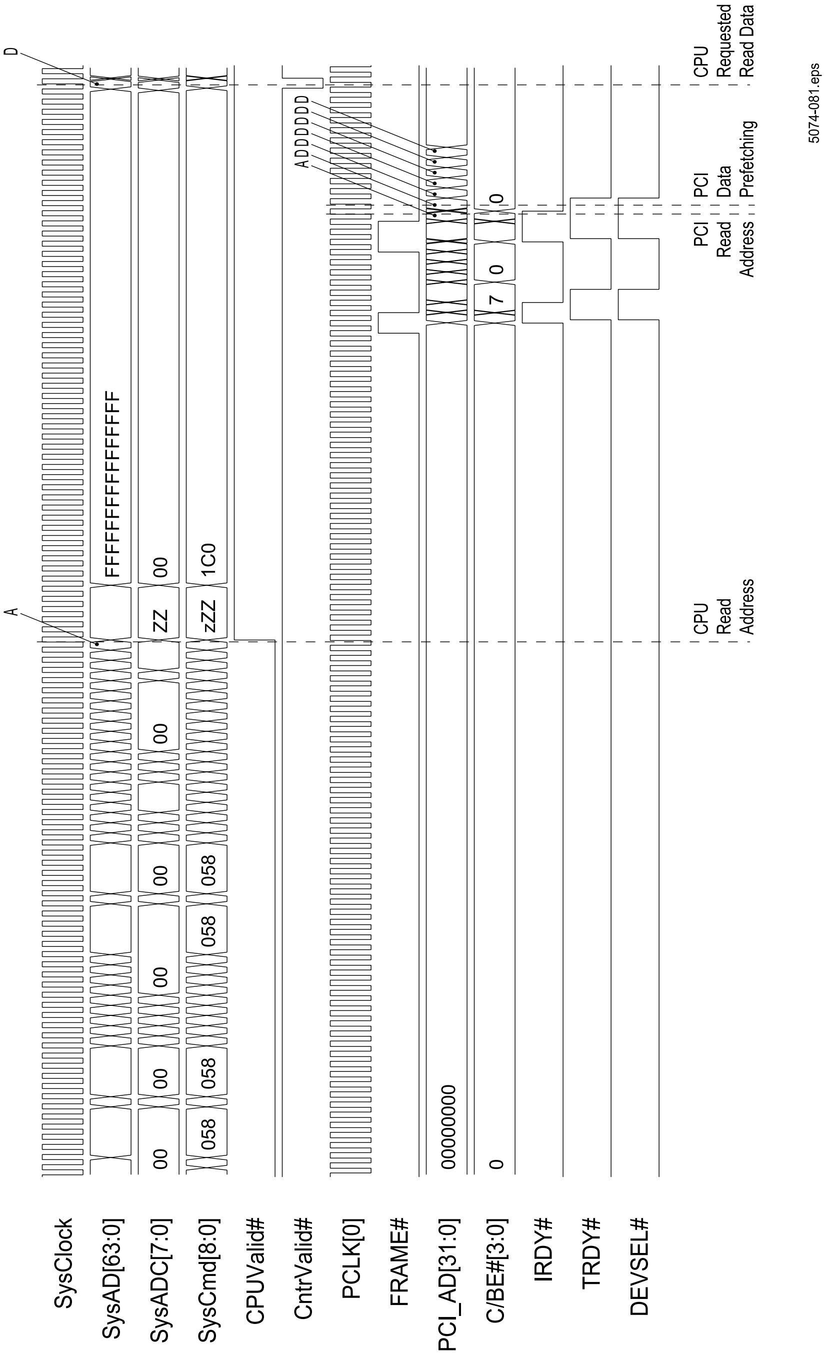


Figure 36: PCI Memory Eight-Byte Writes, With Combining

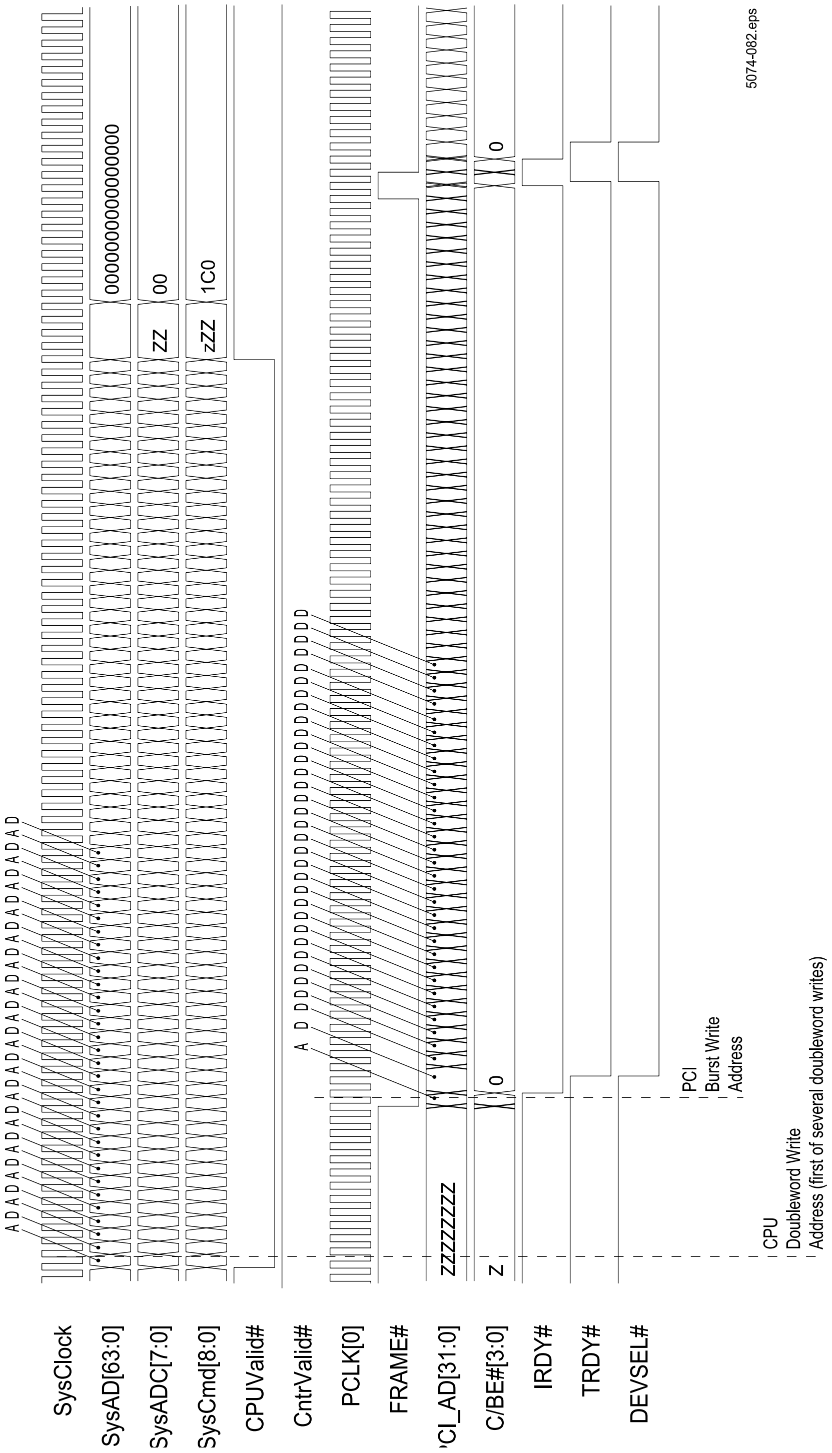


Figure 37: PCI Memory Dual Address Cycle (DAC) Write/Read

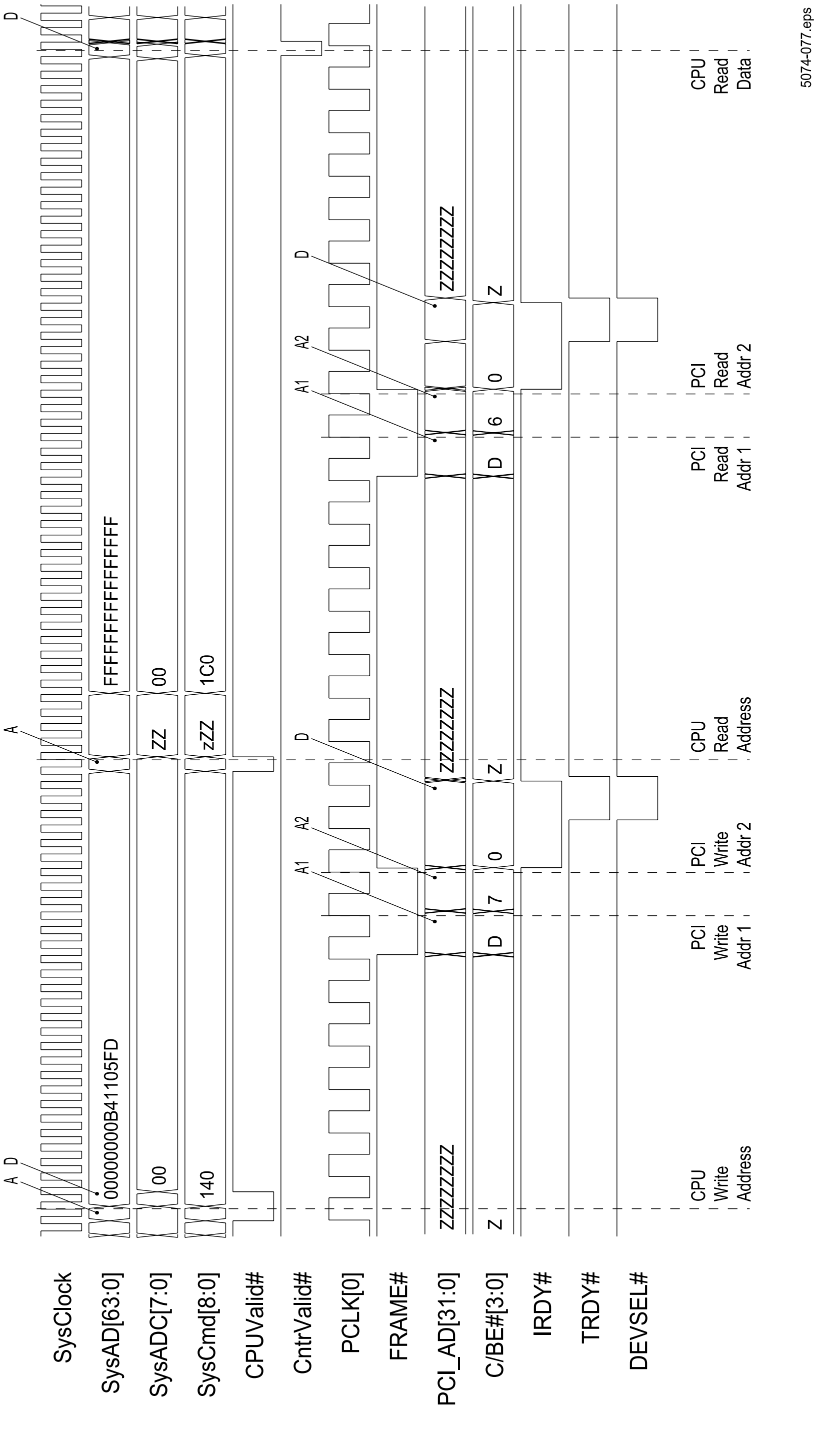


Figure 38: PCI-Bus Master Read/Write to Controller Memory

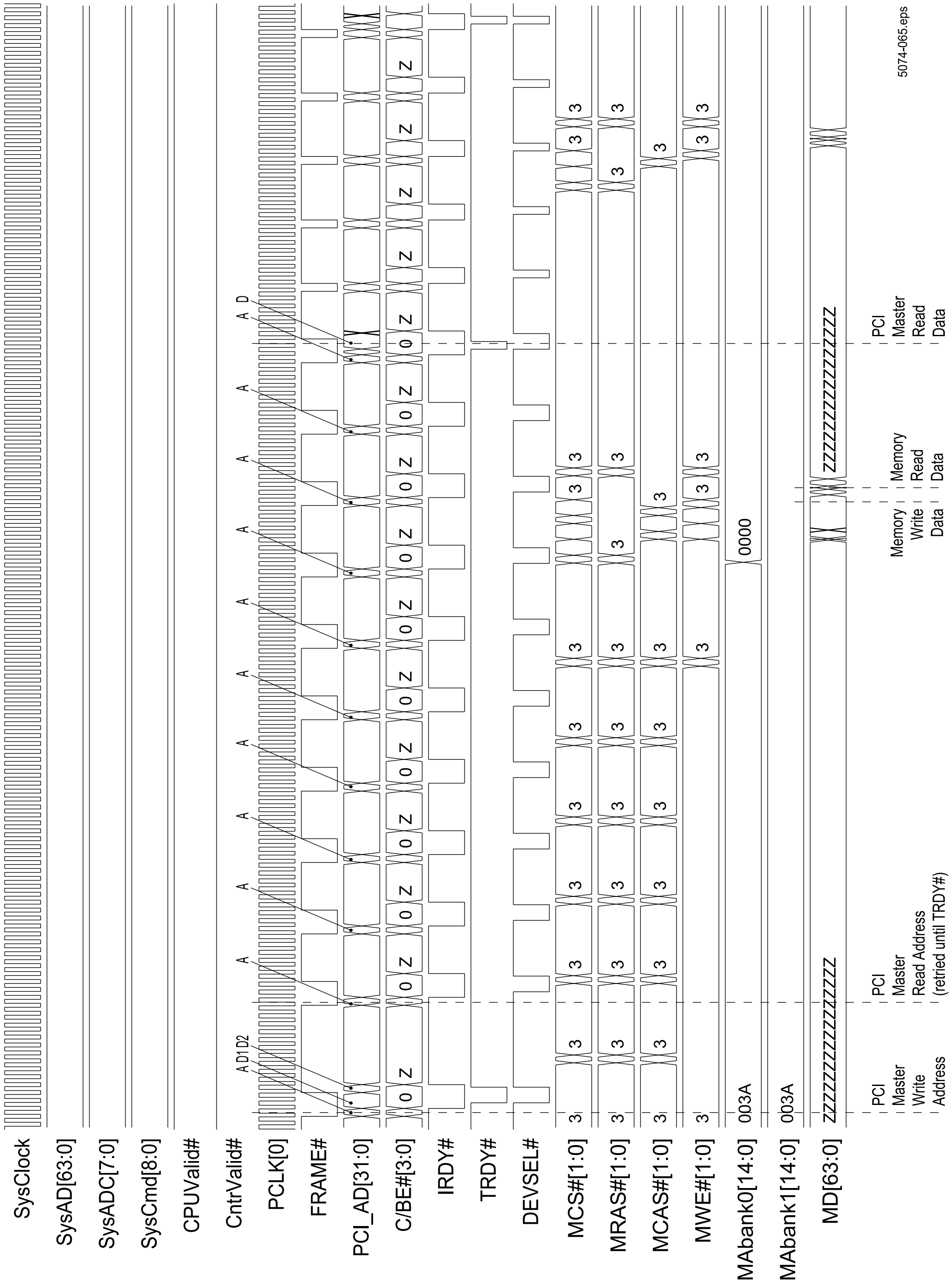
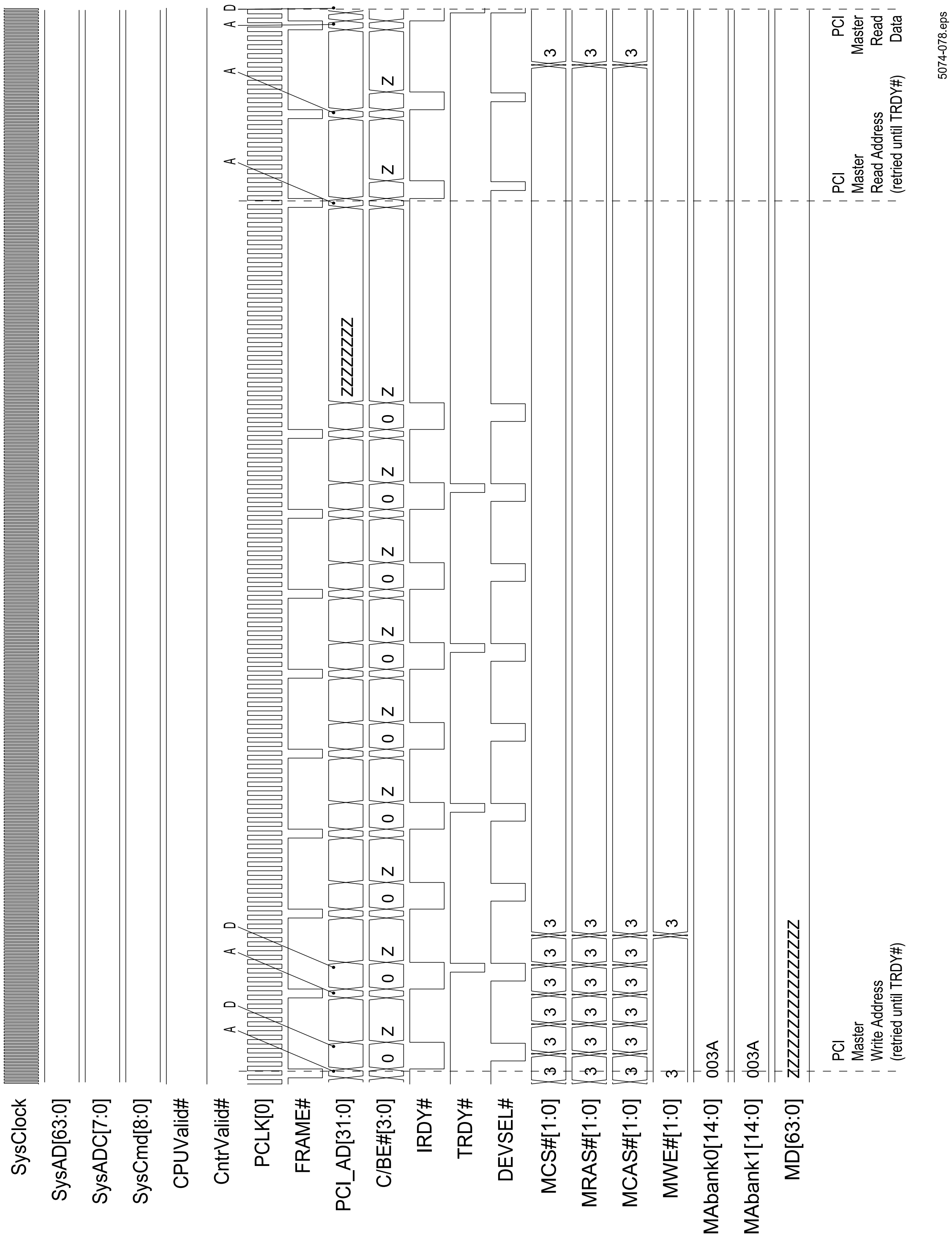


Figure 39: PCI-Bus Master Read/Write to Controller’s Internal Registers



14.3

Local-Bus Accesses

Figure 40 through Figure 47 show the timing for various transactions on the Local Bus, including:

- ❑ Controller as Local-Bus Master
  - CPU Byte Write/Read to 8-Bit Local-Bus Target (Figure 40)
  - CPU Four-Byte Write/Read to 8-Bit Local-Bus Target (Figure 41)
  - CPU Eight-Byte Write/Read to 8-Bit Local-Bus Target (Figure 42)
  - CPU Burst Write/Read to 32-Bit Local-Bus Target (Figure 43)
- ❑ Controller as Local-Bus Target
  - Local-Bus Master Four-Byte Write/Read to Controller Memory, 68000 Mode (Figure 44)
  - Local-Bus Master Burst Write/Read to Controller Memory, 68000 Mode (Figure 45)
  - Local-Bus Master Four-Byte Write/Read to Controller Memory, Intel Mode (Figure 46)
  - Local-Bus Master Burst Write/Read to Controller Memory, Intel Mode (Figure 47)

Figure 40: CPU Byte Write/Read to 8-Bit Local-Bus Target

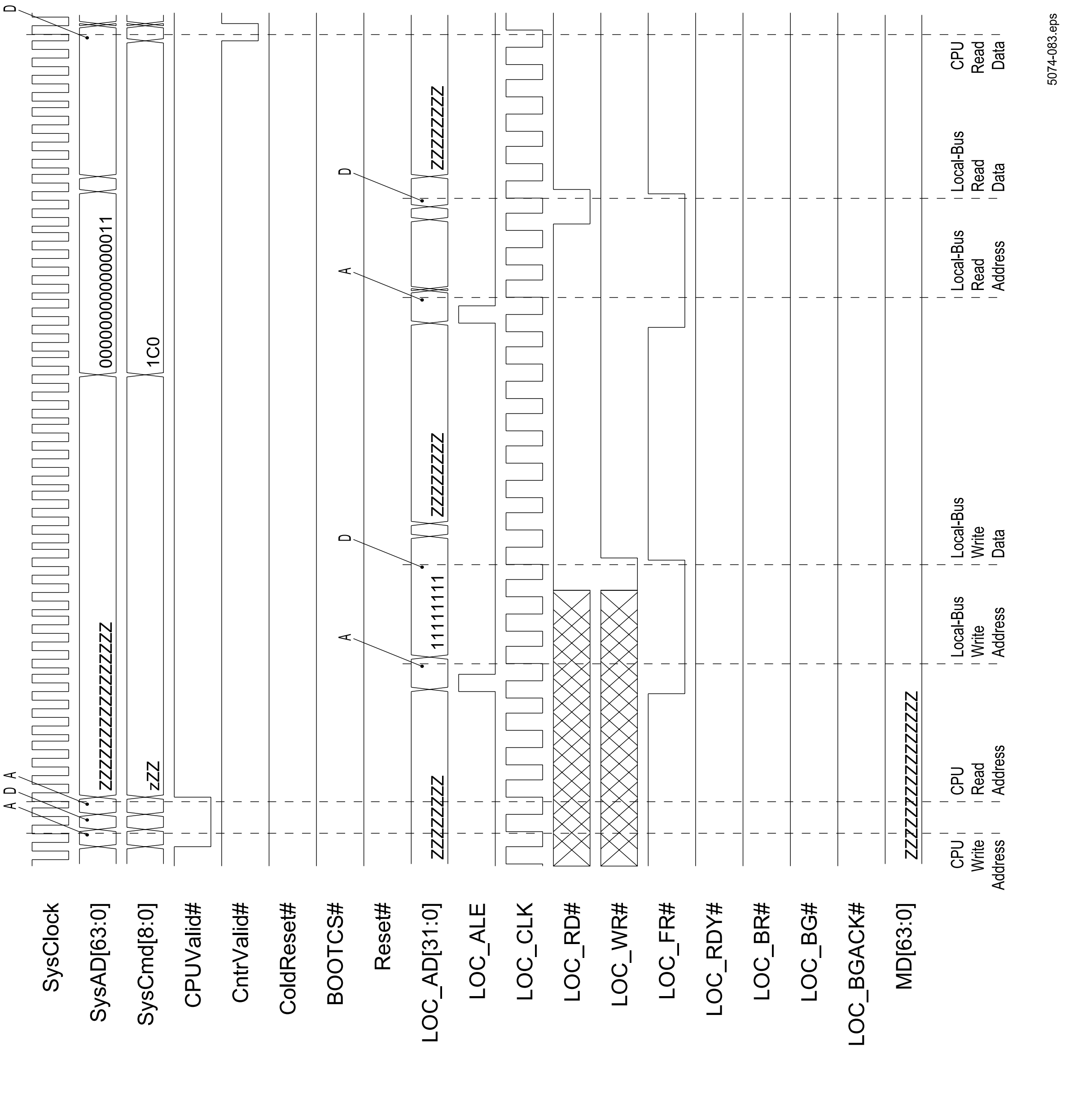


Figure 41: CPU Four-Byte Write/Read to 8-Bit Local-Bus Target

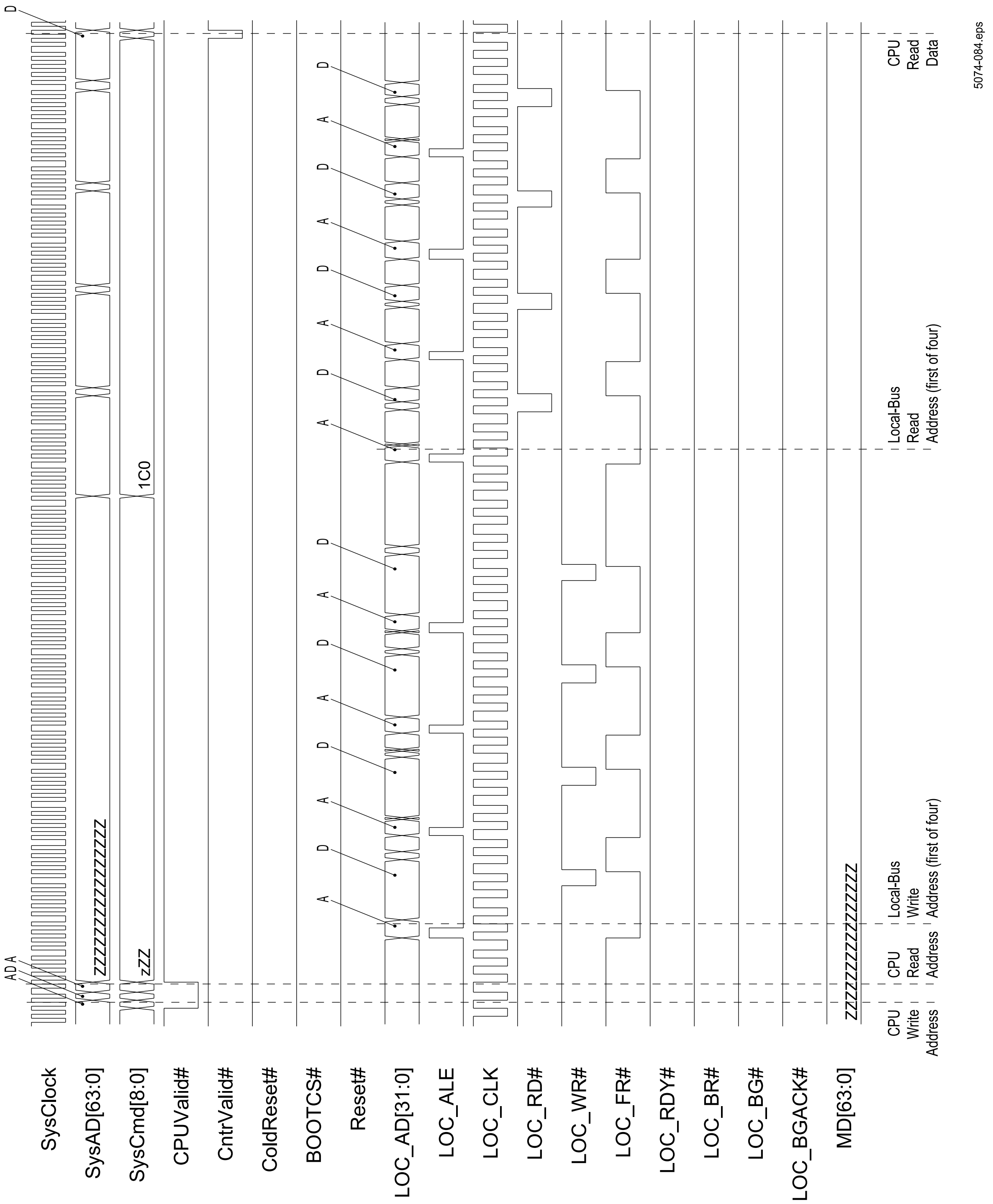


Figure 42: CPU Eight-Byte Write/Read to 8-Bit Local-Bus Target

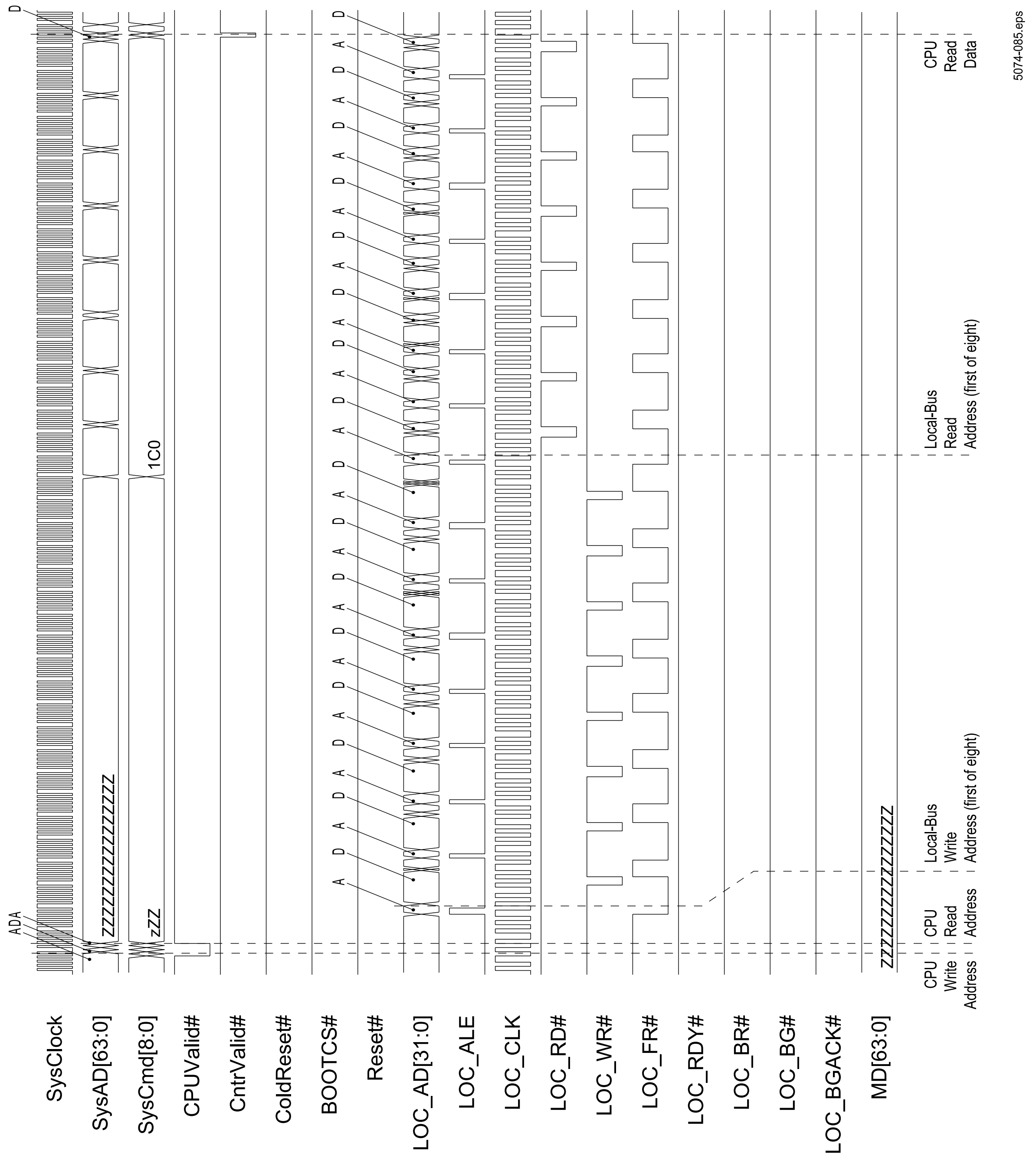


Figure 43: CPU Burst Write/Read to 32-Bit Local-Bus Target

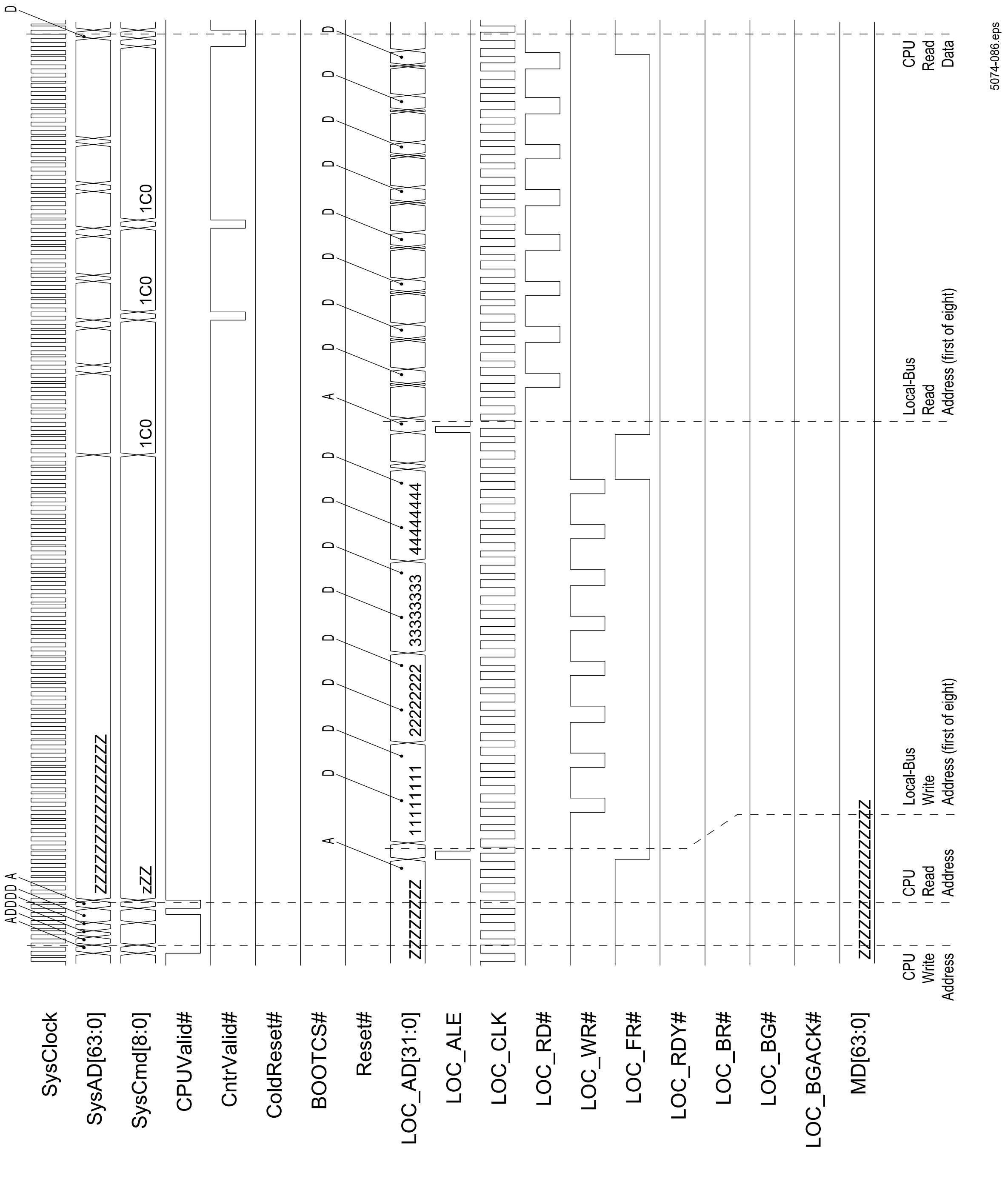


Figure 44: Local-Bus Master Four-Byte Write/Read to Controller Memory, 68000 Mode

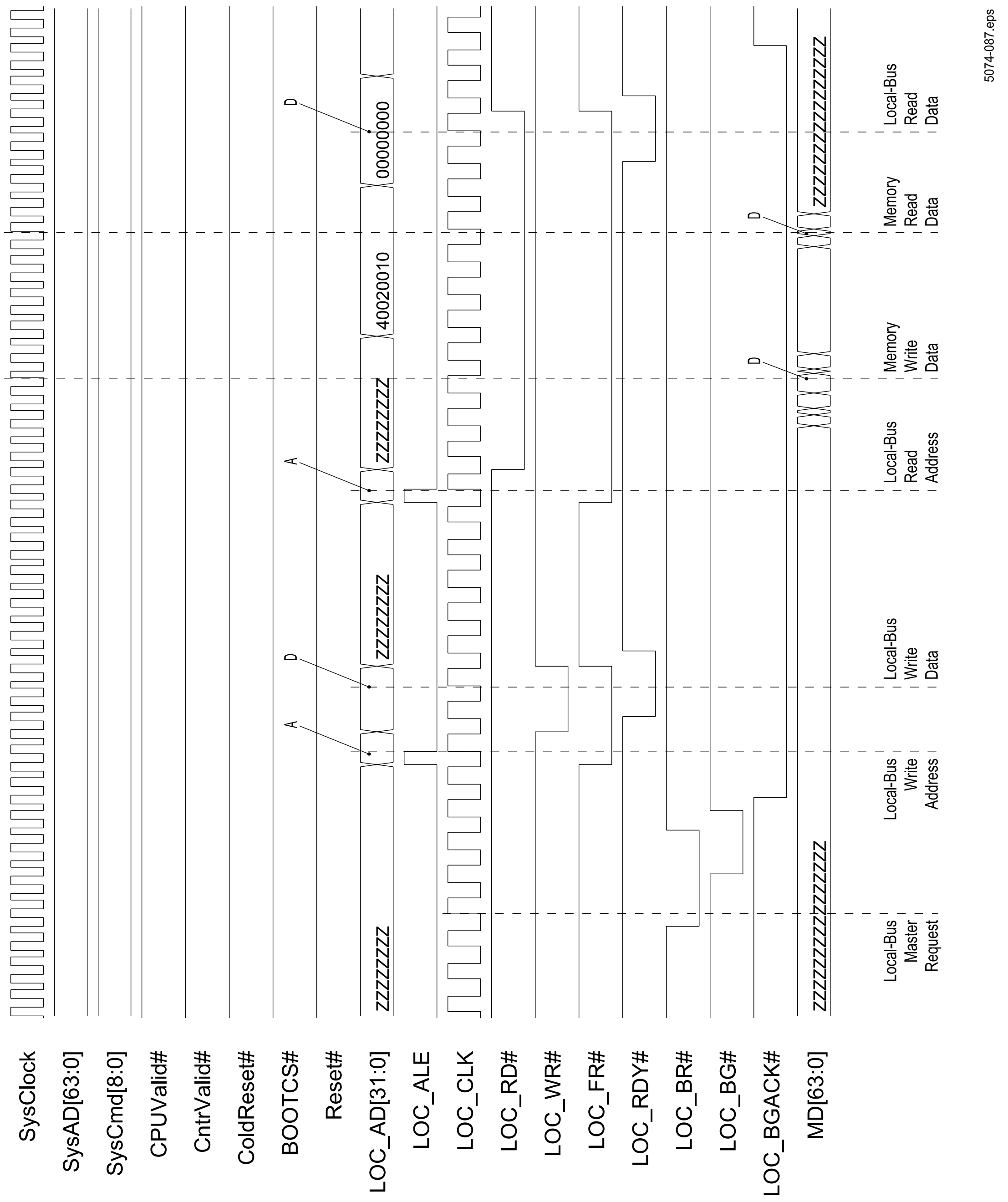


Figure 45: Local-Bus Master Burst Write/Read to Controller Memory, 68000 Mode

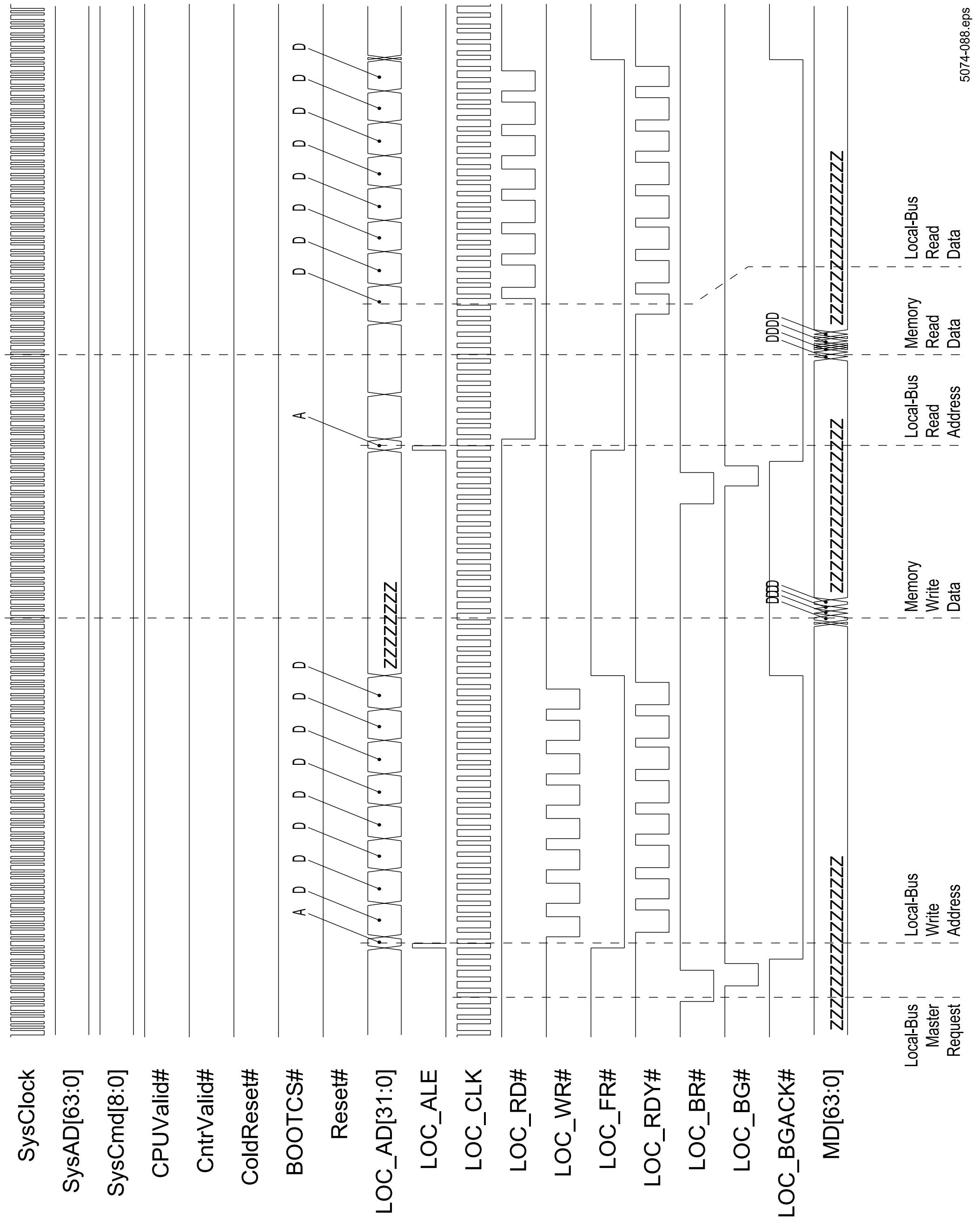


Figure 46: Local-Bus Master Four-Byte Write/Read to Controller Memory, Intel Mode

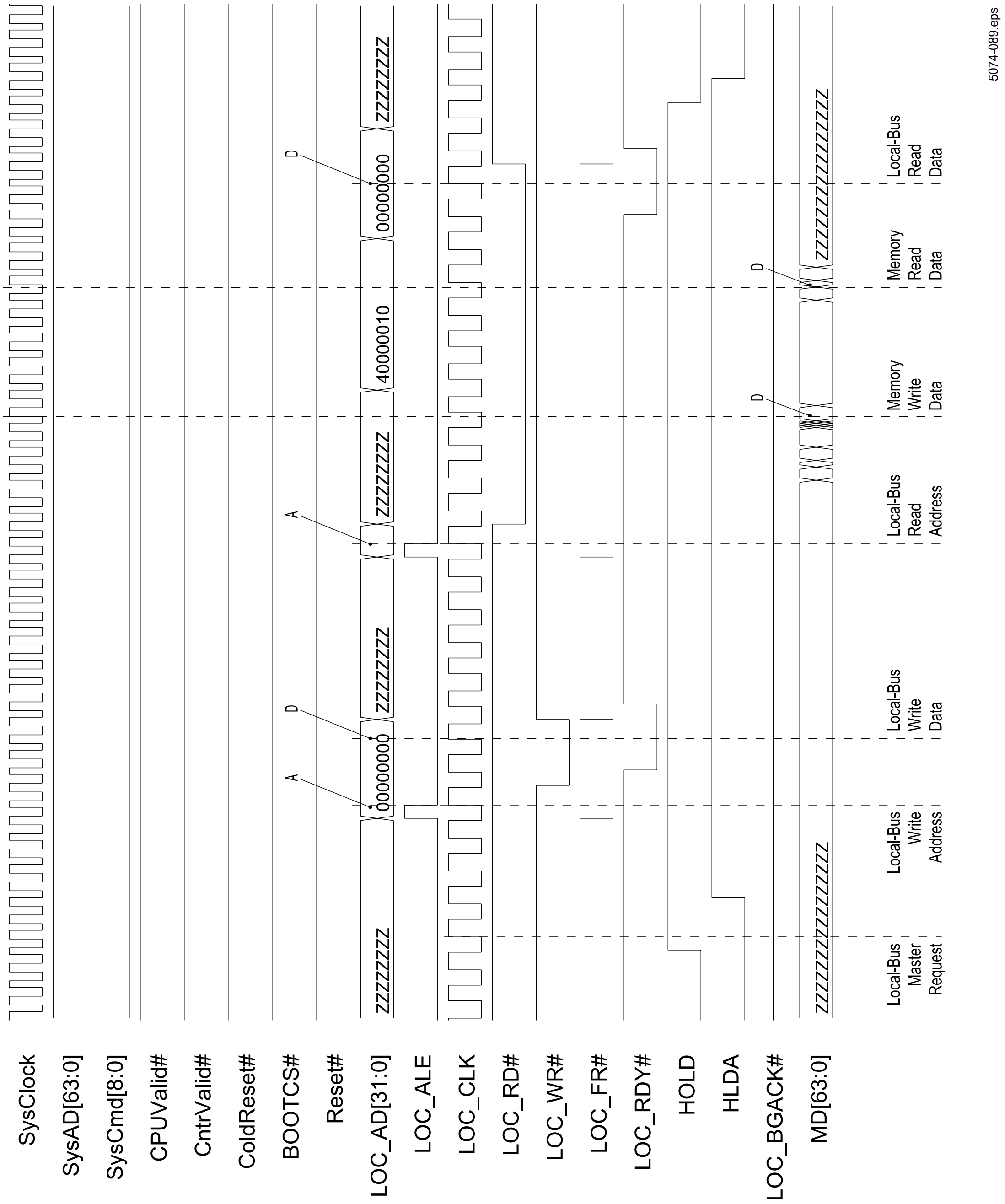
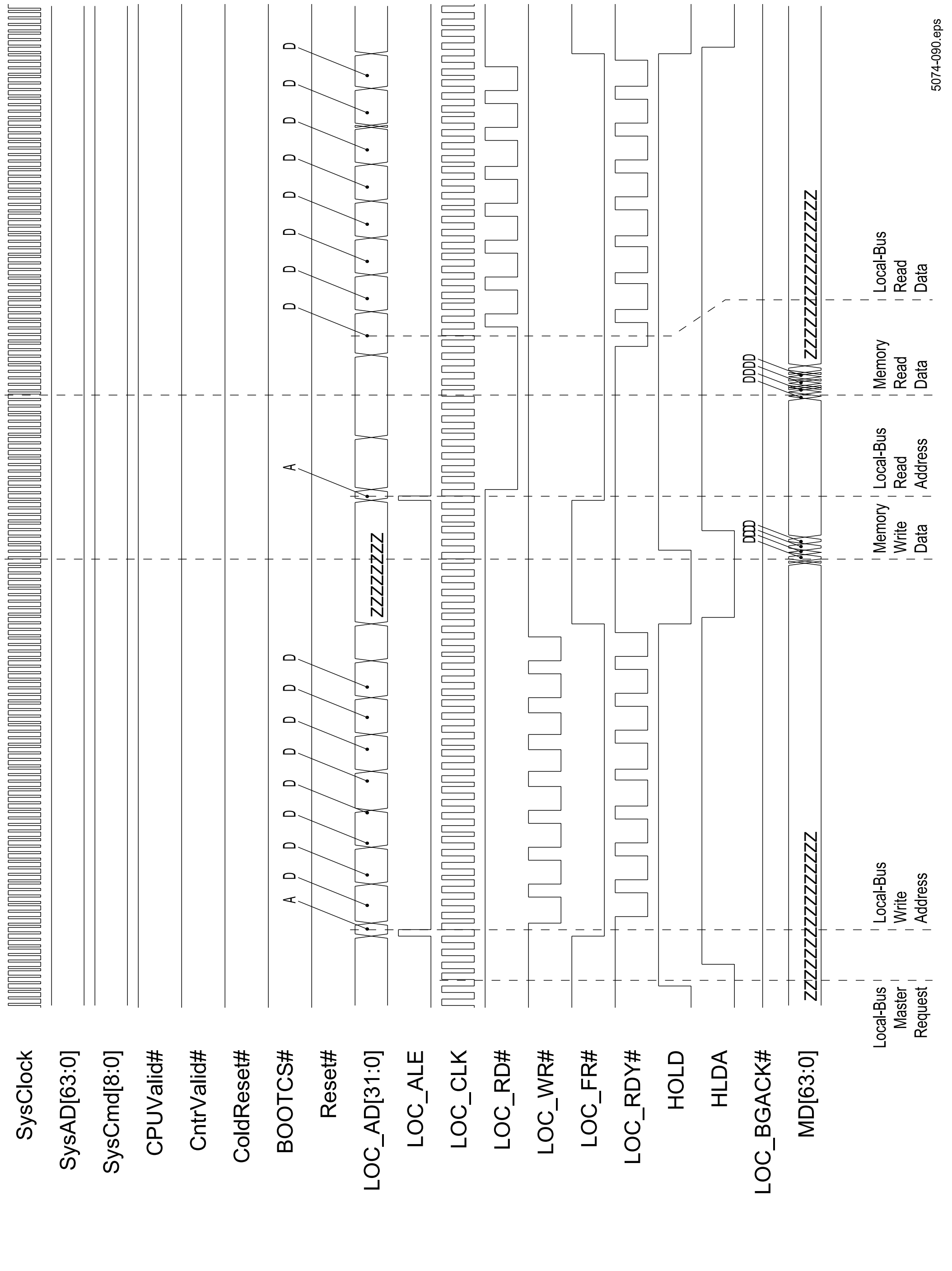


Figure 47: Local-Bus Master Burst Write/Read to Controller Memory, Intel Mode



15.0 Testing

The controller does not support JTAG testing or any other type of boundary scan. It does, however, support board-level testing. Table 37 shows the board-level test modes that can be configured with the TEST#, SMC and TEST_SEL inputs.

Table 37: Test-Mode Configuration

TEST#	SMC	TEST_SEL	Description
0	0	0	All Outputs Tri-State
0	0	1	*unused, reserved*
0	1	0	*unused, reserved*
0	1	1	*unused, reserved*
1	0	0	Normal Operation
1	0	1	Normal Operation, with PLL by-passed
1	1	0	*unused, reserved*
1	1	1	Wiggle Mode

In the *Wiggle Mode*, the BigEndian signal becomes an output and all other signals are inputs. BigEndian is driven by the XOR of all other signals. This mode can be used by a board tester to verify connectivity to all controller signals.

16.0 Electrical Specifications

16.1 Terminology

Table 38: Terminology for Absolute Maximum Ratings

Item	Symbol	Meaning
Power supply voltage	V_{DD}	Range of voltages which will not cause destruction or reduce reliability when applied to the VDD pin.
Input voltage	V_I	Range of voltages which will not cause destruction or reduce reliability when applied to the input pin.
Output voltage	V_O	Range of voltages which will not cause destruction or reduce reliability when applied to the output pin.
Input current	I_I	Allowable absolute value of current which will not cause latchup when applied to the input pin.
Output current	I_O	Allowable absolute value of DC current which will not cause destruction or reduce reliability when flowing to or from the output pin.
Operating ambient temperature	T_A	Range of ambient temperatures for normal logical operation.
Storage temperature	T_{stg}	Range of element temperatures which will not cause destruction or reduce reliability in the state where neither voltage nor current is applied.

Table 39: Terminology for Recommended Operating Conditions

Item	Symbol	Meaning
Power supply voltage	V_{DD}	Range of a voltage for normal logical operation when V_{SS} = 0 V.
Input voltage, high	V_{IH}	Indicates a high-level voltage applied to the cell-based IC input that allows normal operation of the input buffer. Applying a voltage of the Min. value or above ensures that the input voltage is at high level.
Input voltage, low	V_{IL}	Indicates a low-level voltage applied to the cell-based IC input that allows normal operation of the input buffer. Applying a voltage of the Max. value or below ensures that the input voltage is at low level.
Positive trigger voltage	V_P	Refers to the input level at which the output level is inverted when the cell-based IC input is changed from low- level to high-level.
Negative trigger voltage	V_N	Refers to the input level at which the output level is inverted when the cell-based IC input is changed from high- level to low-level.
Hysteresis voltage	V_H	Refers to the difference between the positive trigger voltage and negative trigger voltage.
Input rise time	t_{ri}	Indicates the limit value of the time in which the input voltage applied to the cell-based IC rises from 10% to 90%.
Input fall time	t_{fi}	Indicates the limit value of the time in which the input voltage applied to the cell-based IC input falls from 90% to 10%.

Table 40: Terminology for DC Characteristics

Item	Symbol	Meaning
Static current consumption	I_{DDS}	Indicates the current that flows in from the power supply pin at a specified supply voltage without changing the voltage of the input and output pins.
Off-state output current	I_{OZ}	For a 3-state output, this value indicates the current that flows through the output pin at specified voltage when the output is at high impedance.
Output short-circuit current	I_{OS}	Current that flows out when the output pin is short-circuited to GND, when output is at high level.

Table 40: Terminology for DC Characteristics (continued)

Item	Symbol	Meaning
Input leakage current	I_I	Current that flows through the input pin when a voltage is applied to the input pin.
Output current, low	I_{OL}	Current that flows to the output pin at a specified low-level output voltage.
Output current, high	I_{OH}	Current that flows from the output pin at a specified high- level output voltage.
Output voltage, low	V_{OL}	Indicates a low-level output voltage when output is open.
Output voltage, high	V_{OH}	Indicates a high-level output voltage when output pin is open.

16.2 Absolute Maximum Ratings

Table 41: Absolute Maximum Ratings

Item	Symbol	Conditions	Ratings	Unit
Power supply voltage	V_{DD}		–0.5 to +4.6	V
Input voltage [a]	V_I	$V_I < V_{DD} + 0.5$ V	–0.5 to +4.6	V
Output voltage	V_O	$V_O < V_{DD}+ 0.5$ V	–0.5 to +4.6	V
Output current:	I_O			
$I_{OL} = 1.0$ mA		FV0A	3	mA
$I_{OL} = 2.0$ mA		FV0B	7	mA
$I_{OL} = 3.0$ mA		FO09, FV09	10	mA
$I_{OL} = 6.0$ mA		FO04, FV04	20	mA
$I_{OL} = 9.0$ mA		FV01, FV01	30	mA
$I_{OL} = 12.0$ mA		FO02, FV02	40	mA
$I_{OL} = 18.0$ mA		FO03	60	mA
$I_{OL} = 24.0$ mA		FO06	75	mA
Operating ambient temperature	T_A		–40 to +85	°C
Storage temperature	T_{stg}		–65 to +150	°C

[a] Apply voltage to the input pins only after the power supply voltage has been applied.

16.3 Recommended Operating Range

Table 42: Recommended Operating Range

Item	Symbol	Conditions	Min.	Typical	Max.	Unit
Power supply voltage	V_{DD}		3.0	3.3	3.6	V
Input voltage, high	V_{IH}	3V interface	2.0		V_{DD}	V
Input voltage, low	V_{IL}		0		0.8	V
Positive trigger voltage	V_P		1.50		2.70	V
Negative trigger voltage	V_N		0.60		1.40	V
Hysteresis voltage	V_H		1.10		1.50	V
Input voltage, high	V_{IH}	5V interface	2.0		5.5	V
Input voltage, low	V_{IL}		0		0.8	V
Positive trigger voltage	V_P		2.20		2.55	V
Negative trigger voltage	V_N		0.84		1.01	V
Hysteresis voltage	V_H		1.36		1.54	V

Table 42: Recommended Operating Range (continued)

Item	Symbol	Conditions	Min.	Typical	Max.	Unit
Input rise time	t_{ri}	Normal input	0		200	ns
Input fall time	t_{fi}		0		200	ns
Input rise time	t_{ri}	Schmitt input [a]	0		10	ms
Input fall time	t_{fi}		0		10	ms

a. Use a Schmitt trigger input buffer for input signals with very slow rise or fall times.

16.4

DC Characteristics

The "+" and "–" next to the current value in the table indicate the current direction. Current flowing into the device is "+" and current flowing out of the device is "–". Structurally, the CMOS 5V output buffer has no DC output High level.

Table 43: DC Characteristics

VDD = 3.3V ± 0.3V; T_A = –40 to +85°C; T_j = –40 to +125°C

Item	Symbol	Conditions	Min.	Typical	Max.	Unit
Static current consumption: [a]						
H49-M97	I_{DDS}	$V_I = V_{DD}$ or GND	—	40	800	µA
E80-H10	I_{DDS}		—	20	400	µA
Step sizes other than the above	I_{DDS}		—	10	200	µA
OFF-state output current	I_{OZ}	$V_O = V_{DD}$ or GND	—	—	±10	µA
Output short-circuit current [b]	I_{OS}	V_O = GND	—	—	–250	mA
Input leakage current:						
Normal input	I_I	$V_I = V_{DD}$ or GND	—	$\pm 10^{-4}$	±10	µA
With pull-up resistor (50 kΩ)	I_I	V_I = GND	36	89	165	µA
With pull-up resistor (5 kΩ)	I_I	V_I = GND	284	654	1305	µA
With pull-down resistor (50 kΩ)	I_I	$V_I = V_{DD}$	28	79	141	µA
Pull-up resistor (50 kΩ)	R_{PU}	—	21.8	37.1	83.1	kΩ
Pull-up resistor (5 kΩ)	R_{PU}	—	2.8	5.0	10.6	kΩ
Pull-down resistor (50 kΩ)	R_{PD}	—	25.6	41.9	105.8	kΩ
Output current low:						
3.0 mA type FO09	I_{OL}	V_{OL}= 0.4 V	3.00	—	—	mA
6.0 mA type FO04	I_{OL}	V_{OL}= 0.4 V	6.00	—	—	mA
9.0 mA type FO01	I_{OL}	V_{OL}= 0.4 V	9.00	—	—	mA
12.0 mA type FO02	I_{OL}	V_{OL}= 0.4 V	12.00	—	—	mA
18.0 mA type FO03	I_{OL}	V_{OL}= 0.4 V	18.00	—	—	mA
24.0 mA type FO06	I_{OL}	V_{OL}= 0.4 V	24.00	—	—	mA
Output current high:						
3.0 mA type FO09	I_{OH}	V_{OH} = 2.4 V	–3.00	—	—	mA
6.0 mA type FO04	I_{OH}	V_{OH} = 2.4 V	–6.00	—	—	mA
9.0 mA type FO01	I_{OH}	V_{OH} = 2.4 V	–9.00	—	—	mA
12.0 mA type FO02	I_{OH}	V_{OH} = 2.4 V	–12.00	—	—	mA
18.0 mA type FO03	I_{OH}	V_{OH} = 2.4 V	–18.00	—	—	mA

Table 43: DC Characteristics (continued)

VDD = 3.3V ± 0.3V; T_A = –40 to +85°C; T_j = –40 to +125°C

Item	Symbol	Conditions	Min.	Typical	Max.	Unit
24.0 mA type FO06	I_{OH}	V_{OH} = 2.4 V	–24.00	—	—	mA
Output voltage low	V_{OL}	I_{OL} = 0 mA	—	—	0.1	V
Output voltage high	V_{OH}	I_{OH} = 0 mA	V_{DD} - 0.1	—	—	V

a. The static current consumption increases if an I/O block with a pull-up or pull-down resistor is used.

b. Output short-circuit current is 1 second or less and only 1 pin of the chip.

16.5 AC Specifications

16.5.1 Clock Timing

Table 44 shows the timing requirements for SysClock with and without L2 cache, and for PCLK[0] with an internal and an external arbiter.

Table 44: SysClock and PCLK[0] Timing Requirements

Signal	Min. Period	Min. Low	Max. High	Units	Notes
SysClock	11.6	5.0	5.0	ns	Without L2 cache
SysClock	13.5	5.0	5.0	ns	With L2 cache
PCLK[0]	15.7	12.0	5.0	ns	With external PCI arbiter
PCLK[0]	15.7	5.0	5.0	ns	With internal PCI arbiter

16.5.2 CPU, Memory, Local Bus and Interrupt Signals

Table 45 shows the timing requirements, relative to SysClock, for the signals on the CPU Bus, Memory Bus, Local-Bus, and the interrupt signals (both CPU and PCI interrupt signals). Figure 48 defines the setup, hold, and valid parameters.

Figure 48: AC Timing Waveforms

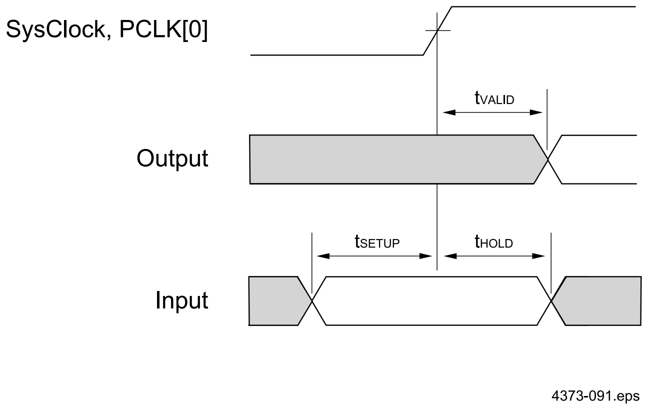


Table 45: Signal Timing Relative to SysClock

Signal	Output Min. Valid	Output Max. Valid	Output Pin Load	Input Min. Setup	Input Min. Hold	Units
BigEndian	7.5	17.8	50	—	—	ns
BootCS#	2.8	9.5	50	—	—	ns
CntrValid#	3.9	9.6	50	1.5	0.0	ns
CntrVccOk	5.9	13.2	50	—	—	ns
ColdReset#	5.7	13.4	50	—	—	ns
CPUValid#	—	—	—	4.5	0.0	ns
DCS#[8:2]	2.8	10.4	50	1.0	0.2	ns
DQM	2.8	6.7	80	—	—	ns
INTA#	2.2	7.8	50	1.3	0.0	ns
INTB#	—	—	—	1.3	0.0	ns
INTC#	—	—	—	1.3	0.0	ns
INTD#	—	—	—	1.3	0.0	ns
INTE#	—	—	—	1.2	0.0	ns
Int#[5:0]	4.2	11.2	50	—	—	ns
LOC_A[4:0]	2.3	8.4	50	5.2	0.0	ns
LOC_AD[31:0]	2.3	12.6	50	4.2	0.0	ns
LOC_ALE	2.3	14.2	50	4.4	0.0	ns
LOC_BG# or **HLDA**	3.4	9.5	50	—	—	ns
LOC_BGACK#	—	—	—	3.0	0.0	ns
LOC_BR# or **HOLD**	—	—	—	3.3	0.0	ns
LOC_CLK	2.2	5.6	50	—	—	ns
LOC_FR#	3.0	8.3	50	5.0	0.0	ns
LOC_RD#	3.0	8.3	50	3.5	0.0	ns
LOC_RDY#	2.8	8.0	50	10.7	0.0	ns
LOC_WR#	3.0	8.3	50	5.1	0.0	ns
MAbank0[14:0]	2.4	8.6	80	—	—	ns
MAbank1[14:0]	2.5	7.8	80	—	—	ns
MCAS#[1:0]	3.5	7.2	80	—	—	ns
MCS#[1:0]	3.5	7.3	80	—	—	ns
MCWrRdy#	3.6	8.3	50	—	—	ns
MD[63:0]	2.0	8.1	40	1.0	0.5	ns
MDC[7:0]	2.8	9.5	40	1.2	0.1	ns
ModeClock	—	—	—	1.0	1.6	ns
ModeOut	4.6	11.8	50	—	—	ns
MRAS#[1:0]	3.5	7.5	80	—	—	ns
MRDY#	—	—	—	3.0	0.0	ns
MWE#[1:0]	3.5	7.3	80	—	—	ns
NMI#	4.4	10.8	50	—	—	ns
PROM_CLK	5.2	11.9	50	—	—	ns
PROM_SD	4.8	10.2	40	1.0	1.2	ns
Reset#	5.9	13.4	50	—	—	ns

Table 45: Signal Timing Relative to SysClock (continued)

Signal	**Output Min. Valid**	**Output Max. Valid**	**Output Pin Load**	**Input Min. Setup**	**Input Min. Hold**	**Units**
ScDOE#	3.3	9.1	50	—	—	ns
ScMatch	—	—	—	1.0	0.2	ns
ScWord[1:0]	2.3	9.1	50	1.0	0.1	ns
SysAD[63:0]	2.5	9.3	50	2.0	0.1	ns
SysADC[7:0]	1.9	9.5	50	1.3	0.1	ns
SysClock	See Table 44					ns
SysCmd[8:0]	3.3	9.9	50	3.6	0.3	ns
UART_DSR#						ns
UART_DTR#	5.0	11.8	50	1.0	1.1	ns
UART_RxDRDY#						ns
UART_TxDRDY#	5.1	11.9	50	1.0	1.1	ns
VccOk	—	—	—	1.9	0.0	ns
WrRdy#	3.3	8.3	50	1.0	0.2	ns

16.5.3

PCI-Bus Interface

Table 46 shows the timing requirements, relative to PCLK[0], for signals on the PCI Bus, except for the PCI interrupt signals which are included in Table 45.

Table 46: Signal Timing Relative to PCLK[0]

Signal	**Output Min. Valid**	**Output Max. Valid**	**Output Pin Load**	**Input Min. Setup**	**Input Min. Hold**	**Units**
ACK64#	5.1	10.3	50	5.8	2.2	ns
C/BE#[7:0]	5.0	15.0	50	7.3	2.6	ns
DEVSEL#	5.0	9.9	50	8.4	2.4	ns
FRAME#	5.5	13.8	50	8.6	2.5	ns
GNT#[0]	5.2	9.3	50	10.6	2.5	ns
GNT#[4:1]	5.1	9.3	50	—	—	ns
IDSEL	—	—	—	1.0	2.2	ns
INT[E:A]#	See Table 45					ns
IRDY#	5.2	12.5	50	8.6	2.3	ns
LOCK#	TBD	TBD	TBD	TBD	TBD	ns
M66EN	Static signal. Tie High or Low per Table 3					ns
PCI_AD[63:0]	5.1	13.0	50	1.0	2.9	ns
PAR	5.0	9.2	50	1.0	2.3	ns
PAR64	5.0	9.4	50	1.4	2.0	ns
PCI64#	Static signal. Tie High or Low per Table 3					ns
PCICR#	Static signal. Tie High or Low per Table 3					ns
PCIRST#	5.9	11.8	50	2.1	2.5	ns
PCLK[0]	See Table 44					ns
PERR#	4.8	9.0	50	1.0	2.4	ns
REQ#[0]	5.4	10.2	50	1.0	2.6	ns
REQ#[4:1]	—	—	—	1.3	2.8	ns
REQ64#	5.3	15.1	50	1.0	2.2	ns
SERR#	4.9	9.2	50	1.0	2.2	ns
STOP#	5.1	10.4	50	8.1	2.4	ns
TRDY#	5.0	10.5	50	8.5	2.3	ns

17.0 Pinout

The controller is packaged in a 500-pin TBGA package. Table 47 shows the pin assignments, sorted by signal names (left side), pin number (middle), and grid number (right side). Figure 49 on page 210 shows the package diagram.

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number

Signal Name	Alternate Signal	Pin #	Grid #
AGND		170	AJ28
AGND		345	AD4
AVDD		247	AE3
AVDD		458	AF25
BigEndian		171	AJ29
BootCS#		160	AJ18
C/BE#0		339	V4
C/BE#1		427	R5
C/BE#2		334	N4
C/BE#3		9	J1
CntrValid#		223	B4
CntrVccOk		324	C4
ColdReset#		500	E6
CPUValid#		224	B3
DCS#2	see Table 4 on page 27	156	AJ14
DCS#3	see Table 4 on page 27	261	AH14
DCS#4	see Table 4 on page 27	42	AK13
DCS#5	see Table 4 on page 27	155	AJ13
DCS#6	see Table 4 on page 27	260	AH13
DCS#7	see Table 4 on page 27	357	AG13
DCS#8	see Table 4 on page 27	41	AK12
DEVSEL#		18	V1
DQM		265	AH18
FRAME#		13	N1
GND		3	C1
GND		4	D1
GND		15	R1
GND		24	AD1

Pin #	Signal Name	Alternate Signal	Grid #
1	NC		A1
2	PCLK3		B1
3	GND		C1
4	GND		D1
5	PCI_AD31		E1
6	PCI_AD28		F1
7	REQ#3		G1
8	REQ#2		H1
9	C/BE#3		J1
10	PCI_AD22		K1
11	PCI_AD19		L1
12	PCI_AD16		M1
13	FRAME#		N1
14	SERR#		P1
15	GND		R1
16	VDD		T1
17	PCI_AD12		U1
18	DEVSEL#		V1
19	PCI_AD7		W1
20	PERR#		Y1
21	PCI_AD1		AA1
22	LOC_AD30	PCI_AD62	AB1
23	LOC_AD28	PCI_AD60	AC1
24	GND		AD1
25	LOC_AD23	PCI_AD55	AE1
26	NC		AF1
27	LOC_AD22	PCI_AD54	AG1
28	VDD		AH1

Grid #	Signal Name	Alternate Signal	Pin #
A1	NC		1
A2	NC		116
A3	SysAD0		115
A4	SysAD3		114
A5	SysAD7		113
A6	SysAD10		112
A7	SysAD13		111
A8	SysAD18		110
A9	SysAD20		109
A10	SysAD23		108
A11	SysAD28		107
A12	SysAD31		106
A13	SysAD36		105
A14	SysAD39		104
A15	GND		103
A16	SysAD47		102
A17	SysAD48		101
A18	SysAD51		100
A19	SysAD56		99
A20	SysAD59		98
A21	SysADC0		97
A22	SysADC3		96
A23	SysADC5		95
A24	SysCmd2		94
A25	SysCmd5		93
A26	SysCmd8		92
A27	ScMatch		91
A28	Int#1		90

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
GND		36	AK7
GND		48	AK19
GND		53	AK24
GND		65	AD30
GND		103	A15
GND		146	AJ4
GND		167	AJ25
GND		226	D3
GND		244	AB3
GND		248	AF3
GND		251	AH4
GND		256	AH9
GND		269	AH22
GND		277	AF28
GND		287	T28
GND		328	G4
GND		336	R4
GND		351	AG7
GND		374	AD27
GND		417	E5
GND		420	H5
GND		421	J5
GND		422	K5
GND		423	L5
GND		424	M5
GND		425	N5
GND		426	P5
GND		428	T5
GND		429	U5
GND		430	V5
GND		431	W5
GND		432	Y5
GND		433	AA5
GND		435	AC5
GND		437	AE5
GND		438	AF5
GND		440	AF7
GND		441	AF8
GND		443	AF10
GND		444	AF11
GND		445	AF12
GND		446	AF13
GND		447	AF14

Pin #	Signal Name	Alternate Signal	Grid #
29	NC		AJ1
30	NC		AK1
31	LOC_AD14	PCI_AD46	AK2
32	LOC_AD10	PCI_AD42	AK3
33	LOC_AD9	PCI_AD41	AK4
34	LOC_AD6	PCI_AD38	AK5
35	LOC_AD3	PCI_AD35	AK6
36	GND		AK7
37	LOC_A3	C/BE#7	AK8
38	LOC_A1	C/BE#5	AK9
39	LOC_CLK	ACK64#	AK10
40	LOC_RDY#		AK11
41	DCS#8	see Table 4 on page 27	AK12
42	DCS#4	see Table 4 on page 27	AK13
43	INTE#		AK14
44	INTD#		AK15
45	INTA#		AK16
46	UART_DTR#		AK17
47	MRDY#		AK18
48	GND		AK19
49	MAbank110		AK20
50	MAbank16		AK21
51	MAbank13		AK22
52	MAbank11		AK23
53	GND		AK24
54	TEST_SEL		AK25
55	TEST#		AK26
56	NC		AK27
57	NC		AK28
58	NC		AK29
59	NC		AK30
60	MAbank010		AJ30
61	MAbank06		AH30
62	MAbank03		AG30
63	MAbank00		AF30
64	MWE#1		AE30
65	GND		AD30
66	MDC7		AC30
67	MDC5		AB30
68	MDC2		AA30
69	MD61		Y30
70	MD58		W30
71	MD53		V30

Grid #	Signal Name	Alternate Signal	Pin #
A29	Int#5		89
A30	NC		88
AA1	PCI_AD1		21
AA2	PCI_AD0		136
AA3	LOC_AD31	PCI_AD63	243
AA4	VDD		342
AA5	GND		433
AA26	GND		464
AA27	VDD		377
AA28	MDC4		282
AA29	MDC3		179
AA30	MDC2		68
AB1	LOC_AD30	PCI_AD62	22
AB2	LOC_AD29	PCI_AD61	137
AB3	GND		244
AB4	LOC_AD27	PCI_AD59	343
AB5	VDD		434
AB26	MWE#0		463
AB27	MCS#0		376
AB28	NC		281
AB29	MDC6		178
AB30	MDC5		67
AC1	LOC_AD28	PCI_AD60	23
AC2	LOC_AD26	PCI_AD58	138
AC3	LOC_AD25	PCI_AD57	245
AC4	VDD		344
AC5	GND		435
AC26	GND		462
AC27	VDD		375
AC28	MRAS#0		280
AC29	MCAS#0		177
AC30	MDC7		66
AD1	GND		24
AD2	LOC_AD24	PCI_AD56	139
AD3	PCLKIN		246
AD4	AGND		345
AD5	PCICR#		436
AD26	VDD		461
AD27	GND		374
AD28	MCAS#1		279
AD29	MCS#1		176
AD30	GND		65
AE1	LOC_AD23	PCI_AD55	25

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
GND		448	AF15
GND		449	AF16
GND		450	AF17
GND		452	AF19
GND		453	AF20
GND		454	AF21
GND		456	AF23
GND		459	AF26
GND		460	AE26
GND		462	AC26
GND		464	AA26
GND		466	W26
GND		468	U26
GND		471	P26
GND		473	M26
GND		475	K26
GND		477	H26
GND		478	G26
GND		479	F26
GND		480	E26
GND		481	E25
GND		482	E24
GND		483	E23
GND		485	E21
GND		487	E19
GND		489	E17
GND		492	E14
GND		494	E12
GND		496	E10
GND		498	E8
GNT#0		228	F3
GNT#1		120	E2
GNT#2		419	G5
GNT#3		327	F4
GNT#4		227	E3
IDSEL		232	K3
Int#0		200	B27
Int#1		90	A28
Int#2		395	D26
Int#3		301	C27
Int#4		199	B28
Int#5		89	A29
INTA#		45	AK16
INTB#		359	AG15
INTC#		157	AJ15

Pin #	Signal Name	Alternate Signal	Grid #
72	MD50		U30
73	MD49		T30
74	MD45		R30
75	MD40		P30
76	MD37		N30
77	MD32		M30
78	MD29		L30
79	MD24		K30
80	MD21		J30
81	MD19		H30
82	MD14		G30
83	MD11		F30
84	MD8		E30
85	MD4		D30
86	MD0		C30
87	NC		B30
88	NC		A30
89	Int#5		A29
90	Int#1		A28
91	ScMatch		A27
92	SysCmd8		A26
93	SysCmd5		A25
94	SysCmd2		A24
95	SysADC5		A23
96	SysADC3		A22
97	SysADC0		A21
98	SysAD59		A20
99	SysAD56		A19
100	SysAD51		A18
101	SysAD48		A17
102	SysAD47		A16
103	GND		A15
104	SysAD39		A14
105	SysAD36		A13
106	SysAD31		A12
107	SysAD28		A11
108	SysAD23		A10
109	SysAD20		A9
110	SysAD18		A8
111	SysAD13		A7
112	SysAD10		A6
113	SysAD7		A5
114	SysAD3		A4
115	SysAD0		A3
116	NC		A2

Grid #	Signal Name	Alternate Signal	Pin #
AE2	PCI64#		140
AE3	AVDD		247
AE4	LOC_AD21	PCI_AD53	346
AE5	GND		437
AE26	GND		460
AE27	MAbank04		373
AE28	MAbank01		278
AE29	MRAS#1		175
AE30	MWE#1		64
AF1	NC		26
AF2	NC		141
AF3	GND		248
AF4	LOC_AD19	PCI_AD51	347
AF5	GND		438
AF6	LOC_AD12	PCI_AD44	439
AF7	GND		440
AF8	GND		441
AF9	VDD		442
AF10	GND		443
AF11	GND		444
AF12	GND		445
AF13	GND		446
AF14	GND		447
AF15	GND		448
AF16	GND		449
AF17	GND		450
AF18	MAbank113		451
AF19	GND		452
AF20	GND		453
AF21	GND		454
AF22	MAbank014		455
AF23	GND		456
AF24	NC		457
AF25	AVDD		458
AF26	GND		459
AF27	MAbank07		372
AF28	GND		277
AF29	MAbank02		174
AF30	MAbank00		63
AG1	LOC_AD22	PCI_AD54	27
AG2	LOC_AD20	PCI_AD52	142
AG3	LOC_AD18	PCI_AD50	249
AG4	VDD		348
AG5	LOC_AD11	PCI_AD43	349
AG6	VDD		350

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
INTD#		44	AK15
INTE#		43	AK14
IRDY#		128	N2
LOC_A0	C/BE#4	257	AH10
LOC_A1	C/BE#5	38	AK9
LOC_A2	C/BE#6	151	AJ9
LOC_A3	C/BE#7	37	AK8
LOC_A4	PAR64	353	AG9
LOC_AD0	PCI_AD32	150	AJ8
LOC_AD1	PCI_AD33	255	AH8
LOC_AD10	PCI_AD42	32	AK3
LOC_AD11	PCI_AD43	349	AG5
LOC_AD12	PCI_AD44	439	AF6
LOC_AD13	PCI_AD45	145	AJ3
LOC_AD14	PCI_AD46	31	AK2
LOC_AD15	PCI_AD47	144	AJ2
LOC_AD16	PCI_AD48	250	AH3
LOC_AD17	PCI_AD49	143	AH2
LOC_AD18	PCI_AD50	249	AG3
LOC_AD19	PCI_AD51	347	AF4
LOC_AD2	PCI_AD34	149	AJ7
LOC_AD20	PCI_AD52	142	AG2
LOC_AD21	PCI_AD53	346	AE4
LOC_AD22	PCI_AD54	27	AG1
LOC_AD23	PCI_AD55	25	AE1
LOC_AD24	PCI_AD56	139	AD2
LOC_AD25	PCI_AD57	245	AC3
LOC_AD26	PCI_AD58	138	AC2
LOC_AD27	PCI_AD59	343	AB4
LOC_AD28	PCI_AD60	23	AC1
LOC_AD29	PCI_AD61	137	AB2
LOC_AD3	PCI_AD35	35	AK6
LOC_AD30	PCI_AD62	22	AB1
LOC_AD31	PCI_AD63	243	AA3
LOC_AD4	PCI_AD36	254	AH7
LOC_AD5	PCI_AD37	148	AJ6
LOC_AD6	PCI_AD38	34	AK5
LOC_AD7	PCI_AD39	253	AH6
LOC_AD8	PCI_AD40	147	AJ5

Pin #	Signal Name	Alternate Signal	Grid #
117	PCLK4		B2
118	PCLK2		C2
119	VDD		D2
120	GNT#1		E2
121	PCI_AD30		F2
122	PCI_AD27		G2
123	PCI_AD26		H2
124	REQ#1		J2
125	PCI_AD23		K2
126	PCI_AD20		L2
127	PCI_AD17		M2
128	IRDY#		N2
129	LOCK#		P2
130	PAR		R2
131	PCI_AD15		T2
132	PCI_AD11		U2
133	PCI_AD9		V2
134	PCI_AD6		W2
135	PCI_AD4		Y2
136	PCI_AD0		AA2
137	LOC_AD29	PCI_AD61	AB2
138	LOC_AD26	PCI_AD58	AC2
139	LOC_AD24	PCI_AD56	AD2
140	PCI64#		AE2
141	NC		AF2
142	LOC_AD20	PCI_AD52	AG2
143	LOC_AD17	PCI_AD49	AH2
144	LOC_AD15	PCI_AD47	AJ2
145	LOC_AD13	PCI_AD45	AJ3
146	GND		AJ4
147	LOC_AD8	PCI_AD40	AJ5
148	LOC_AD5	PCI_AD37	AJ6
149	LOC_AD2	PCI_AD34	AJ7
150	LOC_AD0	PCI_AD32	AJ8
151	LOC_A2	C/BE#6	AJ9
152	LOC_ALE	REQ64#	AJ10
153	LOC_FR#		AJ11
154	LOC_BGACK#		AJ12
155	DCS#5	see Table 4 on page 27	AJ13

Grid #	Signal Name	Alternate Signal	Pin #
AG7	GND		351
AG8	VDD		352
AG9	LOC_A4	PAR64	353
AG10	VDD		354
AG11	LOC_RD#		355
AG12	VDD		356
AG13	DCS#7	see Table 4 on page 27	357
AG14	VDD		358
AG15	INTB#		359
AG16	UART_DSR#		360
AG17	VDD		361
AG18	MAbank114		362
AG19	VDD		363
AG20	MAbank17		364
AG21	VDD		365
AG22	MAbank10		366
AG23	VDD		367
AG24	NC		368
AG25	NC		369
AG26	NC		370
AG27	VDD		371
AG28	MAbank08		276
AG29	MAbank05		173
AG30	MAbank03		62
AH1	VDD		28
AH2	LOC_AD17	PCI_AD49	143
AH3	LOC_AD16	PCI_AD48	250
AH4	GND		251
AH5	LOC_BG#	HLDA	252
AH6	LOC_AD7	PCI_AD39	253
AH7	LOC_AD4	PCI_AD36	254
AH8	LOC_AD1	PCI_AD33	255
AH9	GND		256
AH10	LOC_A0	C/BE#4	257
AH11	LOC_WR#		258
AH12	LOC_BR#	HOLD	259
AH13	DCS#6	see Table 4 on page 27	260
AH14	DCS#3	see Table 4 on page 27	261
AH15	NC		262

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
LOC_AD9	PCI_AD41	33	AK4
LOC_ALE	REQ64#	152	AJ10
LOC_BG#	HLDA	252	AH5
LOC_BGACK#		154	AJ12
LOC_BR#	HOLD	259	AH12
LOC_CLK	ACK64#	39	AK10
LOC_FR#		153	AJ11
LOC_RD#		355	AG11
LOC_RDY#		40	AK11
LOC_WR#		258	AH11
LOCK#		129	P2
M66EN		158	AJ16
MAbank00		63	AF30
MAbank01		278	AE28
MAbank02		174	AF29
MAbank03		62	AG30
MAbank04		373	AE27
MAbank05		173	AG29
MAbank06		61	AH30
MAbank07		372	AF27
MAbank08		276	AG28
MAbank09		172	AH29
MAbank010		60	AJ30
MAbank011		166	AJ24
MAbank012		270	AH23
MAbank013		165	AJ23
MAbank014		455	AF22
MAbank10		366	AG22
MAbank11		52	AK23
MAbank12		164	AJ22
MAbank13		51	AK22
MAbank14		268	AH21
MAbank15		163	AJ21
MAbank16		50	AK21
MAbank17		364	AG20
MAbank18		267	AH20
MAbank19		162	AJ20
MAbank110		49	AK20
MAbank111		266	AH19
MAbank112		161	AJ19

Pin #	Signal Name	Alternate Signal	Grid #
156	DCS#2	see Table 4 on page 27	AJ14
157	INTC#		AJ15
158	M66EN		AJ16
159	UART_RxDRDY#		AJ17
160	BootCS#		AJ18
161	MAbank112		AJ19
162	MAbank19		AJ20
163	MAbank15		AJ21
164	MAbank12		AJ22
165	MAbank013		AJ23
166	MAbank011		AJ24
167	GND		AJ25
168	NC		AJ26
169	NC		AJ27
170	AGND		AJ28
171	BigEndian		AJ29
172	MAbank09		AH29
173	MAbank05		AG29
174	MAbank02		AF29
175	MRAS#1		AE29
176	MCS#1		AD29
177	MCAS#0		AC29
178	MDC6		AB29
179	MDC3		AA29
180	MD62		Y29
181	MD59		W29
182	MD54		V29
183	MD51		U29
184	MD48		T29
185	MD44		R29
186	MD39		P29
187	MD36		N29
188	MD31		M29
189	MD28		L29
190	MD23		K29
191	MD20		J29
192	MD15		H29
193	MD12		G29
194	MD9		F29
195	MD5		E29

Grid #	Signal Name	Alternate Signal	Pin #
AH16	VccOk		263
AH17	UART_TxDRDY#		264
AH18	DQM		265
AH19	MAbank111		266
AH20	MAbank18		267
AH21	MAbank14		268
AH22	GND		269
AH23	MAbank012		270
AH24	SysClock		271
AH25	NC		272
AH26	NC		273
AH27	NC		274
AH28	NC		275
AH29	MAbank09		172
AH30	MAbank06		61
AJ1	NC		29
AJ2	LOC_AD15	PCI_AD47	144
AJ3	LOC_AD13	PCI_AD45	145
AJ4	GND		146
AJ5	LOC_AD8	PCI_AD40	147
AJ6	LOC_AD5	PCI_AD37	148
AJ7	LOC_AD2	PCI_AD34	149
AJ8	LOC_AD0	PCI_AD32	150
AJ9	LOC_A2	C/BE#6	151
AJ10	LOC_ALE	REQ64#	152
AJ11	LOC_FR#		153
AJ12	LOC_BGACK#		154
AJ13	DCS#5	see Table 4 on page 27	155
AJ14	DCS#2	see Table 4 on page 27	156
AJ15	INTC#		157
AJ16	M66EN		158
AJ17	UART_RxDRDY#		159
AJ18	BootCS#		160
AJ19	MAbank112		161
AJ20	MAbank19		162
AJ21	MAbank15		163
AJ22	MAbank12		164
AJ23	MAbank013		165
AJ24	MAbank011		166
AJ25	GND		167

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
MAbank113		451	AF18
MAbank114		362	AG18
MCAS#0		177	AC29
MCAS#1		279	AD28
MCS#0		376	AB27
MCS#1		176	AD29
MCWrRdy#		397	D24
MD0		86	C30
MD1		196	D29
MD2		298	E28
MD3		392	F27
MD4		85	D30
MD5		195	E29
MD6		297	F28
MD7		391	G27
MD8		84	E30
MD9		194	F29
MD10		296	G28
MD11		83	F30
MD12		193	G29
MD13		295	H28
MD14		82	G30
MD15		192	H29
MD16		476	J26
MD17		389	J27
MD18		294	J28
MD19		81	H30
MD20		191	J29
MD21		80	J30
MD22		293	K28
MD23		190	K29
MD24		79	K30
MD25		474	L26
MD26		387	L27
MD27		292	L28
MD28		189	L29
MD29		78	L30
MD30		291	M28
MD31		188	M29
MD32		77	M30
MD33		472	N26
MD34		385	N27
MD35		290	N28

Pin #	Signal Name	Alternate Signal	Grid #
196	MD1		D29
197	ModeOut		C29
198	NMI#		B29
199	Int#4		B28
200	Int#0		B27
201	ScDOE#		B26
202	SysCmd7		B25
203	SysCmd4		B24
204	SysCmd1		B23
205	SysADC4		B22
206	SysADC1		B21
207	SysAD60		B20
208	SysAD57		B19
209	SysAD52		B18
210	SysAD49		B17
211	SysAD46		B16
212	SysAD43		B15
213	SysAD38		B14
214	SysAD35		B13
215	SysAD30		B12
216	SysAD27		B11
217	SysAD22		B10
218	SysAD19		B9
219	SysAD14		B8
220	SysAD11		B7
221	SysAD8		B6
222	SMC		B5
223	CntrValid#		B4
224	CPUValid#		B3
225	PCIRST#		C3
226	GND		D3
227	GNT#4		E3
228	GNT#0		F3
229	PCI_AD29		G3
230	REQ#4		H3
231	PCI_AD24		J3
232	IDSEL		K3
233	REQ#0		L3
234	PCI_AD18		M3
235	TRDY#		N3
236	STOP#		P3
237	NC		R3
238	PCI_AD14		T3

Grid #	Signal Name	Alternate Signal	Pin #
AJ26	NC		168
AJ27	NC		169
AJ28	AGND		170
AJ29	BigEndian		171
AJ30	MAbank010		60
AK1	NC		30
AK2	LOC_AD14	PCI_AD46	31
AK3	LOC_AD10	PCI_AD42	32
AK4	LOC_AD9	PCI_AD41	33
AK5	LOC_AD6	PCI_AD38	34
AK6	LOC_AD3	PCI_AD35	35
AK7	GND		36
AK8	LOC_A3	C/BE#7	37
AK9	LOC_A1	C/BE#5	38
AK10	LOC_CLK	ACK64#	39
AK11	LOC_RDY#		40
AK12	DCS#8	see Table 4 on page 27	41
AK13	DCS#4	see Table 4 on page 27	42
AK14	INTE#		43
AK15	INTD#		44
AK16	INTA#		45
AK17	UART_DTR#		46
AK18	MRDY#		47
AK19	GND		48
AK20	MAbank110		49
AK21	MAbank16		50
AK22	MAbank13		51
AK23	MAbank11		52
AK24	GND		53
AK25	TEST_SEL		54
AK26	TEST#		55
AK27	NC		56
AK28	NC		57
AK29	NC		58
AK30	NC		59
B1	PCLK3		2
B2	PCLK4		117
B3	CPUValid#		224
B4	CntrValid#		223
B5	SMC		222
B6	SysAD8		221
B7	SysAD11		220
B8	SysAD14		219

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
MD36		187	N29
MD37		76	N30
MD38		289	P28
MD39		186	P29
MD40		75	P30
MD41		470	R26
MD42		383	R27
MD43		288	R28
MD44		185	R29
MD45		74	R30
MD46		382	T27
MD47		469	T26
MD48		184	T29
MD49		73	T30
MD50		72	U30
MD51		183	U29
MD52		286	U28
MD53		71	V30
MD54		182	V29
MD55		285	V28
MD56		380	V27
MD57		467	V26
MD58		70	W30
MD59		181	W29
MD60		284	W28
MD61		69	Y30
MD62		180	Y29
MD63		283	Y28
MDC0		378	Y27
MDC1		465	Y26
MDC2		68	AA30
MDC3		179	AA29
MDC4		282	AA28
MDC5		67	AB30
MDC6		178	AB29
MDC7		66	AC30
ModeClock		300	C28
ModeOut		197	C29
MRAS#0		280	AC28
MRAS#1		175	AE29
MRDY#		47	AK18
MWE#0		463	AB26

Pin #	Signal Name	Alternate Signal	Grid #
239	PCI_AD10		U3
240	PCI_AD8		V3
241	PCI_AD5		W3
242	PCI_AD3		Y3
243	LOC_AD31	PCI_AD63	AA3
244	GND		AB3
245	LOC_AD25	PCI_AD57	AC3
246	PCLKIN		AD3
247	AVDD		AE3
248	GND		AF3
249	LOC_AD18	PCI_AD50	AG3
250	LOC_AD16	PCI_AD48	AH3
251	GND		AH4
252	LOC_BG#	HLDA	AH5
253	LOC_AD7	PCI_AD39	AH6
254	LOC_AD4	PCI_AD36	AH7
255	LOC_AD1	PCI_AD33	AH8
256	GND		AH9
257	LOC_A0	C/BE#4	AH10
258	LOC_WR#		AH11
259	LOC_BR#	HOLD	AH12
260	DCS#6	see Table 4 on page 27	AH13
261	DCS#3	see Table 4 on page 27	AH14
262	NC		AH15
263	VccOk		AH16
264	UART_TxDRDY#		AH17
265	DQM		AH18
266	MAbank111		AH19
267	MAbank18		AH20
268	MAbank14		AH21
269	GND		AH22
270	MAbank012		AH23
271	SysClock		AH24
272	NC		AH25
273	NC		AH26
274	NC		AH27
275	NC		AH28
276	MAbank08		AG28
277	GND		AF28
278	MAbank01		AE28
279	MCAS#1		AD28
280	MRAS#0		AC28

Grid #	Signal Name	Alternate Signal	Pin #
B9	SysAD19		218
B10	SysAD22		217
B11	SysAD27		216
B12	SysAD30		215
B13	SysAD35		214
B14	SysAD38		213
B15	SysAD43		212
B16	SysAD46		211
B17	SysAD49		210
B18	SysAD52		209
B19	SysAD57		208
B20	SysAD60		207
B21	SysADC1		206
B22	SysADC4		205
B23	SysCmd1		204
B24	SysCmd4		203
B25	SysCmd7		202
B26	ScDOE#		201
B27	Int#0		200
B28	Int#4		199
B29	NMI#		198
B30	NC		87
C1	GND		3
C2	PCLK2		118
C3	PCIRST#		225
C4	CntrVccOk		324
C5	SysAD1		323
C6	SysAD5		322
C7	SysAD9		321
C8	SysAD12		320
C9	SysAD17		319
C10	SysAD21		318
C11	SysAD26		317
C12	SysAD29		316
C13	SysAD34		315
C14	SysAD37		314
C15	SysAD42		313
C16	NC		312
C17	SysAD50		311
C18	SysAD53		310
C19	SysAD58		309
C20	SysAD61		308

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
MWE#1		64	AE30
NC		29	AJ1
NC		237	R3
NC		1	A1
NC		116	A2
NC		312	C16
NC		88	A30
NC		87	B30
NC		281	AB28
NC		59	AK30
NC		58	AK29
NC		262	AH15
NC		30	AK1
NMI#		198	B29
PAR		130	R2
PCI64#		140	AE2
PCI_AD0		136	AA2
PCI_AD1		21	AA1
PCI_AD2		341	Y4
PCI_AD3		242	Y3
PCI_AD4		135	Y2
PCI_AD5		241	W3
PCI_AD6		134	W2
PCI_AD7		19	W1
PCI_AD8		240	V3
PCI_AD9		133	V2
PCI_AD10		239	U3
PCI_AD11		132	U2
PCI_AD12		17	U1
PCI_AD13		337	T4
PCI_AD14		238	T3
PCI_AD15		131	T2
PCI_AD16		12	M1
PCI_AD17		127	M2
PCI_AD18		234	M3
PCI_AD19		11	L1
PCI_AD20		126	L2
PCI_AD21		332	L4
PCI_AD22		10	K1
PCI_AD23		125	K2
PCI_AD24		231	J3
PCI_AD25		330	J4
PCI_AD26		123	H2
PCI_AD27		122	G2
PCI_AD28		6	F1
PCI_AD29		229	G3

Pin #	Signal Name	Alternate Signal	Grid #
281	NC		AB28
282	MDC4		AA28
283	MD63		Y28
284	MD60		W28
285	MD55		V28
286	MD52		U28
287	GND		T28
288	MD43		R28
289	MD38		P28
290	MD35		N28
291	MD30		M28
292	MD27		L28
293	MD22		K28
294	MD18		J28
295	MD13		H28
296	MD10		G28
297	MD6		F28
298	MD2		E28
299	PROM_CLK		D28
300	ModeClock		C28
301	Int#3		C27
302	ScWord1		C26
303	WrRdy#		C25
304	SysCmd6		C24
305	SysCmd3		C23
306	SysADC6		C22
307	SysADC2		C21
308	SysAD61		C20
309	SysAD58		C19
310	SysAD53		C18
311	SysAD50		C17
312	NC		C16
313	SysAD42		C15
314	SysAD37		C14
315	SysAD34		C13
316	SysAD29		C12
317	SysAD26		C11
318	SysAD21		C10
319	SysAD17		C9
320	SysAD12		C8
321	SysAD9		C7
322	SysAD5		C6
323	SysAD1		C5
324	CntrVccOk		C4
325	VDD		D4
326	PCLK0		E4

Grid #	Signal Name	Alternate Signal	Pin #
C21	SysADC2		307
C22	SysADC6		306
C23	SysCmd3		305
C24	SysCmd6		304
C25	WrRdy#		303
C26	ScWord1		302
C27	Int#3		301
C28	ModeClock		300
C29	ModeOut		197
C30	MD0		86
D1	GND		4
D2	VDD		119
D3	GND		226
D4	VDD		325
D5	Reset#		416
D6	SysAD2		415
D7	SysAD6		414
D8	VDD		413
D9	SysAD16		412
D10	VDD		411
D11	SysAD25		410
D12	VDD		409
D13	SysAD33		408
D14	VDD		407
D15	SysAD41		406
D16	SysAD44		405
D17	VDD		404
D18	SysAD54		403
D19	VDD		402
D20	SysAD62		401
D21	VDD		400
D22	SysADC7		399
D23	VDD		398
D24	MCWrRdy#		397
D25	ScWord0		396
D26	Int#2		395
D27	VDD		394
D28	PROM_CLK		299
D29	MD1		196
D30	MD4		85
E1	PCI_AD31		5
E2	GNT#1		120
E3	GNT#4		227
E4	PCLK0		326
E5	GND		417
E6	ColdReset#		500

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
PCI_AD30		121	F2
PCI_AD31		5	E1
PCICR#		436	AD5
PCIRST#		225	C3
PCLK0		326	E4
PCLK1		418	F5
PCLK2		118	C2
PCLK3		2	B1
PCLK4		117	B2
PCLKIN		246	AD3
PERR#		20	Y1
NC		141	AF2
NC		26	AF1
NC		457	AF24
NC		168	AJ26
NC		272	AH25
NC		368	AG24
NC		275	AH28
NC		274	AH27
NC		370	AG26
NC		57	AK28
NC		169	AJ27
NC		273	AH26
NC		369	AG25
NC		56	AK27
PROM_CLK		299	D28
PROM_SD		393	E27
REQ#0		233	L3
REQ#1		124	J2
REQ#2		8	H1
REQ#3		7	G1
REQ#4		230	H3
Reset#		416	D5
ScDOE#		201	B26
ScMatch		91	A27
ScWord0		396	D25
ScWord1		302	C26
SERR#		14	P1
SMC		222	B5
STOP#		236	P3
SysAD0		115	A3
SysAD1		323	C5
SysAD2		415	D6

Pin #	Signal Name	Alternate Signal	Grid #
327	GNT#3		F4
328	GND		G4
329	VDD		H4
330	PCI_AD25		J4
331	VDD		K4
332	PCI_AD21		L4
333	VDD		M4
334	C/BE#2		N4
335	VDD		P4
336	GND		R4
337	PCI_AD13		T4
338	VDD		U4
339	C/BE#0		V4
340	VDD		W4
341	PCI_AD2		Y4
342	VDD		AA4
343	LOC_AD27	PCI_AD59	AB4
344	VDD		AC4
345	AGND		AD4
346	LOC_AD21	PCI_AD53	AE4
347	LOC_AD19	PCI_AD51	AF4
348	VDD		AG4
349	LOC_AD11	PCI_AD43	AG5
350	VDD		AG6
351	GND		AG7
352	VDD		AG8
353	LOC_A4	PAR64	AG9
354	VDD		AG10
355	LOC_RD#		AG11
356	VDD		AG12
357	DCS#7	see Table 4 on page 27	AG13
358	VDD		AG14
359	INTB#		AG15
360	UART_DSR#		AG16
361	VDD		AG17
362	MAbank114		AG18
363	VDD		AG19
364	MAbank17		AG20
365	VDD		AG21
366	MAbank10		AG22
367	VDD		AG23
368	NC		AG24
369	NC		AG25

Grid #	Signal Name	Alternate Signal	Pin #
E7	SysAD4		499
E8	GND		498
E9	SysAD15		497
E10	GND		496
E11	SysAD24		495
E12	GND		494
E13	SysAD32		493
E14	GND		492
E15	SysAD40		491
E16	SysAD45		490
E17	GND		489
E18	SysAD55		488
E19	GND		487
E20	SysAD63		486
E21	GND		485
E22	SysCmd0		484
E23	GND		483
E24	GND		482
E25	GND		481
E26	GND		480
E27	PROM_SD		393
E28	MD2		298
E29	MD5		195
E30	MD8		84
F1	PCI_AD28		6
F2	PCI_AD30		121
F3	GNT#0		228
F4	GNT#3		327
F5	PCLK1		418
F26	GND		479
F27	MD3		392
F28	MD6		297
F29	MD9		194
F30	MD11		83
G1	REQ#3		7
G2	PCI_AD27		122
G3	PCI_AD29		229
G4	GND		328
G5	GNT#2		419
G26	GND		478
G27	MD7		391
G28	MD10		296
G29	MD12		193

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
SysAD3		114	A4
SysAD4		499	E7
SysAD5		322	C6
SysAD6		414	D7
SysAD7		113	A5
SysAD8		221	B6
SysAD9		321	C7
SysAD10		112	A6
SysAD11		220	B7
SysAD12		320	C8
SysAD13		111	A7
SysAD14		219	B8
SysAD15		497	E9
SysAD16		412	D9
SysAD17		319	C9
SysAD18		110	A8
SysAD19		218	B9
SysAD20		109	A9
SysAD21		318	C10
SysAD22		217	B10
SysAD23		108	A10
SysAD24		495	E11
SysAD25		410	D11
SysAD26		317	C11
SysAD27		216	B11
SysAD28		107	A11
SysAD29		316	C12
SysAD30		215	B12
SysAD31		106	A12
SysAD32		493	E13
SysAD33		408	D13
SysAD34		315	C13
SysAD35		214	B13
SysAD36		105	A13
SysAD37		314	C14
SysAD38		213	B14
SysAD39		104	A14
SysAD40		491	E15
SysAD41		406	D15
SysAD42		313	C15
SysAD43		212	B15
SysAD44		405	D16
SysAD45		490	E16
SysAD46		211	B16
SysAD47		102	A16
SysAD48		101	A17

Pin #	Signal Name	Alternate Signal	Grid #
370	NC		AG26
371	VDD		AG27
372	MAbank07		AF27
373	MAbank04		AE27
374	GND		AD27
375	VDD		AC27
376	MCS#0		AB27
377	VDD		AA27
378	MDC0		Y27
379	VDD		W27
380	MD56		V27
381	VDD		U27
382	MD46		T27
383	MD42		R27
384	VDD		P27
385	MD34		N27
386	VDD		M27
387	MD26		L27
388	VDD		K27
389	MD17		J27
390	VDD		H27
391	MD7		G27
392	MD3		F27
393	PROM_SD		E27
394	VDD		D27
395	Int#2		D26
396	ScWord0		D25
397	MCWrRdy#		D24
398	VDD		D23
399	SysADC7		D22
400	VDD		D21
401	SysAD62		D20
402	VDD		D19
403	SysAD54		D18
404	VDD		D17
405	SysAD44		D16
406	SysAD41		D15
407	VDD		D14
408	SysAD33		D13
409	VDD		D12
410	SysAD25		D11
411	VDD		D10
412	SysAD16		D9
413	VDD		D8
414	SysAD6		D7
415	SysAD2		D6

Grid #	Signal Name	Alternate Signal	Pin #
G30	MD14		82
H1	REQ#2		8
H2	PCI_AD26		123
H3	REQ#4		230
H4	VDD		329
H5	GND		420
H26	GND		477
H27	VDD		390
H28	MD13		295
H29	MD15		192
H30	MD19		81
J1	C/BE#3		9
J2	REQ#1		124
J3	PCI_AD24		231
J4	PCI_AD25		330
J5	GND		421
J26	MD16		476
J27	MD17		389
J28	MD18		294
J29	MD20		191
J30	MD21		80
K1	PCI_AD22		10
K2	PCI_AD23		125
K3	IDSEL		232
K4	VDD		331
K5	GND		422
K26	GND		475
K27	VDD		388
K28	MD22		293
K29	MD23		190
K30	MD24		79
L1	PCI_AD19		11
L2	PCI_AD20		126
L3	REQ#0		233
L4	PCI_AD21		332
L5	GND		423
L26	MD25		474
L27	MD26		387
L28	MD27		292
L29	MD28		189
L30	MD29		78
M1	PCI_AD16		12
M2	PCI_AD17		127
M3	PCI_AD18		234
M4	VDD		333
M5	GND		424

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
SysAD49		210	B17
SysAD50		311	C17
SysAD51		100	A18
SysAD52		209	B18
SysAD53		310	C18
SysAD54		403	D18
SysAD55		488	E18
SysAD56		99	A19
SysAD57		208	B19
SysAD58		309	C19
SysAD59		98	A20
SysAD60		207	B20
SysAD61		308	C20
SysAD62		401	D20
SysAD63		486	E20
SysADC0		97	A21
SysADC1		206	B21
SysADC2		307	C21
SysADC3		96	A22
SysADC4		205	B22
SysADC5		95	A23
SysADC6		306	C22
SysADC7		399	D22
SysClock		271	AH24
SysCmd0		484	E22
SysCmd1		204	B23
SysCmd2		94	A24
SysCmd3		305	C23
SysCmd4		203	B24
SysCmd5		93	A25
SysCmd6		304	C24
SysCmd7		202	B25
SysCmd8		92	A26
TEST#		55	AK26
TEST_SEL		54	AK25
TRDY#		235	N3
UART_DSR#		360	AG16
UART_DTR#		46	AK17
UART_RxDRDY#		159	AJ17
UART_TxDRDY#		264	AH17
VccOk		263	AH16
VDD		16	T1
VDD		28	AH1
VDD		119	D2
VDD		325	D4

Pin #	Signal Name	Alternate Signal	Grid #
416	Reset#		D5
417	GND		E5
418	PCLK1		F5
419	GNT#2		G5
420	GND		H5
421	GND		J5
422	GND		K5
423	GND		L5
424	GND		M5
425	GND		N5
426	GND		P5
427	C/BE#1		R5
428	GND		T5
429	GND		U5
430	GND		V5
431	GND		W5
432	GND		Y5
433	GND		AA5
434	VDD		AB5
435	GND		AC5
436	PCICR#		AD5
437	GND		AE5
438	GND		AF5
439	LOC_AD12	PCI_AD44	AF6
440	GND		AF7
441	GND		AF8
442	VDD		AF9
443	GND		AF10
444	GND		AF11
445	GND		AF12
446	GND		AF13
447	GND		AF14
448	GND		AF15
449	GND		AF16
450	GND		AF17
451	MAbank113		AF18
452	GND		AF19
453	GND		AF20
454	GND		AF21
455	MAbank014		AF22
456	GND		AF23
457	NC		AF24
458	AVDD		AF25
459	GND		AF26
460	GND		AE26

Grid #	Signal Name	Alternate Signal	Pin #
M26	GND		473
M27	VDD		386
M28	MD30		291
M29	MD31		188
M30	MD32		77
N1	FRAME#		13
N2	IRDY#		128
N3	TRDY#		235
N4	C/BE#2		334
N5	GND		425
N26	MD33		472
N27	MD34		385
N28	MD35		290
N29	MD36		187
N30	MD37		76
P1	SERR#		14
P2	LOCK#		129
P3	STOP#		236
P4	VDD		335
P5	GND		426
P26	GND		471
P27	VDD		384
P28	MD38		289
P29	MD39		186
P30	MD40		75
R1	GND		15
R2	PAR		130
R3	NC		237
R4	GND		336
R5	C/BE#1		427
R26	MD41		470
R27	MD42		383
R28	MD43		288
R29	MD44		185
R30	MD45		74
T1	VDD		16
T2	PCI_AD15		131
T3	PCI_AD14		238
T4	PCI_AD13		337
T5	GND		428
T26	MD47		469
T27	MD46		382
T28	GND		287
T29	MD48		184
T30	MD49		73

Table 47: Pinout Sorted By Signal Name, Pin Number, and Grid Number (continued)

Signal Name	Alternate Signal	Pin #	Grid #
VDD		329	H4
VDD		331	K4
VDD		333	M4
VDD		335	P4
VDD		338	U4
VDD		340	W4
VDD		342	AA4
VDD		344	AC4
VDD		348	AG4
VDD		350	AG6
VDD		352	AG8
VDD		354	AG10
VDD		356	AG12
VDD		358	AG14
VDD		361	AG17
VDD		363	AG19
VDD		365	AG21
VDD		367	AG23
VDD		371	AG27
VDD		375	AC27
VDD		377	AA27
VDD		379	W27
VDD		381	U27
VDD		384	P27
VDD		386	M27
VDD		388	K27
VDD		390	H27
VDD		394	D27
VDD		398	D23
VDD		400	D21
VDD		402	D19
VDD		404	D17
VDD		407	D14
VDD		409	D12
VDD		411	D10
VDD		413	D8
VDD		434	AB5
VDD		442	AF9
VDD		461	AD26
WrRdy#		303	C25

Pin #	Signal Name	Alternate Signal	Grid #
461	VDD		AD26
462	GND		AC26
463	MWE#0		AB26
464	GND		AA26
465	MDC1		Y26
466	GND		W26
467	MD57		V26
468	GND		U26
469	MD47		T26
470	MD41		R26
471	GND		P26
472	MD33		N26
473	GND		M26
474	MD25		L26
475	GND		K26
476	MD16		J26
477	GND		H26
478	GND		G26
479	GND		F26
480	GND		E26
481	GND		E25
482	GND		E24
483	GND		E23
484	SysCmd0		E22
485	GND		E21
486	SysAD63		E20
487	GND		E19
488	SysAD55		E18
489	GND		E17
490	SysAD45		E16
491	SysAD40		E15
492	GND		E14
493	SysAD32		E13
494	GND		E12
495	SysAD24		E11
496	GND		E10
497	SysAD15		E9
498	GND		E8
499	SysAD4		E7
500	ColdReset#		E6

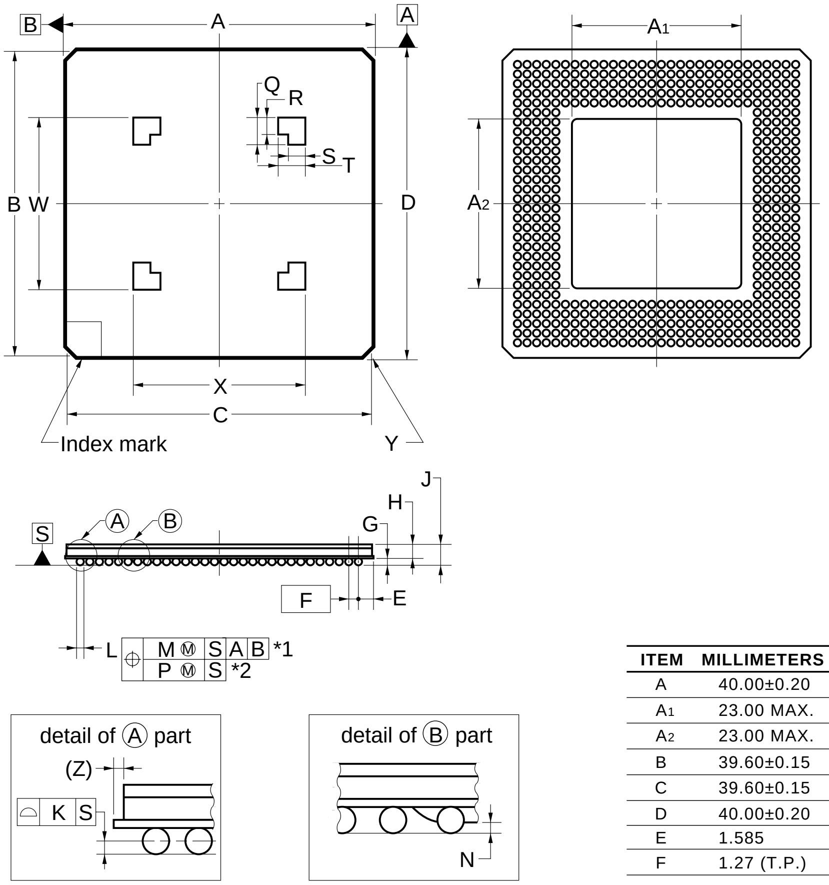
Grid #	Signal Name	Alternate Signal	Pin #
U1	PCI_AD12		17
U2	PCI_AD11		132
U3	PCI_AD10		239
U4	VDD		338
U5	GND		429
U26	GND		468
U27	VDD		381
U28	MD52		286
U29	MD51		183
U30	MD50		72
V1	DEVSEL#		18
V2	PCI_AD9		133
V3	PCI_AD8		240
V4	C/BE#0		339
V5	GND		430
V26	MD57		467
V27	MD56		380
V28	MD55		285
V29	MD54		182
V30	MD53		71
W1	PCI_AD7		19
W2	PCI_AD6		134
W3	PCI_AD5		241
W4	VDD		340
W5	GND		431
W26	GND		466
W27	VDD		379
W28	MD60		284
W29	MD59		181
W30	MD58		70
Y1	PERR#		20
Y2	PCI_AD4		135
Y3	PCI_AD3		242
Y4	PCI_AD2		341
Y5	GND		432
Y26	MDC1		465
Y27	MDC0		378
Y28	MD63		283
Y29	MD62		180
Y30	MD61		69

18.0 Package

Figure 49 shows the controller’s 500-pin TBGA package.

Figure 49: 500-Pin TBGA Package

500 PIN TAPE BGA (HEAT SPREADER TYPE) (40x40)



NOTES

*1 Each ball centerline is located within ϕ0.30 mm (ϕ0.012 inch) of its true position (T.P.) at maximum material condition.

*2 Each ball centerline is located within ϕ0.10 mm (ϕ0.004 inch) of its true position (T.P.) at maximum material condition.

ITEM	MILLIMETERS	INCHES
A	40.00±0.20	1.575±0.008
A_1	23.00 MAX.	0.906 MAX.
A_2	23.00 MAX.	0.906 MAX.
B	39.60±0.15	1.559±0.006
C	39.60±0.15	1.559±0.006
D	40.00±0.20	1.575±0.008
E	1.585	0.062
F	1.27 (T.P.)	0.050 (T.P.)
G	0.60±0.10	$0.024^{+0.004}_{-0.005}$
H	$0.80^{+0.20}_{-0.10}$	$0.031^{+0.009}_{-0.004}$
J	$1.40^{+0.30}_{-0.20}$	$0.055^{+0.012}_{-0.008}$
K	0.15	0.006
L	ϕ0.75±0.15	$\phi 0.030^{+0.006}_{-0.007}$
M	0.30	0.012
N	0.25 MIN.	0.009 MIN.
P	0.10	0.004
Q	3.0	0.118
R	2.0	0.079
S	2.0	0.079
T	3.0	0.118
W	22.73	0.895
X	22.73	0.895
Y	C 0.40	C 0.016
Z	0.20	0.008

S500N7-H6

Appendix A Revision 2 Errata

The following bugs or revision-specific states exist for Revision 2 of the controller.

A.1 Serial Configuration Stream

The controller generates the default serial configuration stream incorrectly (See Section 12.4.2). Use an external Serial Mode EEPROM, connected to the controller. There is no need to connect a Serial Mode EEPROM to the CPU.

A.2 PCI-Bus Interface

A.2.1 Revision ID

The PCI Revision ID register has the value 0x02.

A.2.2 PCI Timing Problems

The PCI bus does not meet the 66 MHz PCI specification, due to setup/hold timing on a variety of signals. Thus, the PCI Bus cannot be clocked at 66 MHz.

A.2.3 PCI Loopback Reads

If the controller initiates a PCI read, and the target is the same controller, bad things happen. This bug applies to all PCI reads, including PCI configuration cycles.

For example, the CPU can only read the controller’s PCI configuration registers by accessing them directly with the internal address described in Section 7.12. The CPU cannot access these registers via PCI configuration cycles on the PCI Bus.

A.2.4 PCI LOCK#

The controller does not generate or respond to PCI locked cycles. If PCI locked cycles are being used by other devices in the system, the LOCK# signal on controller should be tied off (driven with a constant value 0 or 1), otherwise controller may not properly complete delayed transactions as a target.

If multiple VRC5074 controllers are connected to a CPU, both controllers can be connected to LOCK# if LOCK# is pulled up.

A.2.5 PCI Address Parity Error

When a parity error occurs during an access by a PCI-Bus master to the controller as target, the controller accepts the access. Here's what happens:

1. The external master does an access (e.g. a write) to somewhere. During the address phase there is a parity error. This means the address is corrupted, and this access should be ignored by all targets! (but see Section 3.8.2.2. in the *PCI Local Bus Specification*.)
2. The controller decodes the corrupted address and thinks this access is for it. The controller completes the access (if a write to SDRAM, then bad data gets written).

The controller can assert SERR# and/or interrupt the CPU when a PCI address error occurs, as described in Section 7.5.4. This should be considered a catastrophic system error. Reset is an appropriate response.

Most or all PCI targets will respond normally to a PCI access with an address parity error. Also most PC systems will assert NMI#, causing a system panic or reset.

A.2.6 PCI Target Prefetch May Cause OUTFIFO Overrun

When a PCI read is performed with the controller as the target, and prefetching is enabled with the PREFETCHABLE bit in the PCI Master (Initiator) Control Registers (PCIINITn, Section 7.11.3), a data overrun may occur in the OUTFIFO. Prefetching must not be used on PCI target reads. This means:

- The PREFETCHABLE bit must be cleared in the Base Address Register. See Section 7.13.10.
- Memory Read Line and Memory Read Multiple commands must not be used. Only Memory Read should be used.

If both the PREFETCHABLE bit and the cache line size register (CLSIZ, Section 7.13.7) are cleared to 0, Memory Read and Memory Read Multiple commands may still cause overruns, but only if there are many pending PCI writes that use up the OUTFIFO's 32-dword capacity.

For example, writing 8 to the CLSIZ register causes the controller to fetch 8 words (4 dwords) during PCI target reads, using four dword entries in the OUTFIFO. There can be a maximum of four outstanding reads, which would use 16 of the OUTFIFO's 32 dword entries. In this case, an OUTFIFO overrun will occur if there are, at the same time, more than four outstanding PCI writes.

See Section 7.4.4.2 for more information on prefetching during target reads.

A.3 Secondary Cache in Multi-Controller Configuration

Section 5.3 describes the controller's operation with multiple external agents. When L2 cache is enabled there can be problems with the ScDOE# signal driven by the controller:

- There must be multiple controllers (e.g. multiple external agents A and B).
- L2 cache must be enabled.
- Agent B must be a VRC5074 controller. Agent A may be a controller or a compatible device.
- If L2 cache miss occurs on a block read to agent A, that agent responds with the correct read data. Thereafter, whenever a non-block read is performed to agent B, the ScDOE# signal is incorrectly left high (negated) at the end of the transaction. This can cause subsequent transactions to be corrupted.

There are several workarounds:

- Do not allow cacheable accesses to agent A.
- After any cacheable access to agent A, the first access to agent B must be a block read cache miss. This will cause the state of ScDOE# to be set properly.
- Agent B must be the Main Controller (see Section 5.3.2). After any cacheable

access to agent A, the first access to agent B must cause a CPU-Bus Read Timeout (seeSection 5.3.3). This will cause the state of ScDOE# to be set properly.

❑ Put a strong pulldown resistor on ScDOE#. After agent B leaves ScDOE# in a high state, there are a minimum of three idle clocks before ScDOE# must be low for a subsequent cache hit.

A.4

UART External Clock

Section 10.2 and Section 8.6.3 describe how DCS#[5] can optionally be configured as the UART_XIN signal. This feature does not work. The UART must be clocked with the internally-generated UART clock (SYS_CLK divided by 12).

Appendix B Index