System Controller for PowerPC Processors

FEATURES

The MV64360/1/2 devices are a family of integrated system controllers designed for high-performance embedded control applications. They offer a perfect solution for most internetworking applications.

MV64360 – Two 64-bit PCI-X buses and three integrated Gigabit Ethernet (GbE) MAC controllers.

MV64361 –Two 32-bit PCI-X buses and two integrated Gigabit Ethernet MAC controllers.

MV64362 –One 64-bit PCI-X bus and one integrated Gigabit Ethernet MAC controller.

Up to 133 MHz (commercial) or 125 MHz (Industrial) CPU bus frequency.

Selectable 1.8V, 2.5V, or 3.3VCPU bus interface.

Supports PowerPC CPUs.

- Motorola MPC750,755,74xx.
- IBM PPC750,750Cx, 750Fx.
- Other 64-bit 60x bus CPUs.

CPU specific features:

- Demultiplexed 36-bit address 64-bit data busses.
- Supports both 60x and MPX bus modes.
- Supports up to 16 outstanding transactions in dual CPU configuration. (Dual CPU configuration is only valid for the MV64360 and MV64361.)
- Supports MPX out of order completion.
- Supports MPX address and data streaming.
- Supports MPC7450 extended 36-bit address.
- Supports dual CPU in both 60x and MPX bus modes. Only applies to the MV64360/1 devices.
- Supports MPX bus data intervention (cache to cache data transfer).

Supports dual CPU SMP

- Inter CPU doorbell interrupts
- Separate CPU interrupts
- Inter CPU semaphores

Supports PowerPC cache coherency between the DRAM and the integrated SRAM to the processor caches.

- Configurable cache coherency regions. Any PCI, DMA, or Ethernet port access to the DRAM, or to the integrated SRAM, may result in snoop transaction on the CPU bus.
- Supports WB and WT cache policy.

PowerPC bus master capability:

- Generates address only snoop transactions.
- In 60x bus mode, also allows data transfers between the MV64360/1/2 interfaces (PCI, DMAs) and other devices on the CPU bus.

Supports up to four slave devices (MV64360/1/2 or other slave devices) on the same 60x bus.

CPU address remapping to the PCI.

Support access, write, and caching protection to a configurable address range.

Integrated 2 Mb SRAM (only applies to the MV64360 and MV64361 devices).

- Low latency CPU access.
- Accessible from any of the MV64360/1/2 interfaces.
- Useful for descriptors and packet headers
- Packets headers retargeted to the SRAM

DDR SDRAM Controller:

- Up to 183 MHz clock frequency (366 MHz data rate) for single load.
- SSTL_2 I/Os.
- Supports four DRAM banks.
- Supports all DDR devices densities up to 1 Gb.
- Up to 8 GB address space (with 1 Gb devices).
- Supports DRAM bank interleaving between all DRAM banks (both the physical banks, and the four internal banks of the DRAM devices).
- Supports up to 16 open pages.
- Supports configurable DRAM timing parameters.
- Supports up to 128 byte burst per single DRAM access.

FEATURES

Device controller:

- 32-bit multiplexed address/data bus.
- Up to 133 MHz clock frequency.
- 3.3V I/Os.
- Can be used as a high bandwidth interface to user specific logic.
- Supports many types of standard memory devices such as FLASH, ROM, and SyncBurst SRAM.
- Five chip selects with programmable timing.
- Optional external wait-state support.
- 8-,16-,32-bit width device support.
- Support for boot ROMs.

Two 64-bit (MV64360), two 32-bit (MV64361), or one 64-bit (MV64362) PCI/PCI-X interfaces.

- 66 MHz PCI 2.2 compliant interface.
- 133 MHz PCI-X compliant interface.
- 3.3V I/Os, 5V tolerant.
- Supports PCI-to-PCI memory, I/O, and configuration transactions between the two PCI interfaces.
- Supports PCI to PCI-X bridging between the two PCI interfaces. Only applies to the MV64360/1 devices.
- The PCI interfaces can run in asynchronous clock mode to each other and to the MV64360/1/2 core clock. The MV64362 can only run in asynchronous clock mode to the core clock.
- 32/64-bit PCI master and target operations.Only applies to the MV64360/2 devices.
- Supports flexible byte swapping for interfacing Big and Little Endian PCI devices.
- Supports 64-bit addressing via DAC transactions.
- Configurable PCI arbiter for up to six external masters.

PCI master specific features:

- Supports all PCI and PCI-X cycles.
- Host to PCI bridge - translates CPU cycles to PCI Memory, I/O, or configuration cycles.
- Supports DMA bursts between PCI and memory.
- Supports transaction combining to unlimited PCI burst (conventional PCI).
- Supports up to four split transactions (PCI-X).

PCI target specific features:

- Supports all PCI and PCI-X cycles.
- Supports programmable read prefetch (conventional PCI).
- Supports up to 4 KB read per single transaction (PCI-X).
- Supports unlimited burst write with zero wait states.
- Supports up to four delayed reads (conventional PCI).
- Supports up to four split reads (PCI-X)
- Supports PCI access to all of the MV64360/1/2's internal registers.
- PCI address remapping to local memory

PICMG Compact PCI Hot-Swap ready.

"Plug and Play" support:

- Plug and Play compatible configuration registers.
- PCI configuration registers that are accessible from both CPU and PCI
- Vital Product Data (VPD) support.
- PCI Power Management (PMG) support.
- Message Signal Interrupts (MSI) support.

Messaging Unit:

- Doorbell and message interrupts between the CPU and the PCI.
- I_2O support.

Data integrity support between all interfaces.

- ECC support on DRAM, single bit correction, two bits detection.
- DRAM RMW in case of partial write access.
- Parity support on integrated SRAM.
- Parity support on CPU, PCI, and device buses.
- Full error reporting.
- Errors propagation between the different interfaces.

Ethernet MAC Controllers:

- Support 10/100/1000 Mbps.
- MII, GMII, or TBI interface.
- Priority queueing on receive based on DA, VLAN-Tag, IP-TOS.
- Layer2/3/4 frame encapsulation detection.
- Supports long frames (up to 9K) on both receive and transmit.
- TCP/IP checksum on receive and transmit.
- Simplified DA address filtering.

Two Multi-Protocol Serial Controllers (MPSC):

- Each channel supports HDLC, BISYNC, UART, and Transparent protocols.
- Bit rate of up to 55 Mb/s per channel.
- Dedicated SDMAs per each channel.

FEATURES

32 multi-purpose pins dedicated for peripheral functions and general purpose I/O.

- Each pin can be configured independently.
- GPIO inputs can be used to register interrupts from external devices, and generate maskable interrupts.

Four channel Independent DMA controller:

- Chaining via linked-lists of descriptors.
- Moves data from any to any interface.
- DMA trigger by software or external hardware.
- Early DMA termination by software or external hardware.
- Supports increment or hold on both source and destination address

Interrupt controller:

- Maskable interrupts to CPU and PCI.
- Drives up to four interrupt pins.

Four general purpose 32-bit timer/counters.

TWSI interface:

- Master/slave operation.
- Serial ROM initialization.

Two pinout configurations:

- Two 64-bit PCI interfaces plus two Gb ports.
- Two PCI interfaces - one 64-bit and the other 32-bit, and three Gb ports.

724PBGA package, 1mm ball pitch.

Advanced 0.18u process.

Document Status	
Advance Information	This document contains design specifications for initial product development. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.
Preliminary Information	This document contains preliminary data, and a revision of this document will be published at a later date. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.
Final Information	This document contains specifications on a product that is in final release. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.
Revision Code:	
Preliminary	

Table of Contents

List of Tables

List of Figures

Preface

About this Document

This datasheet provides the hardware specifications for the following products:

- MV64360
- MV64361
- MV64362

The word "devices" is used throughout the datasheet when referring to all three parts. This is to avoid repeating the part numbers.

Unless otherwise specified, all of the feature descriptions in this datasheet refer to all three devices. It is explicitly noted in the specific case if information only applies to certain parts and not all three.

Note

The differences between these devices is described in Section 2. "Device Differences" on page 38.

Document Conventions

Document Conventions
The following name and usage conventions are used in this document:

Signal Range	A signal name followed by a range enclosed in brackets represents a range of logically related signals. The first number in the range indicates the most significant bit (MSb) and the last number indicates the least significant bit (LSb). Example: DB_AD[31:0]
Active Low Signals #	A # symbol at the end of a signal name indicates that the signal's active state occurs when voltage is low. Example: INT#

State Names	State names are indicated in *italic* font. Example: *linkfail*
Register Naming Conventions	Register field names are indicated in courier blue font. Example: `RegInit` Register field bits are enclosed in brackets. Example: Field [1:0] Register addresses are represented in hexadecimal format Example: 0x0 Reserved: The contents of the register are reserved for internal use only or for future use.

Abbreviations The following abbreviations are used in this document:			
b	Bit	Mb	Megabit
B	Byte	Mbps	Megabit per second
Gb	Gigabit	MB	Megabyte
GbE	Gigabit Ethernet	MHz	Megahertz
Gbps	Gigabits per second	Mpps	Million packets per second
GB	Gigabyte	ms	Millisecond
Kb	Kilobit	ns	Nanosecond
KB	Kilobyte	V	Volt
KHz	Kilohertz	VDD	Power
mA	Milliampere	Vss	Ground

All register bits/fields have one of the following classifications in the register's Type/Init Val column.

Type	**Description**
CPURI	Initial value is reset (initialized) upon a CPU read.
CPUWI	Initial value is undefined at reset and set by a CPU write.
RES	Reserved for future use. All reserved bits are read as zero unless otherwise noted.
RO	Read Only. Writing to this type of field may cause unpredictable results.
ROC	Read Only Clear. After read, the register field is cleared to zero. Writing to this type of field may cause unpredictable results.
RW	Read and Write with initial value indicated.
RWC	Read/Write Clear on Read. All bits are readable and writable. After reset or after the register is read, the register field is cleared to zero.
RWR	Read/Write Reset. All bits are readable and writable. After reset the register field is cleared to zero.
RWS	Read/Write Set. All bits are readable and writable. After reset the register field is set to a non-zero value specified in the text.
SC	Self-Clear. Writing a one to this register causes the desired function to be immediately executed, then the register field is cleared to zero when the function is complete.
Update	Value written to the register field does not take effect until a soft reset is executed.
WO	Write Only. Reads to this type of register field return undefined data.
WCPU	Undefined at reset, must be set by CPU.
RCPU	Initialized when read by CPU.

Related Documentation

The following documents contain additional information related to the these family of devices.

- MV64360 Functional Errata and Restrictions
- AN-63: Thermal Management for Selected Marvell® Products
- AN-67: Powering Up/Powering Down Marvell® Devices with Multiple Power Supplies of Different Voltages
- AN-84: PCI Ordering Implementation
- AN-85: PCI Deadlock Considerations
- TB-99: Differences Between the MV64360 Stepping A1 and A2

Section 1. Overview

The MV64360/1/2 provides a single-chip high performance solution for PowerPC CPU based applications, such as routers, web switches, storage applications, wireless infrastructure, and many more.

Each of the devices' interfaces use dedicated read and write buffers. The Ethernet MACs and the MPSCs have their own dedicated SDMAs to transfer Rx/Tx data and descriptors to/from memory.

All of these interfaces are connected through a crossbar fabric. The crossbar enables concurrent transactions between units. For example, the crossbar can simultaneously control:

- A Gb Ethernet MAC fetching a descriptor from the integrated SRAM. (Applies only to the MV64360 and MV64361 devices.)
- The CPU reading from the DRAM.
- The DMA moving data from the device bus to the PCI bus.

1.1 CPU Bus Interface

The MV64360/1/2 supports PowerPC CPUs. With a maximum frequency of 133 MHz, the CPU can transfer in excess of 8Gbps.

The MV64360/1/2 supports up to eight pipelined transactions on the CPU bus (up to 16 transactions in dual CPU configuration). For example, if the CPU initiates a data read from the PCI interface and starts a code read from DRAM, the two cycles are pipelined. The CPU interface reads, in parallel, from the PCI interface and from DRAM.

Notes

- When "DRAM" is used in this datasheet, it refers to DDR SDRAM. See Section 11. "DDR SDRAM Controller" on page 129.
- The dual CPU configuration is only supported in the MV64360 and MV64361 devices.

By the time read data is returned from the PCI interface, read data from DRAM is already available – since a DRAM access is faster than a PCI access. In case of interfacing MPC74XX CPU in MPX mode, that supports out-of-order completion, the MV64360/1/2 first drives the DRAM read data on the CPU bus, and then the PCI read data that arrives later. In case the MV64360/1/2 is running in 60x bus mode (which requires in-order completion), it drives DRAM read data immediately after a PCI read data with zero wait states.

The CPU can connect with up to four MV64360/1/2 or other slave devices on the 60x bus. This increases system flexibility. For example, it enables CPU access to more than two PCI-X busses, or more than three Ethernet Gigabit MACs. However, it limits the maximum CPU bus frequency.

The MV64360/1/2 supports 1.8V, 2.5V and 3.3V CMOS levels.

The MV64360/1/2 supports dual CPU configuration, both in 60x and MPX bus modes. It supports MPX bus data intervention (cache-to-cache data transfers) and Data Snarfing (write of the intervention data back to DRAM in parallel to the cache to cache transfer).

Note

Dual CPU configuration is only supported in the MV64360 and MV64361 devices.

The MV64360/1/2 also contains some hardware hooks for multi processing applications, such as dedicated interrupt per CPU, inter-CPUs doorbell interrupts, hardware semaphores, and more.

The MV64360/1/2 fully supports PowerPC cache coherency. DRAM specific address ranges and integrated SRAM can be marked as cache coherent. The MV64360/1/2 maintains cache coherency on PCI, IDMA, Ethernet MAC, or MPSC accesses to these regions. Access to a cache coherent region, results in snoop transaction generated by the MV64360/1/2 on the CPU bus, and in case of hit in a modified cache line, the access is suspended until the cache line copy-back is completed. It supports both the conventional ARTRY# snoop response, and the MPX bus HIT# snoop response, which better utilizes the CPU bus.

The MV64360/1/2 also supports master capability on the 60x bus. It can convert IDMA, Ethernet MAC, MPSC, or PCI transactions to CPU like transactions driven on the bus. This is useful for accessing 60x compliant slave devices seating on the bus.

The MV64360/1/2 supports CPU address remapping to the PCI interface. It also supports access, write, and caching protection, per user specified address ranges.

Note

For further information about the CPU interface, see Section 9. “CPU Interface” on page 98.

1.2 DDR SDRAM Interface

The MV64360/1/2 supports DDR SDRAM (Double Data Rate-Synchronous DRAM). It can run up to 183 MHz with single DRAM load, resulting in 25 Gbps bandwidth. With higher loads, it can run up to 133 MHz, resulting in 16 Gbps.

The DRAM controller supports four DRAM banks. It supports different DDR DIMMs configurations and up to 8 GB DRAM space when using 1Gbx4 DDR DRAM devices (currently unavailable).

The MV64360/1/2 includes an advanced bank interleaving mechanism, enabling maximum utilization of the bus. It also supports up to 16 simultaneously opened pages.

The DRAM controller supports all the functionality required for DRAM interface, including DRAM refresh and DRAM initialization sequence.

Note

For further information about the SDRAM interface, see Section 11. “DDR SDRAM Controller” on page 129.

1.3 Integrated SRAM

Note

Only the MV64360 and MV64361 devices include the integrated SRAM feature. The MV64362 device does not have integrated SRAM.

The MV64360 and MV64361 integrates 2Mb of general purpose SRAM. It is accessible from the CPU or any of the other interfaces. It can be used for off loading DRAM traffic, and as a fast CPU access memory (6-cycle latency).

A typical usage of the SRAM can be a descriptor RAM for the Gb port.

The devices also supports a sophisticated headers retarget mechanism. With this mechanism, packets headers coming from the PCI are retargeted from DRAM space to the SRAM, transparent to the software. This is very useful in communication applications where the processor manipulates only the packet header and requires low latency access to these headers.

Note

For further information about the SRAM interface, see Section 10. “Integrated SRAM” on page 127.

1.4 Device Interface

The MV64360/1/2 device controller supports different types of memory and I/O devices, such as Flash, ROM, SyncBurst SRAM, or an application specific FPGA. It supports flexible timing parameters, that enables access to slow asynchronous devices. On the other hand, it is a 32-bit wide bus, running at 133 MHz, that can be used as a high bandwidth interface to user specific logic.

It supports 8-, 16-, or 32-bit wide devices. It supports five device banks, up to 512 MB address space each.

Note

For further information about the device interface, see Section 12. “Device Controller” on page 147.

1.5 PCI/PCI-X Interface

The MV64360 interfaces two 64-bit PCI/PCI-X busses. The MV64361interfaces two 32-bit PCI/PCI-X busses. And, The MV64362 interfaces one 64-bit PCI/PCI-X bus.

In the commercial devices, the MV64360/1/2 operate at a maximum frequency of 133 MHz when configured to PCI-X mode, and up to 66 MHz when running in conventional PCI mode. The industrial versions operate at a maximum frequency of 125 MHz when configured to PCI-X mode, and up to 66 MHz when running in conventional PCI mode. Each PCI interface can act both as a master initiating a PCI bus transaction or as a target responding to a PCI bus transaction.

The MV64360/1/2 becomes a PCI bus master when the CPU, IDMA, Gigabit Ethernet (GbE) controller or MPSC SDMAs initiates a bus cycle to a PCI device. When running conventional PCI mode, it supports transaction combining, enabling unlimited DMA bursts between PCI and memory. The MV64360/1/2 supports up to four split transactions and write combining in PCI-X mode. It supports all PCI commands including 64-bit addressing using DAC cycles.

The MV64360/1/2 acts as a target when a PCI device initiates a memory access (or an I/O access in the case of internal registers or a P2P transaction). It responds to all memory read/write accesses, including DAC, and to all configuration and I/O cycles, in the case of internal registers. It supports up to four pending delayed reads in conventional PCI mode. It supports up to four split read transactions in PCI-X mode. It is capable of bursting a full 4 KB (without disconnect), both in write transaction, and in read transaction Split Completion.

The MV64360 and MV64361 also support P2P bridging. Each PCI interface can directly transfer memory and I/O transactions to the other PCI interface. Also, type-1 configuration transactions can be transferred to the other PCI interface as a type-1 or type-0 configuration cycle. There is also support for PCI to PCI-X bridging.

Note

The P2P bridging feature is not supported in the MV64362 device.

The PCI slave performs PCI address remapping. It also supports configurable read prefetch, and byte swapping per user specified address ranges.

The MV64360/1/2 PCI interface is fully PCI rev. 2.2 and PCI-X compliant. It contains all the required PCI configuration registers. All internal registers, including the PCI configuration registers, are accessible from the CPU bus or the PCI bus.

The MV64360/1/2 configuration register set is PC Plug and Play compatible. It supports PCI spec rev. 2.2 features such as VPD, MSI, and PMG. It also supports PCI Hot-Plug and CompactPCI Hot-Swap ready.

The MV64360/1/2 also includes a messaging unit to support PCI to CPU messaging in general, and especially the industry standard I^{2}O messaging. The messaging unit includes:

- Two doorbell registers.
- Two message registers.
- Four messages queues located in DRAM.

Note

For further information about the PCI interface, see Section 13. “PCI Interface” on page 158.

1.6 IDMA Engines

The MV64360/1/2 incorporates four IDMA engines. Each IDMA engine has the capability to transfer data between any interface.

The DMA can run in chain mode, via linked list of descriptors and transfer up to 16MB per a single descriptor. The DMA can be controlled from both CPU or external PCI master.

The DMA independently supports increment/hold on source and destination addresses. Source and destination addresses can be non-aligned on any byte address boundary.

The DMA transfer rate can be controlled via DMAReq# pins, when configured to demand mode. This is useful when interfacing devices that do not contain the whole byte count in advance. For example, a serial MAC placed on the device bus triggers DMAReq# whenever it has small chunk of data to be transferred to DRAM.

The DMA also supports some hardware hooks that enable a friendly interface to external devices. It supports a descriptor ownership bit, to enable re-use of descriptors by the CPU, and End of Transfer pins to allow early termination of a DMA transfer, and fetch a new descriptor (e.g. reach end of packet).

Note

For further information about the IDMA engines and operation, see Section 18. “IDMA Controller” on page 301.

1.7 Data Integrity

The MV64360/1/2 supports full data integrity on its different interfaces.

The MV64360/1/2 supports ECC on SDRAM. It supports detection and correction of one error, detection of two errors, and detection of three and four errors, if they are in the same nibble. It supports DRAM read-modify-write for partial writes. In case of a non-correctable ECC error detection, an interrupt is set and the transaction address and data are registered.

The MV64360/1/2 supports parity checking and generation on the PCI bus, CPU bus, device bus, and integrated SRAM. In each of these interfaces, a parity error detection results in an interrupt (if not masked) and registration of the transaction address and data.

ECC/parity errors are optionally propagated between the interfaces. For example, in case a CPU read from DRAM results in detection of uncorrectable ECC error, the MV64360/1/2 may drive the wrong parity with the read data on the CPU bus.

Note

For further information about data integrity operation, see Section 7. “Address Space Decoding” on page 82.

1.7.1 Gigabit Ethernet Ports

The MV64360 has two dedicated interfaces for two of the three MACs. The third MAC is multiplexed on the upper bits of one of the PCI interfaces (if using the third Gb MAC, PCI_1 interface is used as a 32-bit interface).

The MV64361has two dedicated interfaces and the MV64362 only has one interface.

The ports can be configured to 10/100 Mbps MII interface, GMII 1Gbps interface, or TBI Gbps interface.

Receive and transmit buffer management is based on buffer-descriptor linked list. Descriptors and data transfers are performed by the port dedicated SDMA.

The Ethernet port includes advanced DA address filtering on received packets. It also detects packet type/encapsulations, which can be used by the CPU, for packet routing:

- Layer2: BPDU,VLAN (programmable VLAN-Ethertype), Ethernet v2, LLC/SNAP.
- Layer3: IPv4, IPv6 (according to Ethertype), other (no MPLS detection).
- Layer4 (only over IPv4): TCP, UDP, other.

Each port has eight receive priority queues. Queuing is performed based on DA, VLAN-Tag, IP-TOS.

Each port also has eight transmit queues, with programmable band-width distribution between them.

Each port supports long frames, up to 9 KB. It also supports TCP and UDP checksum calculation on receive, and generation on transmit.

Note

For further information about GbE controllers, see Section 15. “Gigabit Ethernet Controller” on page 210.

1.8 Multi-Protocol Serial Controllers

The MV64360/1/2 integrates two Multi-Protocol Serial Controllers (MPSCs). They support UART, HDLC, BISYNC, and transparent protocols. The MPSCs are implemented in hardware. This implementation allows for superior performance versus microcoded implementations.

In HDLC mode, the MPSCs perform all framing operations, such as; bit stuffing/stripping, flag generation, and part of the data link operations (e.g. address recognition functions). The MPSCs directly support common HDLC protocols including those used by ISDN and frame relay.

The MPSCs operation is based on link list of descriptors in memory. They have a dedicated SDMA that handles the descriptors and transmit/receive data transfer. The MPSCs also support simple register based UART operations.

Note

For further information about the MPSCs, seeSection 16. “Multi-Protocol Serial Controllers” on page 252.

1.9 Interrupt Controller

The MV64360/1/2 supports up to four interrupt outputs:

- Two CPU interrupt pins
- Two open drain PCI interrupt pins.

Each interrupt has it’s own mask register. This enables the presentation of different interrupt events on different interrupt pins.

The MV64360/1/2 MPP lines can also be configured to act as interrupt inputs, enabling registration of external interrupts toward the CPU.

Note

For further information about the interrupt controller, see Section 25. “Interrupt Controller” on page 340.

Section 2. Device Differences

This datasheet provides specification details for the MV64360, MV64361, and MV64362 devices.

Table 1 outlines the various interface and feature difference between these three devices.

Note

For further information, see the specific interface section in this document.

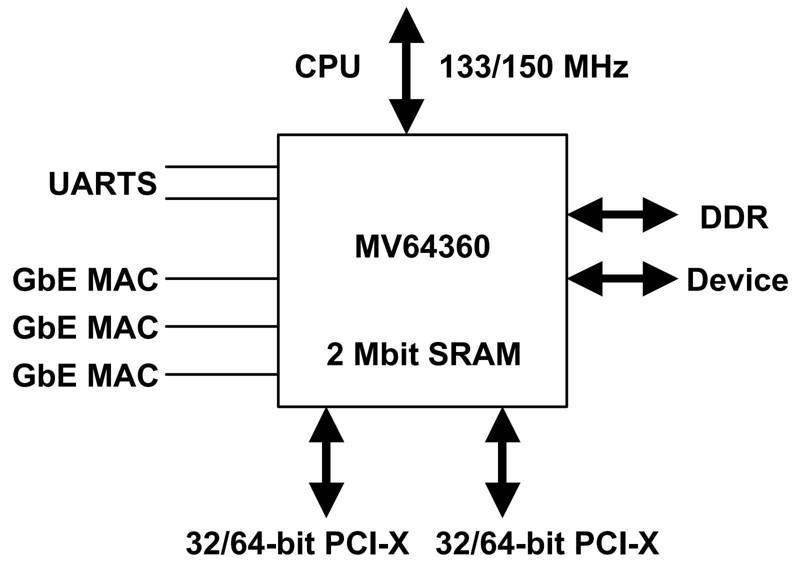
Table 1: Device Architecture

Interface	MV64360	MV64361	MV64362
CPU	• 64-bit bus • Acts as a slave interface, responding to CPU transactions, or as a master interface, generating CPU like transactions. **NOTE:** There is no multi-CPU support in the MV64362 device. •		
DRAM	• Supports up to four DRAM banks (four DRAM chip selects). • 16-bit address bus (DA[13:0] and BA[1:0]) and a 72-bit data bus (DQ[63:0], CB[7:0]).		
Device	• Supports up to five banks of devices. • Each bank supports up to 512 MB of address space, resulting in total device space of 2.5 GB.		
PCI	Supports two 64-bit PCI/ PCI-X interfaces. Supports PCI to PCI, PCI-X to PCI-X, and PCI to PCI-X bridging between the two PCI interfaces.	Supports two 32-bit PCI/ PCI-X interfaces.	Supports one 64-bit PCI/ PCI-X interface.
GbE Port	Implements three Gigabit Ethernet Ports. The third port is multiplexed on the PCI interface's upper bits.	Implements two Gigabit Ethernet Ports.	Implements one Gigabit Ethernet Ports.
Integrated SRAM	Includes 2 Mb of integrated SRAM.		No integrated SRAM
MPSC	Includes two MPSCs that support: • • Bit oriented protocols (e.g. HDLC) • • Byte oriented protocols (e.g. BISYNC) • • Transparent protocols • • The UART (Start/Stop) mode The two MPSCs can operate independently and up to a guaranteed bit rate of 55Mbps.		
IDMA Engine	Four independent IDMA engines.		

The following block diagrams also display the basic interfaces for the three different devices.

Figure 1 shows the MV64360 interfaces.

Figure 1: MV64360 Interfaces



Notes

- The MV64360 has three GbE ports. Two ports have a dedicated interface. The third port is multiplexed on the upper pins of PCI_1 interface.

Figure 2 shows the MV64361 interfaces.

Figure 2: MV64361 Interfaces

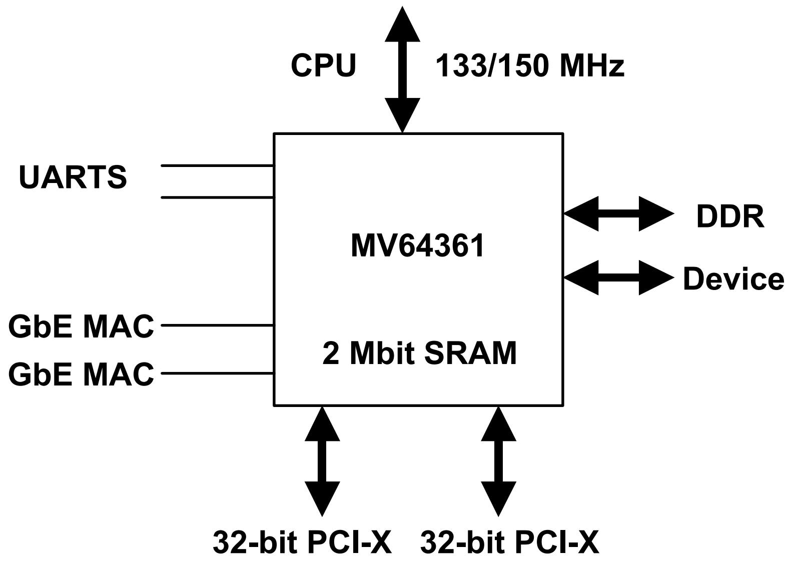
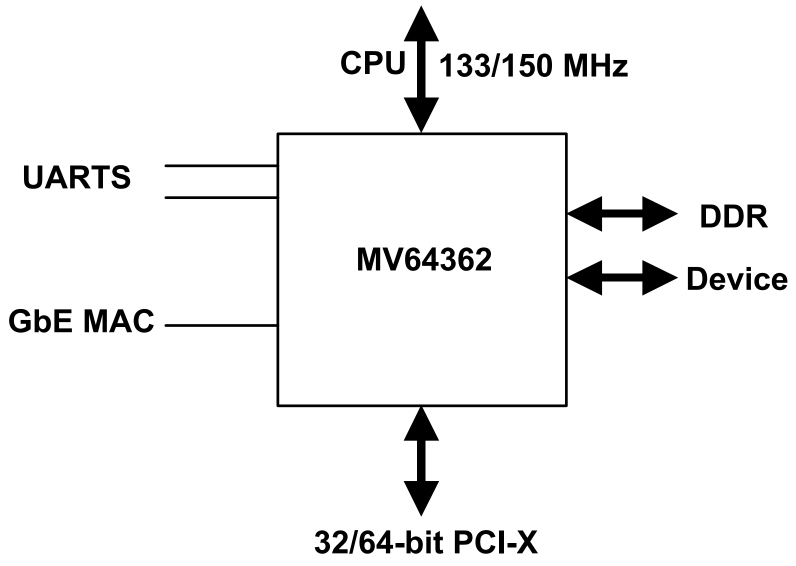


Figure 3 shows the MV64362 interfaces.

Figure 3: MV64362 Interfaces



Section 3. Pin Information

Figure 4: MV64360 Pin List

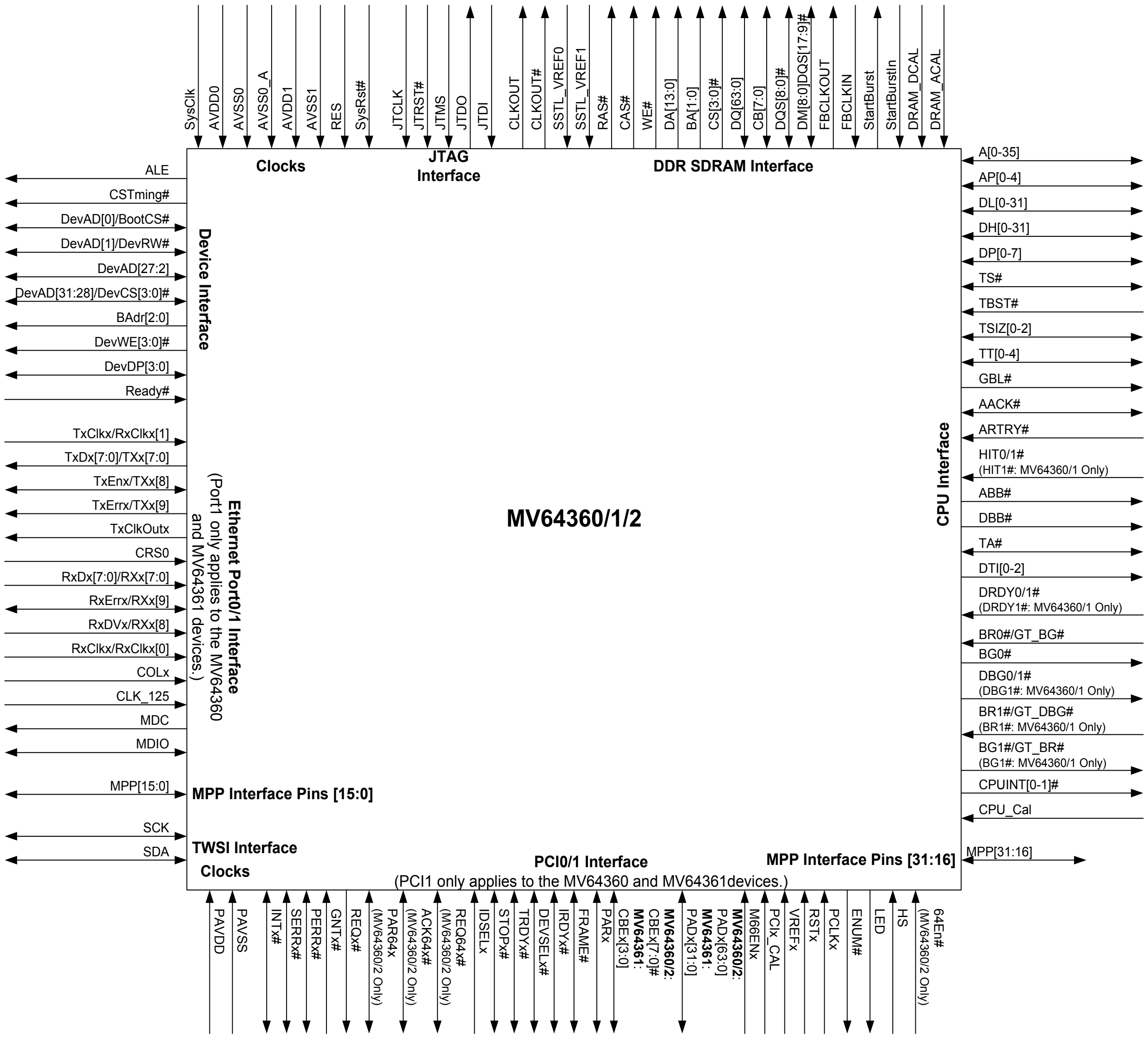


Table 3 lists the conventions that apply to input/output (I/O) or just output (O) type pins.

Table 2: Pin Voltage Levels

Interface	Voltage
CPU Pins	2.5V or 3.3V
PCI VREF	3.3V or 5V
All other signals	3.3V only

Table 3: Pin Assignment Table Conventions

Abbreviation	Description
t/s	Tri-State pin.
s/t/s	Sustained Tri-State pin. Driven to its inactive value for one cycle before float. **NOTE:** A pull up is required to sustain the inactive value.
o/d	Open Drain pin. Allows multiple drivers simultaneously (wire-OR connection). **NOTE:** A pull up is required to sustain the inactive value.

Table 4: Clock Pin Assignments

Pin Name	Type	Full Name	Description
SysClk	I	System Clock	System central clock source. Used as the reference clock for the MV64360/1/2 internal PLL (up to 133MHz). Same clock source is used for the CPU and all other board components.
AVDD0	I	PLL0 AVDD	PLL0 quiet 3.3V power supply.
AVSS0	I	PLL0 AVSS	PLL0 quiet VSS.
AVSS0_A	I	PLL0 AVSS	PLL0 quiet VSS.
AVDD1	I	PLL1 AVDD	PLL1 quiet 3.3V power supply.
AVSS1	I	PLL1 AVSS	PLL1 quiet VSS.
Res	I	PLL Resistor	PLL Resistor (used by both PLL0 and PLL1). Connect to AVSS0 via 12.1 Kohm resistor.
PAVDD	I	PCI AVDD	PCI DLLs quiet 1.8V power supply.
PAVSS	I	PCI AVSS	PCI DLLs quiet VSS.
SysRst#	I	System Reset	Main reset signal of the MV64360/1/2. Resets all units to their initial state. **NOTE:** When in the reset state, all output pins are put into tri-state.
Core Clock Pin Count: 10			

Table 5: CPU Interface Pin Assignments

Pin Name	Type	Full Name	Description
A[0-35]	t/s I/O	Address Bus	A 36-bit CPU (MPC7450) address bus. Valid during the address tenure. Driven by the transaction initiator (whether it is the CPU or the MV64360/1/2). When the MPC7450 extended physical address is disabled, or when interfacing with a different PowerPC CPU, only A[4-35] are used, and A[0-3] must be pulled down. **NOTE:** Requires the following strapping: A[0-3] must be pulled down. A[4-35] must be pulled up.
AP[0-4]	t/s I/O	Address Parity	5-bit odd address parity driven by the transaction initiator during the address tenure. AP[0] corresponds to A[0-4], AP[1] to A[4-11], AP[2] to A[12-19], AP[3] to A[20-27] and AP[4] to A[28-35]. When the MPC7450 extended physical address is disabled, or when interfacing with a non-MPC7450 PowerPC CPU, only AP[1-4] are used, and AP[0-4] must be pulled up. **NOTE:** Requires a pull up
DL[0-31]	t/s I/O	Data-Low Bus	A 32-bit CPU data bus (low 32-bit). Driven by the transaction initiator during the data tenure of a write transaction (whether it is the CPU or the MV64360/1/2). Driven by the target device during the data tenure of a read transaction, (whether it is the MV64360/1/2 or another target device). In case of a dual CPU configuration in MPX bus mode, might also be driven by the peer CPU in case of a hit in an exclusive or a modified cache line (data intervention). **NOTE:** These pins utilize integrated pullups. Dual CPU configuration is only supported in the MV64360 and MV64361 devices.
DH[0-31]	t/s I/O	Data-High Bus	A 32-bit CPU data bus (high 32-bit). Driven by the transaction initiator during the data tenure of a write transaction (whether it is the CPU or the MV64360/1/2). Driven by the target device during the data tenure of a read transaction, (whether it is the MV64360/1/2 or another target device). In case of a dual CPU configuration in MPX bus mode, might also be driven by the peer CPU in case of a hit in an exclusive or a modified cache line (data intervention). **NOTE:** These pins utilize integrated pullups
DP[0-7]	t/s I/O	Data Parity	8-bit odd parity. Driven by the data bus master during data tenure. DP[0] corresponds to DH[0-7], DP[1] to DH[8-15], DP[2] to DH[16-23], DP[3] to DH[24-31], DP[4] to DL[0-7], DP[5] to DL[8-15], DP[6] to DL[16-23] and DP[7] to DL[24-31]. **NOTE:** These pins utilize integrated pullups

Table 5: CPU Interface Pin Assignments (Continued)

Pin Name	Type	Full Name	Description
TS#	t/s I/O	Transfer Start	Asserted during the first address tenure cycle. Driven by the transaction initiator (either it is the CPU or the MV64360/1/2). **NOTE:** Requires a pull up
TBST#	t/s I/O	Transfer Burst	Indicates that a burst transaction is in progress. Valid during the address tenure and driven by the transaction initiator. **NOTE:** Requires a pull up
TSIZ[0-2]	t/s I/O	Transfer Size	Indicates the number of bytes to be transferred. Valid during the address tenure and driven by the transaction initiator. **NOTE:** Requires a pull up
TT[0-4]	t/s I/O	Transfer Type	Indicates transaction attributes (such as read/write and address-only). Valid during the address tenure and is driven by the transaction initiator. **NOTE:** Requires a pull up
GBL#	t/s O	Global	Indicates that the address is global and must be snooped by the CPU. The MV64360/1/2 asserts GBL# on every transaction it drives on the CPU bus. **NOTE:** Requires a pull up
AACK#	t/s I/O[1]	Address Acknowledge	Asserted by the target device to indicate end of the address tenure. In 60x bus mode, the transaction initiator floats the address bus and attribute signals on the next cycle after AACK# assertion. In MPX bus mode, the initiator keeps driving the bus if has bus mastership, and may drive new address tenure (no dead cycle is required between consecutive address tenures). **NOTE:** Requires a pull up
ARTRY#	I	Address Retry	Driven by the CPU. Indicates to the initiator that the transaction must be retried Typically, asserted as a result of an address hit in a modified cache line that needs to be written back to main memory. **NOTE:** Requires a pull up
HIT0#	I	Snoop Hit	Driven by the CPU. Indicates to the MV64360/1/2 that the snooped address (whether it was driven by the CPU1 or the device) hit an exclusive or modified line in the CPU cache, and that the CPU will supply the data latter with using a data only transaction. **NOTE:** Used only in MPX bus configuration. Must be pulled up in 60x bus mode.

Table 5: CPU Interface Pin Assignments (Continued)

Pin Name	Type	Full Name	Description
HIT1#	I	Snoop Hit	Driven by the CPU. Indicates to the MV64360/1/2 that the snooped address (whether it is CPU0 or the device) hit an exclusive or modified line in the CPU cache, and that the CPU will supply the data later with using a data only transaction. **NOTE:** Used only in MPX bus configuration. Must be pulled up in 60x bus mode or if using a single CPU in MPX mode. Hit1# is only valid for the MV64360 and MV64361 devices.
ABB#	t/s O	Address Bus Busy	Asserted during the entire address tenure. In 60x bus mode, driven by the initiator that currently drives the address bus. Not used in MPX bus mode. **NOTE:** Requires a pull up
DBB#	s/t/s O	Data Bus Busy	Asserted during the entire data tenure. Following the cycle after it received DBG#, driven by the initiator that currently has the data bus ownership. **NOTE:** Requires a pull up
NOTE: The MV64360/1/2 supports CPUs such as IBM PPC750Cx and Motorola MPC7450 that do not support ABB# and DBB# signals. It does not sample ABB# and DBB# signals. These signals are driven as outputs.			
TA#	t/s I/O	Transfer Acknowledge	Asserted by the target device to indicate it drives valid read data on the data bus or it sampled write data from the data bus. Used by the initiator to qualify read data and to drive new data during burst write transaction. In MPX bus mode dual CPU configuration (MV64360 and MV64361), also used as a qualifier for one CPU to sample data driven by the peer CPU (data intervention).
DTI[0-2]	t/s O	Data Transfer Index[0-2]	Driven by the MV64360/1/2 to indicate data tenure index (out of order support). In MPX bus mode dual CPU configuration, used as data tenure index to both CPUs. The MV64360/1/2 might drive different value to each CPU (CPU DTI is qualified with its own DBG# signal) **NOTE:** Used only in MPX bus configuration. MV64360/1/2 floats DTI in 60x bus mode. When interfacing MPC7450 that has a 4-bit DTI interface, connect the device's DTI[0-2] to CPU DTI[1-3]. Connect CPU DTI[0] to VSS.
DRDY0#	I	Data Only Request	Asserted by the CPU to indicate that it has data ready for intervention. Used as an implicit CPU0 data bus request in order to perform data only transaction. **NOTE:** Used only in MPX bus configuration. Must be pulled up in 60x bus mode.

Table 5: CPU Interface Pin Assignments (Continued)

Pin Name	Type	Full Name	Description
DRDY1#	I	Data Only Request	Asserted by the CPU to indicate that it has data ready for intervention. Used as an implicit CPU1 data bus request in order to perform data only transaction. **NOTE:** Used only in MPX bus configuration. Must be pulled up in 60x bus mode or if using a single CPU in MPX mode. DRDY1# is only valid for the MV64360 and MV64361 devices.
BR0#/GT_BG#	I	Bus Request0	If the MV64360/1/2 CPU bus arbiter is enabled, used as CPU bus request input. **NOTE:** When running in MPX bus mode, the device's CPU bus arbiter must be used.
		MV64360/1/2 Bus Grant	If the MV64360/1/2 CPU bus arbiter is disabled, used as bus grant input for MV64360/1/2 bus master transactions.
BG0#	O	Bus Grant0	If the MV64360/1/2 CPU bus arbiter is enabled, used as CPU bus grant output. **NOTE:** When running in MPX bus mode, the device's CPU bus arbiter must be used.
DBG0#	O	Data Bus Grant0	If the MV64360/1/2 CPU bus arbiter is enabled, used as CPU data bus grant output. **NOTE:** When running in MPX bus mode, the device's CPU bus arbiter must be used.
BR1#/ GT_DBG#	I	Bus Request1	If the MV64360/1/2 CPU bus arbiter is enabled, used as a second CPU bus request input. If having only a single CPU, must be pulled up. **NOTE:** When running in MPX bus mode, the device's CPU bus arbiter must be used. Must be pulled up when interfacing a single CPU. BR1# is only valid for the MV64360 and MV64361 devices.
		MV64360/1/2 Data Bus Grant	If the MV64360/1/2 CPU bus arbiter is disabled, used as data bus grant input for the device's bus master transactions.
BG1#/GT_BR#	t/s O	Bus Grant1	If the MV64360/1/2 CPU bus arbiter is enabled, used as a second CPU bus grant output. **NOTE:** When running in MPX bus mode, the device's CPU bus arbiter must be used. BG1# is only valid for the MV64360 and MV64361 devices.
		MV64360/1/2 Bus Request	If the MV64360/1/2 CPU bus arbiter is disabled, used as bus grant input for the MV64360/1/2 bus master transactions.

Table 5: CPU Interface Pin Assignments (Continued)

Pin Name	Type	Full Name	Description
DBG1#	t/s O	Data Bus Grant1	If the MV64360/1/2 CPU bus arbiter is enabled, used as a second CPU data bus grant output. **NOTE:** When running in MPX bus mode, the device's CPU bus arbiter must be used. DBG1# is only valid for the MV64360 and MV64361 devices.
CPUINT[0-1]#	t/s O	CPU Interrupts	Level sensitive interrupts driven by the MV64360/1/2 to the CPU.
CPU_Cal	I	CPU Channel Calibration	Allows control of the CPU output buffers' strength. Connect to VDD through a resistor. The resistor size determines the Channel drive of the output buffer.
CPU Interface Pin Count: 145 (MV64360 and MV64361),			

1. In multi-MV mode, acts as s/t/s pin.

Table 6: PCI Bus 0 Interface Pin Assignments

NOTE: The "_0" extension to the PCI label only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
PCLK0	I	PCI_0 Clock	PCI bus clock. Up to 133 MHz in PCI-X mode and up to 66MHz in conventional PCI mode. It is completely independent of PCLK1, and SYSCLK.
VREF0	I	PCI_0 VREF	PCI reference input voltage (3.3V or 5V)
PCI0_CAL	I	PCI_0 Channel Calibration	Allows control of the PCI output buffers' strength. Connect to VDD through a resistor. The resistor size determines the channel drive of the output buffer.
RST0#	I	PCI_0 Reset	Dedicated reset signal for PCI_0 interface. This signal does not affect any other unit. In the MV64360 and MV64361devices, this includes the PCI_1 interface. When in the reset state, all PCI_0 output pins are put into tri-state and all open drain signals are floated.
M66EN0	I	PCI_0 66MHz Enable	The MV64360/1/2 samples M66EN0 on reset de-assertion to determine if it is connected to a 66MHz bus. If M66EN0 is sampled '1', the internal PCI interface DLL is enabled. **NOTE:** Only relevant to conventional PCI (if configured to PCI-X, the internal PCI interface DLL is enabled regardless of M66EN0).

Table 6: PCI Bus 0 Interface Pin Assignments (Continued)
NOTE: The "_0" extension to the PCI label only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
MV64360 and MV64362: PAD0[63:0] **MV64361**: PAD0[31:0]	t/s I/O	PCI_0 Address/Data	MV64360 and MV64362: 64-bit PCI_0 multiplexed address/data bus. MV64361: 32-bit PCI_0 multiplexed address/data bus. Driven by the transaction master during address phase and write data phase. Driven by the target device during read data phase. **NOTE:** The PAD0[63:32] signals are only valid for the MV64360 and MV64362 devices and require a pull up. When the PCI_0 interface is configured to 32-bit, the device drives these pins; a pull up is not required.
MV64360 and MV64362: CBE0[7:0]# **MV64361**: CBE0[3:0]#	t/s I/O	PCI_0 Command/Byte Enable	MV64360 and MV64362: 8-bit multiplexed command/byte-enable bus, driven by transaction master. MV64361: 4-bit multiplexed command/byte-enable bus, driven by transaction master. Contains command during the address phase and byte-enable during data phase. **NOTE:** The CBE0[7:4]# signals are only valid for the MV64360 and MV64362 devices and require a pull up. When the PCI_0 interface is configured to 32-bit, the device drives these pins; a pull up is not required.
PAR0	t/s I/O	PCI_0 Parity (low)	Even parity calculated for PAD[31:0] and CBE[3:0]. Driven by transaction master for address phase and write data phase. Driven by target for read data phase.
FRAME#	s/t/s I/O	PCI_0 Frame	Asserted by the transaction master to indicate the beginning of a transaction. The master de-asserts FRAME# to indicate that the next data phase is the final data phase transaction.
IRDY0#	s/t/s I/O	PCI_0 Initiator Ready	Asserted by the transaction master to indicate it is ready to complete the current data phase of the transaction. A data phase is completed TRDY# and IRDY# are asserted.
DEVSEL0#	s/t/s I/O	PCI_0 Device Select	Asserted by the target of the current access. As a master, the MV64360/1/2 expects the target to assert DEVSEL# within five bus cycles. If the target does not assert DEVSEL# within the required bus cycles, the device aborts the cycle. As a target, the device asserts DEVSEL# in a medium speed; two cycles after the assertion of FRAME#.
TRDY0#	s/t/s I/O	PCI_0 Target Ready	Asserted by the target to indicate it is ready to complete the current data phase of the transaction. A data phase is completed when TRDY# and IRDY# are asserted.

Table 6: PCI Bus 0 Interface Pin Assignments (Continued)

NOTE: The "_0" extension to the PCI label only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
STOP0#	s/t/s I/O	PCI_0 Stop	Asserted by target to indicate transaction termination. Used by a target device to generate a Retry, Disconnect, or Target Abort termination signal.
IDSEL0	I	PCI_0 Initialization Device Select	Asserted to act as a target device chip select during PCI configuration transactions.
REQ640#	s/t/s I/O	PCI_0 Request 64-bit Transfer	**NOTE:** Only valid for the MV64360 and MV64362 devices. Asserted by the transaction master to indicate a request of a 64-bit bus width transaction. REQ64# timing is the same as FRAME# timing. **NOTE:** A 64-bit transaction occurs when REQ64# and ACK64# are asserted.
ACK640#	s/t/s I/O	PCI_0 Acknowledge 64-bit Transfer	**NOTE:** Only valid for the MV64360 and MV64362 devices. Asserted by the target in response to REQ64# to indicate it accepts a 64-bit bus width transaction. ACK64# timing is the same as DEVSEL# timing. **NOTE:** A 64-bit transaction occurs when REQ64# and ACK64# are asserted.
PAR640	t/s I/O	PCI_0 Parity (high)	**NOTE:** Only valid for the MV64360 and MV64362 devices. In cases of a 64-bit PCI transaction, even parity is calculated for PAD[63:32] and CBE[7:4]. Driven by the transaction master for address phase and write data phase. Driven by the target for read data phase. **NOTE:** PAR640 requires a pull up. When PCI_0 interface is configured to 32-bit, the device drives this pin; a pull up is not required.
REQ0#	t/s O	PCI_0 Bus Request	In a case using an external PCI arbiter, asserted by the MV64360/1/2 PCI master to indicate it requires PCI bus mastership to initiate a new transaction. **NOTE:** When using the internal PCI arbiter, leave unconnected
GNT0#	I	PCI_0 Bus Grant	In a case using an external PCI arbiter, asserted to indicates to the MV64360/1/2 PCI master that bus mastership is granted. **NOTE:** When using the internal PCI arbiter, a pull down is required.
PERR0#	s/t/s I/O	PCI_0 Parity Error	Asserted when a data parity error is detected. Asserted by a target device in response to bad address or write data parity, or by master device in response to bad read data parity.
SERR0#	o/d O	PCI_0 System Error	Asserted when a serious system error (not necessarily a PCI error) is detected.

Table 6: PCI Bus 0 Interface Pin Assignments (Continued)
NOTE: The "_0" extension to the PCI label only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
INT0#	o/d O	PCI_0 Interrupt Request	Asserted by the MV64360/1/2 when one of the unmasked internal interrupt sources is asserted. **NOTE:** If MSI is enabled, MV64360/1/2 never asserts INT0
PCI Bus 0 Interface Pin Count: 92 (MV64360 and MV64362), 53 (MV64361)			

Table 7: PCI Bus 1 Interface Pin Assignments
NOTE: This interface only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
PCLK1	I	PCI_1 Clock	PCI bus clock. Up to 133MHz in PCI-X mode and up to 66MHz in conventional PCI mode. It is completely independent of PCLK0 and SYSCLK.
VREF1	I	PCI_1 VREF	PCI reference input voltage (3.3V or 5V)
PCI1_CAL	I	PCI_1 Channel Calibration	Allows control of the PCI output buffers' strength. Connect to VDD through a resistor. The resistor size determines the Channel drive of the output buffer.
RST1#	I	PCI_1 Reset	Dedicated reset signal for PCI_1 interface. This signal does not affect any other unit. In the MV64360 and MV64361 devices, this includes the PCI_0 interface. When in the reset state, all PCI output pins are put into tri-state and all open drain signals are floated.
M66EN1	I	PCI_1 66MHz Enable	The MV64360 and MV64361 M66EN1 on reset de-assertion, in order to determine if it is connected to a 66MHz bus. If M66EN1 is sampled '1', the internal PCI interface DLL is enabled. **NOTE:** Only relevant to conventional PCI (if configured to PCI-X, the internal PCI interface DLL is enabled regardless of M66EN1).
MV64360: PAD1[63:0] **MV64361**: PAD1[31:0]	t/s I/O	PCI_1 Address/Data	MV64360: 64-bit PCI_1 multiplexed address/data bus. MV64361: 32-bit PCI_1 multiplexed address/data bus. Driven by the transaction master during address phase and write data phase. Driven by the target device during read data phase. **NOTE:** The PAD0[63:32] signals are only valid for the MV64360 devices and require a pull up. When the PCI_1 interface is configured to 32-bit, the device drives these pins; a pull up is not required.

Table 7: PCI Bus 1 Interface Pin Assignments (Continued)
NOTE: This interface only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
MV64360: CBE1[7:0]# **MV64361**: CBE1[3:0]#	t/s I/O	PCI_1 Command/Byte Enable	MV64360: 8-bit multiplexed command/byte-enable bus, driven by transaction master. MV64361: 4-bit multiplexed command/byte-enable bus, driven by transaction master. Contains command during the address phase and byte-enable during data phase. **NOTE:** The CBE0[7:4]# signals are only valid for the MV64360 devices and require a pull up. When the PCI_0 interface is configured to 32-bit, the device drives these pins; a pull up is not required.
PAR1	t/s I/O	PCI_1 Parity (low)	Even parity calculated for PAD[31:0] and CBE[3:0]. Driven by the transaction master for the address phase and write data phase. Driven by the target for the read data phase.
FRAME#	s/t/s I/O	PCI_1 Frame	Asserted by the transaction master to indicate the beginning of a transaction. The master de-asserts FRAME# to indicate that the next data phase is the final data phase transaction.
IRDY1#	s/t/s I/O	PCI_1 Initiator Ready	Asserted by the transaction master to indicate that it is ready to complete the current data phase of the transaction. A data phase is completed when both TRDY# and IRDY# are asserted.
DEVSEL1#	s/t/s I/O	PCI_1 Device Select	Asserted by the target of the current access. As a master, the MV64360 and MV64361expects the target to assert DEVSEL# within five bus cycles. If the target does not assert DEVSEL# within the required bus cycles, the device aborts the cycle. As a target, the device asserts DEVSEL# in a medium speed, two cycles after the assertion of FRAME#.
TRDY1#	s/t/s I/O	PCI_1 Target Ready	Asserted by the target to indicate it is ready to complete the current data phase of the transaction. A data phase is completed when both TRDY# and IRDY# are asserted.
STOP1#	s/t/s I/O	PCI_1 Stop	Asserted by the target to indicate transaction termination. STOP# is used by a target device to generate a Retry, Disconnect, or Target Abort termination.
IDSEL1	I	PCI_1 Initialization Device Select	Asserted to act as a target device chip select during the PCI configuration transactions.

Table 7: PCI Bus 1 Interface Pin Assignments (Continued)
NOTE: This interface only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
REQ641#	s/t/s I/O	PCI_1 Request 64-bit Transfer	**NOTE:** Only valid for the MV64360 device. Asserted by the transaction master to indicate a request of a 64-bit bus width transaction. REQ64# timing is the same as FRAME# timing. **NOTE:** A 64-bit transaction occurs when REQ64# and ACK64# are asserted.
ACK641#	s/t/s I/O	PCI_1 Acknowledge 64-bit Transfer	**NOTE:** Only valid for the MV64360 device. Asserted by the target in response to REQ64# to indicate it accepts a 64-bit bus width transaction. ACK64# timing is the same as DEVSEL# timing. **NOTE:** A 64-bit transaction occurs when REQ64# and ACK64# are asserted.
PAR641	t/s I/O	PCI_1 Parity (high)	**NOTE:** Only valid for the MV64360 device. In cases of 64-bit PCI transactions, even parity is calculated for PAD[63:32] and CBE[7:4]. Driven by the transaction master for address phase and write data phase. Driven by the target for read data phase. **NOTE:** PAR641 requires a pull up. When PCI_1 interface is configured to 32-bit, the MV64360 drives this pin; a pull up is not required
REQ1#	t/s O	PCI_1 Bus Request	In a case using an external PCI arbiter, asserted by the for the MV64360 device PCI master to indicate it requires the PCI bus mastership to initiate a new transaction. **NOTE:** When using the internal PCI arbiter, leave unconnected.
GNT1#	I	PCI_1 Bus Grant	In a case using an external PCI arbiter, asserted to indicate to the MV64360 and MV64361 PCI master that bus mastership is granted. **NOTE:** When using the internal PCI arbiter, a pull down is required.
PERR1#	s/t/s I/O	PCI_1 Parity Error	Asserted when a data parity error is detected. Asserted by a target device in response to bad address or write data parity, or by a master device in response to bad read data parity.
SERR1#	o/d O	PCI_1 System Error	Asserted when a serious system error (not necessarily a PCI error) is detected.
INT1#	o/d O	PCI_1 Interrupt Request	Asserted by the MV64360 and MV64361 when one of the unmasked internal interrupt sources is asserted. **NOTE:** If MSI is enabled, device never asserts INT1.
PCI Bus 1 Interface Pin Count: 93 (MV64360), 53 (MV64361)			

Table 8: PCI CompactPCI Hot Swap Pin Assignments

Pin Name	Type	Full Name	Description
ENUM#	o/d O	Compact PCI Hot Swap ENUM# interrupt	If ENUM is enabled, asserted by the MV64360/1/2 during hot swap insertion or removal.
LED	t/s O	Compact PCI Hot Swap LED	Driven by the MV64360/1/2 to turn the LED on/off.
HS	I	Compact PCI Hot Swap Handle Switch	Sampled handle switch status to identify board insertion/ removal. **NOTE:** If not using CompactPCI Hot Swap, must be tied to VSS.
64En#	I	Compact PCI Hot Swap 64-bit PCI Enable	**NOTE:** Only valid for the MV64360 and MV64362 devices. The MV64360 and MV64362 samples the 64EN# pin on reset de-assertion, rather then REQ64#, to determine whether it is connected to a 64-bit PCI bus.
CompactPCI Hot Swap Pin Count: 4 (MV64360 and MV64362), 3 (MV64361)			

Table 9: DDR SDRAM Interface Pin Assignments

Pin Name	Type	Full Name	Description
CLKOUT	O	SDRAM Clock	Must be connected to the SDRAM CLK input.
CLKOUT#	O	SDRAM Clock	Must be connected to the SDRAM CLK input.
SSTL_VREF0	I	SSTL2 VREF	Reference voltage for SSTL interface. Must be half of the voltage for the V_{dd} SDRAM @ SSTL mode, see EIA/JEDEC standard EIA/JESD8-9 (Stub series terminated logic for 2.5 volts, SSTL_2).
SSTL_VREF1	I	SSTL2 VREF	Reference voltage for SSTL interface. Must be half of the voltage for the V_{dd} SDRAM @ SSTL mode, see EIA/JEDEC standard EIA/JESD8-9 (Stub series terminated logic for 2.5 volts, SSTL_2).
NOTE: SSTL_VREF0 [D24] and SSTL_VREF1 [E10] are connected to the same VREF rail. Hence, the balls are connected to the same voltage.			
RAS#	O	SDRAM Row Address Select	Asserted by the MV64360/1/2 to indicate an active ROW address driven on the DRAM address lines.
CAS#	O	SDRAM Column Address Select	Asserted by the MV64360/1/2 to indicate an active column address driven on the DRAM address lines.
WE#	O	SDRAM Write Enable	Asserted by the MV64360/1/2 to indicate a write command to the SDRAM.
DA[13:0]	O	SDRAM Address	Driven by the MV64360/1/2 during RAS# and CAS# cycles to generate along with bank address bits 28-bit SDRAM address.

Table 9: DDR SDRAM Interface Pin Assignments (Continued)

Pin Name	Type	Full Name	Description
BA[1:0]	O	SDRAM Bank Address	Driven by the MV64360/1/2 during RAS# and CAS# cycles to select one of the 4 DRAM virtual banks.
CS[3:0]#	O	SDRAM Chip Selects	Asserted by the MV64360/1/2 to select a specific SDRAM bank.
DQ[63:0]	t/s I/O	SDRAM Data Bus	Driven by the MV64360/1/2 during write to SDRAM. Driven by SDRAM during reads.
CB[7:0]	t/s I/O	SDRAM ECC byte	Driven by the MV64360/1/2 during write to SDRAM. Driven by SDRAM during reads.
DQS[8:0]#	t/s I/O	SDRAM Data Strobe	Driven by the MV64360/1/2 during write to SDRAM. Driven by SDRAM during reads.
DM[8:0]/ DQS[17:9]#	ts I/O	SDRAM Data Mask	Asserted by the MV64360/1/2 to select the specific bytes out of the 72-bit data/ECC to be written to the SDRAM.
		SDRAM Data Strob	Additional 9-bit DQS# pins when interfacing x4 DDR devices. **NOTE:** When interfacing x4 devices, data mask (DM) is not supported.
FBCLKOUT	O	SDRAM Clock Feedback Output	Duplication of CLKOUT. **NOTE:** Must be routed on board all the way to the DRAM, and back to the MV64360/1/2 as FBCLKIN.
FBCLKIN	I	SDRAM Clock Feedback Input	FBCLKOUT routed back into the MV64360/1/2 and used as the read data's sampling clock, driven by the SDRAM.
StartBurst	O	Start Burst	MV64360/1/2 indication of starting a burst. Asserted with the first CAS# cycle of DRAM access. **NOTE:** Must be routed on board all the way to the DRAM, and back to the MV64360/1/2 as STARTBURSTIN.
StartBurstIn	I	Start Burst Input	STARTBURST signal routed back to MV64360/1/2. Used as a reference signal for the incoming read data driven by the SDRAM.
DRAM_DCAL	I	DRAM D-Channel Calibration	Allows control of the DRAM output buffers' strength. Connect to VDD through a resistor. The resistor size determines the D-Channel drive of the output buffer.
DRAM_ACAL	I	DRAM A-Channel Calibration	Allows control of the DRAM output buffers' strength. Connect to VDD through a resistor. The resistor size determines the A-Channel drive of the output buffer.
SDRAM Interface Pin Count: 123			

Table 10: Device Interface Pin Assignments

Pin Name	Type	Full Name	Description
ALE	t/s O	Device Address Latch Enable	Used to latch the Address, BOOTCS#, CS[3:0]#, and DEVRW# signals from the AD bus.
CSTming#	t/s O	Device Chip Select Timing	Active for the entire device transaction. Used to qualify DEVRW#, CS[3:0]#and BOOTCS# signals. **NOTE:** This pin is in High-Z during reset assertion and for two cycles after reset de-assertion. A pull up may be added to avoid an erroneous qualification of the CS#[3:0] signals.
DevAD[0]/ BootCS#	t/s I/O	Device Data[0]	Used as device data bit[0] during the data phase. Driven by MV64360/1/2 on write access, and by the device on read access.
		Boot Chip Select	Used as boot device chip select during address phase.
DevAD[1]/ DevRW#	t/s I/O	Device Data[1]	Used as device data bus bit [1] during the data phase. Driven by MV64360/1/2 on write access, and by the device on read access.
		Device Write	Used as device read (‘1’) or write (‘0’) indication during address phase.
DevAD[27:2]	t/s I/O	Device Address	AD[27:2] are used as device address during the address phase. Need to be latched by an external device, using ALE signal. The latched AD[27:2] along with BADR[2:0] are used as the device address.
		Device Data	Used as device data bus during the data phase. Driven by MV64360/1/2 on write access, and by the device on read access.
DevAD[31:28]/ DevCS[3:0]#	t/s I/O	Device Data[31:28]	Used as device data bus bits[31:28] during the data phase. Driven by MV64360/1/2 on write access, and by the device on read access.
		Device Chip Select [3:0]	Used as device chip select [3:0] during address phase.
BAdr[2:0]	t/s O	Device Burst Address	Driven by the MV64360/1/2 during burst read/write transactions to a device. **NOTE:** The MV64360/1/2 increments the burst address with each data transfer.
DevWE[3:0]#	t/s O	Device Write Byte Enables	Asserted by the MV64360/1/2 to select the specific bytes out of the 32-bit AD bus to be written.

Table 10: Device Interface Pin Assignments (Continued)

Pin Name	Type	Full Name	Description
DevDP[3:0]	t/s I/O	Device Parity	Device parity (configurable even or odd parity). DP[0] corresponds to AD[7:0], DP[1] to AD[15:8], DP[2] to AD[23:16] and DP[3] to AD[31:24]. Driven by MV64360/1/2 during address phase and write data phase, and by the device on read data phase.
Ready#	I	Device Ready:	Used as cycle extender when interfacing a slow device. When inactive during a device access, access is extended until Ready# assertion. **NOTE:** If not using Ready#, tie to VSS.
Device Interface Pin Count: 46			

Table 11: Ethernet Port_0 Interface Pin Assignments

NOTE: The "_0" extension to the PCI label only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
TxClk0/ RxClk0[1]	I	Transmit Clock	**For GMII operating in 1000:** This pin is not used and must be pulled down. **For GMII PHYs operating in 10/100:** This pin provides the timing reference for the transmission of the TxEn0,TxD0[3:0] signals. It operates at 2.5 MHz when in 10 Mbps, and at 25 MHz in 100 Mbps speed.
		Receive Clock	**For the 10-bit interface:** This pin provides the receive code group complimentary 62.5 MHz clock input. Both phases are used to strobe Rx0[9:0] inputs.
TxD0[7:0]/ Tx0[7:0]	O	Transmit Data	**For GMII PHYs operating in 1000:** TxD0[7:0] contain the transmit data outputs and are synchronous to the TxClkOut0 output. **For GMII PHYs operating in 10/100:** TxD0[3:0] contain the transmit data outputs and are synchronous to the TxClk0 input. Pins TxD0[7:4] are undefined and must Not be Connected (NC).
		Transmit Data	**For the10-bit interface:** Tx0[7:0] are bits [7:0] of the Tx_code_group[9:0] output to the transceiver. They are synchronous to TxClkOut0.
TxEn0/Tx0[8]	O	Transmit Enable	**For GMII PHYs operating in 10/100/1000:** TxEn0 indicates that the packet is being transmitted to the PHY. It Is synchronous to TxClkOut0 in GMII mode and to TxClk0 in MII mode.
		Transmit Data	**For the 10-bit interface:** Tx0[8] is bit 8 of the Tx_code_group[9:0] output to the transceiver. It is synchronous to TxClkOut0.

Table 11: Ethernet Port_0 Interface Pin Assignments (Continued)
NOTE: The "_0" extension to the PCI label only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
TxErr0/Tx0[9]	O	Transmit Error	**For GMII PHYs operating in 1000:** TxErr0 is used for implementing carrier extension in half-duplex. Is synchronous to TxClkOut0. **For GMII PHYs operating in 10/100:** This output is irrelevant.
		Transmit Data	**For the 10-bit interface:** Tx0[9] is bit 9 of the Tx_code_group[9:0] output to the transceiver. It is synchronous to TxClkOut0. This pin is only relevant for the 10-bit interface.
TxClkOut0	O	Transmit Clock Output	**For GMII PHYs operating in 1000:** Provides the timing reference for the transfer of the TxEn0, TxErr0, and TxD0[7:0] signals. It operates at 125 MHz. **For GMII PHYs operating in 10/100:** This pin is irrelevant and must Not be Connected (NC). **For the 10-bit interface**: This pin is the PMA Transmit Clock. It provides the timing reference for the transfer of the Tx0[9:0] signals. It operates at 125 MHz.
CRS0	I	Carrier Sense	**For GMII PHYs:** Indicates that either the transmit or receive medium is non-idle. CRS0 is not synchronous to any clock. **For the 10-bit interface:** This pin is not used and must be pulled down.
RxD0[3:0] / Rx0[3:0]	I	Receive Data	**For GMII PHYs operating in 10/100:** RxD0[3:0] contains the receive data inputs and is synchronous to RxClk input.
		Receive Data	**For the 10-bit interface**: Rx0[3:0] are bits [3:0] of the Rx_code_group[9:0] input from the transceiver. They are synchronous to RxClk0[1:0].
RxD0[7:4]/ Rx0[7:4]	I	Receive Data	**For GMII PHYs operating in 1000:** RxD0[7:4] contains the receive data inputs and are synchronous to RxClk input. **For GMII PHYs operating in 10/100:** RxD0[7:4] must be held low.
		Receive Data	**For the 10-bit interface**: Rx0[7:4] are bits [7:4] of the Rx_code_group[9:0] input from the transceiver. They are synchronous to RxClk0[1:0].
RxErr0/Rx0[9]	I	Receive Error	**For GMII PHYs operating in 1000:** Also used for implementing carrier extension in half-duplex. Synchronous to RxClk0[1:0]. **For GMII PHYs operating in 10/100:** Synchronous to RxClk input.
		Receive Data	**For the 10-bit interface:** Rx0[9] is bit 9 of the Rx_code_group[9:0] input from the transceiver. It is synchronous to RxClk0[1:0]. This pin is only relevant for the 10-bit interface.

Table 11: Ethernet Port_0 Interface Pin Assignments (Continued)
NOTE: The "_0" extension to the PCI label only applies to the MV64360 and MV64361 devices.

Pin Name	Type	Full Name	Description
RxDV0/Rx0[8]	I	Receive Data Valid	**For GMII PHYs:** Indicates that valid data is present on the RxD0 lines. It is synchronous to RxClk0.
		Receive Data	**For the 10-bit interface:** Rx0[8] is bit [8] of the Rx_code_group [9:0] input from the transceiver. It is synchronous to RxClk0 [1:0].
RxClk0/ RxClk0[0]	I	Receive Clock	**For GMII mode operating in 1000 (RxClk0):** Provides the timing reference for the reception of the RxDv0, RxErr0, and RxD0[7:0] signals. It operates at 125 MHz. **For GMII mode operating in 10/100 (RxClk0):** Provides the timing reference for the reception of the RxDv0, RxErr0, and RxD0[3:0] signals. It operates at 2.5 MHz when in 10 Mbps, and at 25 MHz in 100 Mbps speed. **For the 10-bit interface (RxClk0[0]):** This pin is the receive code group complimentary 62.5 MHz clock input. Both phases are used to strobe Rx0[9:0] inputs.
COL0	I	Collision Detect	**For GMII mode operating in 100 (RxClk0):** Indicates a collision has been detected on the wire. This input is ignored in full-duplex mode. COL0 is not synchronous to any clock. **For GMII mode operating in 100 (RxClk0):** This pin is not used and must be pulled down.
Device Interface Pin Count: 25			

Table 12: Ethernet Port_1 Interface Pin Assignments
NOTE: This interface only applies to the MV64360 and MV64361 devices. Still, the MV64362 device uses the TxD1[4:1] signals in its reset sampling configuration. For further information see, Table 117 on page 348.

Pin Name	Type	Full Name	Description
TxClk1/ RxClk1[1]	I	Transmit Clock	**For GMII operating in 1000:** This pin is not used and must be pulled down. **For GMII PHYs operating in 10/100:** This pin provides the timing reference for the transmission of the TxEn1,TxD1[3:0] signals. It operates at 2.5 MHz when in 10 Mbps, and at 25 MHz in 100 Mbps speed.
		Receive Clock	**For the 10-bit interface:** This pin provides the receive code group complimentary 62.5 MHz clock input. Both phases are used to strobe Rx1[9:0] inputs.

Table 12: Ethernet Port_1 Interface Pin Assignments (Continued)

NOTE: This interface only applies to the MV64360 and MV64361 devices. Still, the MV64362 device uses the TxD1[4:1] signals in its reset sampling configuration. For further information see, Table 117 on page 348.

Pin Name	Type	Full Name	Description
TxD1[7:0]/ Tx1[7:0]	O	Transmit Data	**For GMII PHYs operating in 1000:** TxD1[7:0] contain the transmit data outputs and are synchronous to the TxClkOut1 output. **For GMII PHYs operating in 10/100:** TxD1[3:0] contain the transmit data outputs and are synchronous to the TxClk1 input. Pins TxD1[7:4] are undefined and must Not be Connected (NC).
		Transmit Data	**For the10-bit interface:** Tx1[7:0] are bits [7:0] of the Tx_code_group[9:0] output to the transceiver. They are synchronous to TxClkOut1.
TxEn1/Tx1[8]	O	Transmit Enable	**For GMII PHYs operating in 10/100/1000:** TxEn1 indicates that the packet is being transmitted to the PHY. It Is synchronous to TxClkOut1 in GMII mode and to TxClk1 in MII mode.
		Transmit Data	**For the 10-bit interface:** Tx1[8] is bit 8 of the Tx_code_group[9:0] output to the transceiver. It is synchronous to TxClkOut1.
TxErr1/Tx1[9]	O	Transmit Error	**For GMII PHYs operating in 1000:** TxErr1 is usd for implementing carrier extension in half-duplex. Is synchronous to TxClkOut1. **For GMII PHYs operating in 10/100:** This output is irrelevant.
		Transmit Data	**For the 10-bit interface:** Tx1[9] is bit 9 of the Tx_code_group[9:0] output to the transceiver. It is synchronous to TxClkOut1. This pin is only relevant for the 10-bit interface.
TxClkOut1	O	Transmit Clock Output	**For GMII PHYs operating in 1000:** Provides the timing reference for the transfer of the TxEn1, TxErr1, and TxD1[7:0] signals. It operates at 125 MHz. **For GMII PHYs operating in 10/100:** This pin is irrelevant and must Not be Connected (NC). **For the 10-bit interface**: This pin is the PMA Transmit Clock. It provides the timing reference for the transfer of the Tx1[9:0] signals. It operates at 125 MHz.
CRS1	I	Carrier Sense	**For GMII PHYs:** Indicates that either the transmit or receive medium is non-idle. CRS1 is not synchronous to any clock. **For the 10-bit interface:** This pin is not used and must be pulled down.
RxD1[3:0] / Rx1[3:0]	I	Receive Data	**For GMII PHYs operating in 10/100:** RxD1[3:0] contains the receive data inputs and is synchronous to RxClk input.
		Receive Data	**For the 10-bit interface**: Rx1[3:0] are bits [3:0] of the Rx_code_group[9:0] input from the transceiver. They are synchronous to RxClk1[1:0].

Table 12: Ethernet Port_1 Interface Pin Assignments (Continued)

NOTE: This interface only applies to the MV64360 and MV64361 devices. Still, the MV64362 device uses the TxD1[4:1] signals in its reset sampling configuration. For further information see, Table 117 on page 348.

Pin Name	Type	Full Name	Description
RxD1[7:4]/ Rx1[7:4]	I	Receive Data	**For GMII PHYs operating in 1000:** RxD1[7:4] contains the receive data inputs and are synchronous to RxClk input. **For GMII PHYs operating in 10/100:** RxD1[7:4] must be held low.
		Receive Data	**For the 10-bit interface**: Rx1[7:4] are bits [7:4] of the Rx_code_group[9:0] input from the transceiver. They are synchronous to RxClk1[1:0].
RxErr1/Rx1[9]	I	Receive Error	**For GMII PHYs operating in 1000:** Also used for implementing carrier extension in half-duplex. Synchronous to RxClk1[1:0]. **For GMII PHYs operating in 10/100:** Synchronous to RxClk input.
		Receive Data	**For the 10-bit interface:** Rx1[9] is bit 9 of the Rx_code_group[9:0] input from the transceiver. It is synchronous to RxClk1[1:0]. This pin is only relevant for the 10-bit interface.
RxDV1/Rx1[8]	I	Receive Data Valid	**For GMII PHYs:** Indicates that valid data is present on the RxD1 lines. It is synchronous to RxClk1.
		Receive Data	**For the 10-bit interface:** Rx1[8] is bit [8] of the Rx_code_group [9:0] input from the transceiver. It is synchronous to RxClk1 [1:0].
RxClk1/ RxClk1[0]	I	Receive Clock	**For GMII mode operating in 1000 (RxClk1):** Provides the timing reference for the reception of the RxDv1, RxErr1, and RxD1[7:0] signals. It operates at 125 MHz. **For GMII mode operating in 10/100 (RxClk1):** Provides the timing reference for the reception of the RxDv1, RxErr1, and RxD1[3:0] signals. It operates at 2.5 MHz when in 10 Mbps, and at 25 MHz in 100 Mbps speed. **For the 10-bit interface (RxClk1[0]):** This pin is the receive code group complimentary 62.5 MHz clock input. Both phases are used to strobe Rx1[9:0] inputs.
COL1	I	Collision Detect	**For GMII mode operating in 100 (RxClk0):** Indicates a collision has been detected on the wire. This input is ignored in full-duplex mode. COL0 is not synchronous to any clock. **For GMII mode operating in 100 (RxClk0):** This pin is not used and must be pulled down.
Device Interface Pin Count: 25			

Table 13: Ethernet Control Interface Pin Assignments

Pin Name	Type	Full Name	Description
CLK_125	I	Reference Clock	Provides the timing reference for all modes and must operate at 125 MHz.
MDC	O	Management Data Clock	MDC is the Clk input divided by 64. Provides the timing reference for the transfer of the MDIO signal. **NOTE:** Must be pulled down.
MDIO	I/O	Management Data In/Out	**For GMII mode operating in 100/1000:** Used to transfer control information and status between PHY devices and MV64360/1/2. **For GMII mode operating in 10:** This pin is not used and must be pulled down.
Serial Interface Pin Count: 3			

Table 14: MPP Interface Pin Assignment
NOTE: The MV64360/1/2 has two MPSCs multiplexed on the MPP pins

Pin Name	Type	Full Name	Description
MPP[31:0]	I/O	Multi Purpose Pins	Various functions
MPP Interface Pin Count: 32			

Table 15: TWSI Pin Assignment

Pin Name	Type	Full Name	Description
SCK	o/d I/O	TWSI Clock	TWSI serial clock. Serves as output when the MV64360/1/2 acts as a TWSI master. Serves as input when the device acts as a TWSI slave.
SDA	o/d I/O	TWSI Serial Data	Address or write data driven by the TWSI master or read response data driven by the TWSI slave.
TWSI Interface Pin Count: 2			

Table 16: JTAG Pin Assignment

Pin Name	Type	Full Name	Description
JTCLK	I	JTAG Clock	Clock input for the MV64360/1/2 JTAG controller. **NOTE:** A pull-down is required.
JTRST#	I	JTAG Reset	When asserted, resets the MV64360/1/2 JTAG controller. **NOTE:** A pull-down is required.
JTMS	I	JTAG Mode Select	Controls the MV64360/1/2 JTAG controller state. Sampled with the rising edge of JTCK. **NOTE:** A pull-up is required.

Table 16: JTAG Pin Assignment (Continued)

Pin Name	Type	Full Name	Description
JTDO	O	JTAG Data In	JTAG serial data output. Driven by the MV64360/1/2 on falling edge of JTCK.
JTDI	I	JTAG Data Out	JTAG serial data input. Sampled with JTCK rising edge. **NOTE:** A pull-down is required.
JTAG Interface Pin Count: 5			

Use Table 17 to determine the strapping configuration for systems when one of the following interfaces is not used.

Table 17: Unused Interface Strapping

Unused Interface	Strapping
CPU	Pull up: A[0-35], AP[0-4], BR0#, BR1#, TS#, TBST#, TSIZ[0-2], TT[0-4], ARTRY#, HIT0#, HIT1#, DRDY0#, DRDY1# Pull down: DevAD[5], DevAD[8], DevAD[9]
Ethernet Port	Pull down: TxClk0/1, CRS0/1, COL0/1, RxClk0/1, RxD0/1[9:0], MDIO, MDC
TWSI	Pull up SCK and SDA.
MPP	All signals must be configured as outputs. It is recommended to pull these signals either high or low so the hardware will be protected from software errors.
SDRAM	Pull up: DevAD[18] Route StartBurst to StartBurstIn and FBClkOut to FBClkIn
Device	Pull down the Ready# pin.
PCI	To bypass the need of putting pull ups on the data signals, pull down GNT0/1# and configure the chip to work with external PCI arbiter, see EN bit [31] in Table 383 on page 540. Pull up: FRAME0/1#, IRDY0/1#, DEVSEL0/1#, STOP0/1#, TRDY0/1#, REQ640/1#, ACK640/1#, PERR0/1#, 64En#, HS **NOTE:** The REQ640/1#, ACK640/1#, and 64En# signals only apply to the MV64360 and MV64362 devices. Also, the "1" signals only apply to the MV64360 and MV64361.

Section 4. MV64360 Pinout Map and Table, 724 Pin BGA

Figure 5: MV64360 Pinout Map (Top View, Left Side)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
A	NB*	NB	Col1	JTRST#	AVDD1	DRAM_ ACAL	DQ[1]	DQ[6]	DQ[11]	DM[1]	DQS[0]	DQ[17]	DQ[22]	DQ[28]	CB[1]	CB[7]	DA[8]	A
B	NB	Ready#	Col0	JTMS	AVSS1	DRAM_ DCAL	DQ[2]	DQ[7]	DQ[12]	DM[2]	DQS[1]	DQ[18]	DQ[23]	DQ[29]	CB[2]	DA[13]	DA[7]	B
C	DevWE[1]#	DevWE[0]#	JTCLK	JTDI	RES	DQ[0]	DQ[3]	DQ[8]	DQ[13]	DM[3]	DQS[2]	DQ[19]	DQ[24]	DQ[30]	CB[3]	DA[12]	DA[6]	C
D	BAdr[0]	DevWE[3]#	DevWE[2]#	JTDO	SysRst#	AVDD0	DQ[4]	DQ[9]	DQ[14]	DM[8]	DQS[3]	DQ[20]	DQ[25]	DQ[31]	CB[4]	DA[11]	DA[5]	D
E	CSTiming#	ALE	BAdr[2]	BAdr[1]	SysClk	AVSS0	DQ[5]	DQ[10]	DQ[15]	SSTL_ VREF1	DQS[8]	DQ[21]	DQ[26]	CB[0]	CB[5]	DA[9]	DA[4]	E
F	DevAD[4]	DevAD[3]	DevAD[2]	DevAD[1]/ DevRW#	DevAD[0]/ BootCS#	AVSS0_A	VDD DRAM	VDD DRAM	DM[0]	VDD DRAM	DQ[16]	VDD Core	DQ[27]	VDD DRAM	CB[6]	VDD DRAM	DA[3]	F
G	DevAD[8]	DevDP[0]	DevAD[7]	DevAD[6]	DevAD[5]	VDD I/O												G
H	DevAD[13]	DevAD[12]	DevAD[11]	DevAD[10]	DevAD[9]	VDD I/O		*No Ball										H
J	DevAD[18]	DevAD[17]	DevAD[16]	DevDP[1]	DevAD[15]	DevAD[14]												J
K	DevAD[23]	DevAD[22]	DevAD[21]	DevAD[20]	DevAD[19]	VDD Core												K
L	DevAD[28]/ DevCS[0]#	DevAD[27]	DevAD[26]	DevAD[25]	DevAD[24]	DevDP[2]												L
M	TxD0[7]	DevDP[3]	DevAD[31]/ DevCS[3]#	DevAD[30]/ DevCS[2]#	DevAD[29]/ DevCS[1]#	VDD I/O												M
N	TxD0[2]	TxD0[3]	TxD0[4]	TxD0[5]	TxD0[6]	VDD I/O												N
P	TxClk0	TxErr0	TxEn0	TxClk0Out	TxD0[0]	TxD0[1]								VSS	VSS	VSS	VSS	P
R	RxD0[0]	RxDV0	RxClk0	RxErr0	CRS0	VDD I/O								VSS	VSS	VSS	VSS	R
T	RxD0[6]	RxD0[5]	RxD0[4]	RxD0[3]	RxD0[2]	RxD0[1]								VSS	VSS	VSS	VSS	T
U	TxD1[6]	TxD1[7]	CLK_125	MDIO	MDC	RxD0[7]								VSS	VSS	VSS	VSS	U
V	TxD1[0]	TxD1[1]	TxD1[2]	TxD1[3]	TxD1[4]	TxD1[5]								VSS	VSS	VSS	VSS	V
W	CRS1	TxClk1	TxErr1	TxEn1	TxClk1Out	VSS								VSS	VSS	VSS	VSS	W
Y	RxD1[2]	RxD1[1]	RxD1[0]	RxDV1	RxClk1	RxErr1								VSS	VSS	VSS	VSS	Y
AA	RxD1[7]	RxD1[6]	RxD1[5]	RxD1[4]	RxD1[3]	VDD I/O								VSS	VSS	VSS	VSS	AA
AB	MPP[3]	MPP[2]	MPP[1]	MPP[0]	SDA	SCK												AB
AC	MPP[8]	MPP[7]	MPP[6]	MPP[5]	MPP[4]	VDD I/O												AC
AD	MPP[14]	MPP[13]	MPP[12]	MPP[11]	MPP[10]	MPP[9]												AD
AE	GNT1#	RST1#	INT1#	PCLK1	MPP[15]	VDD Core												AE
AF	PAD1[27]	PAD1[28]	PAD1[29]	PAD1[30]	PAD1[31]	REQ1#												AF
AG	IDSEL1	CBE1[3]#	PAD1[24]	PAD1[25]	PAD1[26]	VDD I/O												AG
AH	PAD1[19]	PAD1[20]	PAD1[21]	PAD1[22]	PAD1[23]	VDD I/O												AH
AJ	FRAME1#	CBE1[2]#	PAD1[16]	PAD1[17]	PAD1[18]	PAD1[3]	**VDD I/O**	VDD I/O	PAD1[59]/ TxD2[3]	VDD I/O	PAD1[48]/ RxDV2	VDD Core	VDD I/O	VDD I/O	INT0#	PAD0[29]	VDD I/O	AJ
AK	STOP1#	DEVSEL1#	TRDY1#	IRDY1#	CBE1[0]#	PAD1[2]	REQ641#	PAR641	PAD1[58]/ TxD2[2]	PAD1[53]/ TxErr2	PAD1[47]/ RxD2[0]	PAD1[42]/ RxD2[5]	PAD1[37]	PAD1[32]	RST0#	PAD0[28]	CBE0[3]#	AK
AL	PAR1	SERR1#	PERR1#	PAD1[10]	PAD1[7]	PAD1[1]	CBE1[7]#	PAD1[63]/ TxD2[7]	PAD1[57]/ TxD2[1]	PAD1[52]/ TxClk2	PAD1[46]/ RxD2[1]	PAD1[41]/ RxD2[6]	PAD1[36]	PCI1_CAL	GNT0#	PAD0[27]	IDSEL0	AL
AM	PAD1[15]	CBE1[1]#	PAD1[13]	M66EN1	PAD1[6]	PAD1[0]	CBE1[6]#	PAD1[62]/ TxD2[6]	PAD1[56]/ TxD2[0]	PAD1[51]/ CRS2	PAD1[45]/ RxD2[2]	PAD1[40]/ RxD2[7]	PAD1[35]	PAVDD	REQ0#	PAD0[26]	PAD0[23]	AM
AN	NB	PAD1[14]	PAD1[12]	PAD1[9]	PAD1[5]	VREF1	CBE1[5]#	PAD1[61]/ TxD2[5]	PAD1[55]/ TxClk2Out	PAD1[50]/ RxErr2	PAD1[44]/ RxD2[3]	PAD1[39]/ Col2	PAD1[34]	PAVSS	PAD0[31]	PAD0[25]	PAD0[22]	AN
AP	NB	NB	PAD1[11]	PAD1[8]	PAD1[4]	ACK641#	CBE1[4]#	PAD1[60]/ TxD2[4]	PAD1[54]/ TxEn2	PAD1[49]/ RxClk2	PAD1[43]/ RxD2[4]	PAD1[38]	PAD1[33]	PCLK0	PAD0[30]	PAD0[24]	PAD0[21]	AP
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	

Figure 6: MV64360 Pinout Map (Top View, Right Side)

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	
A	DA[2]	StartBurstIn	BA[0]	CS[1]#	DQ[34]	DQ[40]	DQ[45]	DM[6]	DQS[7]	DQ[53]	DQ[58]	DQ[63]	DH[3]	DH[7]	DH[9]	NB	NB	A
B	ClkOut	StartBurst	BA[1]	CS[2]#	DQ[35]	DQ[41]	DQ[46]	DM[7]	DQ[48]	DQ[54]	DQ[59]	DP[0]	DH[4]	DP[1]	DH[10]	DH[11]	NB	B
C	ClkOut#	DA[0]	CAS#	CS[3]#	DQ[36]	DQ[42]	DQ[47]	DQS[4]	DQ[49]	DQ[55]	DQ[60]	DH[0]	DH[5]	DH[8]	DH[14]	DH[13]	DH[12]	C
D	FBClkOut	DA[1]	RAS#	DQ[32]	DQ[37]	DQ[43]	SSTL_ VREF0	DQS[5]	DQ[50]	DQ[56]	DQ[61]	DH[1]	DH[6]	DH[17]	DH[16]	DP[2]	DH[15]	D
E	FBClkIn	DA[10]	WE#	DQ[33]	DQ[38]	DQ[44]	DM[4]	DQS[6]	DQ[51]	DQ[57]	DQ[62]	DH[2]	DH[22]	DH[21]	DH[20]	DH[19]	DH[18]	E
F	VSS	VDD DRAM	CS[0]#	VDD DRAM	DQ[39]	VDD DRAM	DM[5]	VDD Core	DQ[52]	VDD DRAM	VDD DRAM	VDD CPU	DH[26]	DH[25]	DH[24]	DP[3]	DH[23]	F
G												VDD CPU	DH[31]	DH[30]	DH[29]	DH[28]	DH[27]	G
H												DBG0#	BG1#	DBG1#	CPUInt[1]#	CPUInt[0]#	NC	H
J												ABB#	DBB#	DTI[2]	DTI[1]	DTI[0]	BG0#	J
K												VDD CPU	DRDY0#	DRDY1#	BR0#	BR1#	GbI#	K
L												VDD Core	AACK#	TA#	ARTRY#	HIT0#	HIT1#	L
M												VDD CPU	TT[4]	TT[3]	TT[2]	TT[1]	TT[0]	M
N												AP[0]	TS#	TBST#	TSIZ[2]	TSIZ[1]	TSIZ[0]	N
P	VSS	VSS	VSS	VSS								VDD CPU	AP[1]	A[3]	A[2]	A[1]	A[0]	P
R	VSS	VSS	VSS	VSS								A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	R
T	VSS	VSS	VSS	VSS								VDD CPU	A[13]	A[12]	AP[2]	A[11]	A[10]	T
U	VSS	VSS	VSS	VSS								A[19]	A[18]	A[17]	A[16]	A[15]	A[14]	U
V	VSS	VSS	VSS	VSS								VSS	A[23]	A[22]	A[21]	A[20]	AP[3]	V
W	VSS	VSS	VSS	VSS								A[28]	AP[4]	A[27]	A[26]	A[25]	A[24]	W
Y	VSS	VSS	VSS	VSS								A[34]	A[33]	A[32]	A[31]	A[30]	A[29]	Y
AA	VSS	VSS	VSS	VSS								VDD CPU	DL[1]	DL[0]	DP[4]	NC	A[35]	AA
AB												DL[7]	DL[6]	DL[5]	DL[4]	DL[3]	DL[2]	AB
AC												VDD CPU	DL[11]	DL[10]	DL[9]	DL[8]	DP[5]	AC
AD												VDD Core	DP[6]	DL[15]	DL[14]	DL[13]	DL[12]	AD
AE												VDD CPU	DL[20]	DL[19]	DL[18]	DL[17]	DL[16]	AE
AF												DL[25]	DL[24]	DP[7]	DL[23]	DL[22]	DL[21]	AF
AG												VDD CPU	DL[30]	DL[29]	DL[28]	DL[27]	DL[26]	AG
AH												VDD I/O	MPP[29]	MPP[30]	MPP[31]	CPU_CAL	DL[31]	AH
AJ	VSS	CBE0[2]#	VDD I/O	**VDD I/O**	M66EN0	VDD I/O	PAD0[0]	VDD Core	PAD0[61]	VDD I/O	VDD I/O	VDD I/O	MPP[24]	MPP[25]	MPP[26]	MPP[27]	MPP[28]	AJ
AK	PAD0[20]	FRAME0#	PERR0#	PAD0[14]	PAD0[9]	PAD0[5]	VREF0	CBE0[5]#	PAD0[60]	PAD0[55]	PAD0[50]	PAD0[45]	PAD0[40]	MPP[20]	MPP[21]	MPP[22]	MPP[23]	AK
AL	PAD0[19]	IRDY0#	SERR0#	PAD0[13]	PAD0[8]	PAD0[4]	ACK640#	CBE0[4]#	PAD0[59]	PAD0[54]	PAD0[49]	PAD0[44]	PAD0[39]	PAD0[35]	ENUM#	MPP[18]	MPP[19]	AL
AM	PAD0[18]	TRDY0#	PAR0	PAD0[12]	CBE0[0]#	PAD0[3]	REQ640#	PAR640	PAD0[58]	PAD0[53]	PAD0[48]	PAD0[43]	PAD0[38]	PAD0[34]	LED	MPP[16]	MPP[17]	AM
AN	PAD0[17]	DEVSEL0#	CBE0[1]#	PAD0[11]	PAD0[7]	PAD0[2]	CBE0[7]#	PAD0[63]	PAD0[57]	PAD0[52]	PAD0[47]	PAD0[42]	PAD0[37]	PAD0[33]	HS	PCI0_CAL	NB	AN
AP	PAD0[16]	STOP0#	PAD0[15]	PAD0[10]	PAD0[6]	PAD0[1]	CBE0[6]#	PAD0[62]	PAD0[56]	PAD0[51]	PAD0[46]	PAD0[41]	PAD0[36]	PAD0[32]	64En	NB	NB	AP
	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	

Table 18: MV64360 Pinout Sorted by Signal Name

Signal Name	Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name	Ball#
64En	AP32	AP[3]	V34	Col0	B03	DevAD[16]	J03	DH[19]	E33
A[0]	P34	AP[4]	W30	Col1	A03	DevAD[17]	J02	DH[20]	E32
A[1]	P33	ARTRY#	L32	CPU_CAL	AH33	DevAD[18]	J01	DH[21]	E31
A[2]	P32	AVDD0	D06	CPUInt[0]#	H33	DevAD[19]	K05	DH[22]	E30
A[3]	P31	AVDD1	A05	CPUInt[1]#	H32	DevAD[20]	K04	DH[23]	F34
A[4]	R34	AVSS0	E06	CRS0	R05	DevAD[21]	K03	DH[24]	F32
A[5]	R33	AVSS0_A	F06	CRS1	W01	DevAD[22]	K02	DH[25]	F31
A[6]	R32	AVSS1	B05	CS[0]#	F20	DevAD[23]	K01	DH[26]	F30
A[7]	R31	BA[0]	A20	CS[1]#	A21	DevAD[24]	L05	DH[27]	G34
A[8]	R30	BA[1]	B20	CS[2]#	B21	DevAD[25]	L04	DH[28]	G33
A[9]	R29	BAdr[0]	D01	CS[3]#	C21	DevAD[26]	L03	DH[29]	G32
A[10]	T34	BAdr[1]	E04	CSTiming#	E01	DevAD[27]	L02	DH[30]	G31
A[11]	T33	BAdr[2]	E03	DA[0]	C19	DevAD[28]/ DevCS[0]#	L01	DH[31]	G30
A[12]	T31	BG0#	J34	DA[1]	D19	DevAD[29]/ DevCS[1]#	M05	DL[0]	AA31
A[13]	T30	BG1#	H30	DA[2]	A18	DevAD[30]/ DevCS[2]#	M04	DL[1]	AA30
A[14]	U34	BR0#	K32	DA[3]	F17	DevAD[31]/ DevCS[3]#	M03	DL[2]	AB34
A[15]	U33	BR1#	K33	DA[4]	E17	DevDP[0]	G02	DL[3]	AB33
A[16]	U32	CAS#	C20	DA[5]	D17	DevDP[1]	J04	DL[4]	AB32
A[17]	U31	CB[0]	E14	DA[6]	C17	DevDP[2]	L06	DL[5]	AB31
A[18]	U30	CB[1]	A15	DA[7]	B17	DevDP[3]	M02	DL[6]	AB30
A[19]	U29	CB[2]	B15	DA[8]	A17	DEVSEL0#	AN19	DL[7]	AB29
A[20]	V33	CB[3]	C15	DA[9]	E16	DEVSEL1#	AK02	DL[8]	AC33
A[21]	V32	CB[4]	D15	DA[10]	E19	DevWE[0]#	C02	DL[9]	AC32
A[22]	V31	CB[5]	E15	DA[11]	D16	DevWE[1]#	C01	DL[10]	AC31
A[23]	V30	CB[6]	F15	DA[12]	C16	DevWE[2]#	D03	DL[11]	AC30
A[24]	W34	CB[7]	A16	DA[13]	B16	DevWE[3]#	D02	DL[12]	AD34
A[25]	W33	CBE0[0]#	AM22	DBB#	J30	DH[0]	C29	DL[13]	AD33
A[26]	W32	CBE0[1]#	AN20	DBG0#	H29	DH[1]	D29	DL[14]	AD32
A[27]	W31	CBE0[2]#	AJ19	DBG1#	H31	DH[2]	E29	DL[15]	AD31
A[28]	W29	CBE0[3]#	AK17	DevAD[0]/ BootCS#	F05	DH[3]	A30	DL[16]	AE34
A[29]	Y34	CBE0[4]#	AL25	DevAD[1]/ DevRW#	F04	DH[4]	B30	DL[17]	AE33
A[30]	Y33	CBE0[5]#	AK25	DevAD[2]	F03	DH[5]	C30	DL[18]	AE32
A[31]	Y32	CBE0[6]#	AP24	DevAD[3]	F02	DH[6]	D30	DL[19]	AE31
A[32]	Y31	CBE0[7]#	AN24	DevAD[4]	F01	DH[7]	A31	DL[20]	AE30
A[33]	Y30	CBE1[0]#	AK05	DevAD[5]	G05	DH[8]	C31	DL[21]	AF34
A[34]	Y29	CBE1[1]#	AM02	DevAD[6]	G04	DH[9]	A32	DL[22]	AF33
A[35]	AA34	CBE1[2]#	AJ02	DevAD[7]	G03	DH[10]	B32	DL[23]	AF32
AACK#	L30	CBE1[3]#	AG02	DevAD[8]	G01	DH[11]	B33	DL[24]	AF30
ABB#	J29	CBE1[4]#	AP07	DevAD[9]	H05	DH[12]	C34	DL[25]	AF29
ACK640#	AL24	CBE1[5]#	AN07	DevAD[10]	H04	DH[13]	C33	DL[26]	AG34
ACK641#	AP06	CBE1[6]#	AM07	DevAD[11]	H03	DH[14]	C32	DL[27]	AG33
ALE	E02	CBE1[7]#	AL07	DevAD[12]	H02	DH[15]	D34	DL[28]	AG32
AP[0]	N29	CLK_125	U03	DevAD[13]	H01	DH[16]	D32	DL[29]	AG31
AP[1]	P30	ClkOut	B18	DevAD[14]	J06	DH[17]	D31	DL[30]	AG30
AP[2]	T32	ClkOut#	C18	DevAD[15]	J05	DH[18]	E34	DL[31]	AH34

Table 18: MV64360 Pinout Sorted by Signal Name (Continued)

Signal Name	Ball#
DM[0]	F09
DM[1]	A10
DM[2]	B10
DM[3]	C10
DM[4]	E24
DM[5]	F24
DM[6]	A25
DM[7]	B25
DM[8]	D10
DP[0]	B29
DP[1]	B31
DP[2]	D33
DP[3]	F33
DP[4]	AA32
DP[5]	AC34
DP[6]	AD30
DP[7]	AF31
DQ[0]	C06
DQ[1]	A07
DQ[2]	B07
DQ[3]	C07
DQ[4]	D07
DQ[5]	E07
DQ[6]	A08
DQ[7]	B08
DQ[8]	C08
DQ[9]	D08
DQ[10]	E08
DQ[11]	A09
DQ[12]	B09
DQ[13]	C09
DQ[14]	D09
DQ[15]	E09
DQ[16]	F11
DQ[17]	A12
DQ[18]	B12
DQ[19]	C12
DQ[20]	D12
DQ[21]	E12
DQ[22]	A13
DQ[23]	B13
DQ[24]	C13
DQ[25]	D13
DQ[26]	E13
DQ[27]	F13
DQ[28]	A14
DQ[29]	B14
DQ[30]	C14
DQ[31]	D14
DQ[32]	D21
DQ[33]	E21
DQ[34]	A22
DQ[35]	B22
DQ[36]	C22
DQ[37]	D22
DQ[38]	E22
DQ[39]	F22
DQ[40]	A23
DQ[41]	B23
DQ[42]	C23
DQ[43]	D23
DQ[44]	E23
DQ[45]	A24
DQ[46]	B24
DQ[47]	C24
DQ[48]	B26
DQ[49]	C26
DQ[50]	D26
DQ[51]	E26
DQ[52]	F26
DQ[53]	A27
DQ[54]	B27
DQ[55]	C27
DQ[56]	D27
DQ[57]	E27
DQ[58]	A28
DQ[59]	B28
DQ[60]	C28
DQ[61]	D28
DQ[62]	E28
DQ[63]	A29
DQS[0]	A11
DQS[1]	B11
DQS[2]	C11
DQS[3]	D11
DQS[4]	C25
DQS[5]	D25
DQS[6]	E25
DQS[7]	A26
DQS[8]	E11
DRAM_ ACAL	A06
DRAM_ DCAL	B06
DRDY0#	K30
DRDY1#	K31
DTI[0]	J33
DTI[1]	J32
DTI[2]	J31
ENUM#	AL32
FBClkIn	E18
FBClkOut	D18
FRAME0#	AK19
FRAME1#	AJ01
GbI#	K34
GNT0#	AL15
GNT1#	AE01
HIT0#	L33
HIT1#	L34
HS	AN32
IDSEL0	AL17
IDSEL1	AG01
INT0#	AJ15
INT1#	AE03
IRDY0#	AL19
IRDY1#	AK04
JTCLK	C03
JTDI	C04
JTDO	D04
JTMS	B04
JTRST#	A04
LED	AM32
M66EN0	AJ22
M66EN1	AM04
MDC	U05
MDIO	U04
MPP[0]	AB04
MPP[1]	AB03
MPP[2]	AB02
MPP[3]	AB01
MPP[4]	AC05
MPP[5]	AC04
MPP[6]	AC03
MPP[7]	AC02
MPP[8]	AC01
MPP[9]	AD06
MPP[10]	AD05
MPP[11]	AD04
MPP[12]	AD03
MPP[13]	AD02
MPP[14]	AD01
MPP[15]	AE05
MPP[16]	AM33
MPP[17]	AM34
MPP[18]	AL33
MPP[19]	AL34
MPP[20]	AK31
MPP[21]	AK32
MPP[22]	AK33
MPP[23]	AK34
MPP[24]	AJ30
MPP[25]	AJ31
MPP[26]	AJ32
MPP[27]	AJ33
MPP[28]	AJ34
MPP[29]	AH30
MPP[30]	AH31
MPP[31]	AH32
NC	H34
NC	AA33
PAD0[0]	AJ24
PAD0[1]	AP23
PAD0[2]	AN23
PAD0[3]	AM23
PAD0[4]	AL23
PAD0[5]	AK23
PAD0[6]	AP22
PAD0[7]	AN22
PAD0[8]	AL22
PAD0[9]	AK22
PAD0[10]	AP21
PAD0[11]	AN21
PAD0[12]	AM21
PAD0[13]	AL21
PAD0[14]	AK21
PAD0[15]	AP20
PAD0[16]	AP18
PAD0[17]	AN18
PAD0[18]	AM18
PAD0[19]	AL18
PAD0[20]	AK18
PAD0[21]	AP17
PAD0[22]	AN17
PAD0[23]	AM17
PAD0[24]	AP16
PAD0[25]	AN16
PAD0[26]	AM16
PAD0[27]	AL16
PAD0[28]	AK16
PAD0[29]	AJ16
PAD0[30]	AP15
PAD0[31]	AN15
PAD0[32]	AP31
PAD0[33]	AN31
PAD0[34]	AM31
PAD0[35]	AL31
PAD0[36]	AP30
PAD0[37]	AN30
PAD0[38]	AM30
PAD0[39]	AL30
PAD0[40]	AK30
PAD0[41]	AP29
PAD0[42]	AN29
PAD0[43]	AM29
PAD0[44]	AL29
PAD0[45]	AK29
PAD0[46]	AP28
PAD0[47]	AN28
PAD0[48]	AM28
PAD0[49]	AL28
PAD0[50]	AK28
PAD0[51]	AP27
PAD0[52]	AN27
PAD0[53]	AM27
PAD0[54]	AL27
PAD0[55]	AK27
PAD0[56]	AP26
PAD0[57]	AN26
PAD0[58]	AM26
PAD0[59]	AL26
PAD0[60]	AK26
PAD0[61]	AJ26
PAD0[62]	AP25
PAD0[63]	AN25
PAD1[0]	AM06
PAD1[1]	AL06
PAD1[2]	AK06

Table 18: MV64360 Pinout Sorted by Signal Name (Continued)

Signal Name	Ball#
PAD1[3]	AJ06
PAD1[4]	AP05
PAD1[5]	AN05
PAD1[6]	AM05
PAD1[7]	AL05
PAD1[8]	AP04
PAD1[9]	AN04
PAD1[10]	AL04
PAD1[11]	AP03
PAD1[12]	AN03
PAD1[13]	AM03
PAD1[14]	AN02
PAD1[15]	AM01
PAD1[16]	AJ03
PAD1[17]	AJ04
PAD1[18]	AJ05
PAD1[19]	AH01
PAD1[20]	AH02
PAD1[21]	AH03
PAD1[22]	AH04
PAD1[23]	AH05
PAD1[24]	AG03
PAD1[25]	AG04
PAD1[26]	AG05
PAD1[27]	AF01
PAD1[28]	AF02
PAD1[29]	AF03
PAD1[30]	AF04
PAD1[31]	AF05
PAD1[32]	AK14
PAD1[33]	AP13
PAD1[34]	AN13
PAD1[35]	AM13
PAD1[36]	AL13
PAD1[37]	AK13
PAD1[38]	AP12
PAD1[39]/Col2	AN12
PAD1[40]/ RxD2[7]	AM12
PAD1[41]/ RxD2[6]	AL12
PAD1[42]/ RxD2[5]	AK12
PAD1[43]/ RxD2[4]	AP11
PAD1[44]/ RxD2[3]	AN11
PAD1[45]/ RxD2[2]	AM11
PAD1[46]/ RxD2[1]	AL11
PAD1[47]/ RxD2[0]	AK11
PAD1[48]/ RxDV2	AJ11
PAD1[49]/ RxClk2	AP10
PAD1[50]/ RxErr2	AN10
PAD1[51]/ CRS2	AM10
PAD1[52]/ TxClk2	AL10
PAD1[53]/ TxErr2	AK10
PAD1[54]/ TxEn2	AP09
PAD1[55]/ TxClk2O	AN09
PAD1[56]/ TxD2[0]	AM09
PAD1[57]/ TxD2[1]	AL09
PAD1[58]/ TxD2[2]	AK09
PAD1[59]/ TxD2[3]	AJ09
PAD1[60]/ TxD2[4]	AP08
PAD1[61]/ TxD2[5]	AN08
PAD1[62]/ TxD2[6]	AM08
PAD1[63]/ TxD2[7]	AL08
PAR0	AM20
PAR1	AL01
PAR640	AM25
PAR641	AK08
PAVDD	AM14
PAVSS	AN14
PCI0_CAL	AN33
PCI1_CAL	AL14
PCLK0	AP14
PCLK1	AE04
PERR0#	AK20
PERR1#	AL03
RAS#	D20
Ready#	B02
REQ0#	AM15
REQ1#	AF06
REQ640#	AM24
REQ641#	AK07
RES	C05
RST0#	AK15
RST1#	AE02
RxClk0	R03
RxClk1	Y05
RxD0[0]	R01
RxD0[1]	T06
RxD0[2]	T05
RxD0[3]	T04
RxD0[4]	T03
RxD0[5]	T02
RxD0[6]	T01
RxD0[7]	U06
RxD1[0]	Y03
RxD1[1]	Y02
RxD1[2]	Y01
RxD1[3]	AA05
RxD1[4]	AA04
RxD1[5]	AA03
RxD1[6]	AA02
RxD1[7]	AA01
RxDV0	R02
RxDV1	Y04
RxErr0	R04
RxErr1	Y06
SCK	AB06
SDA	AB05
SERR0#	AL20
SERR1#	AL02
SSTL_ VREF0	D24
SSTL_ VREF1	E10
StartBurst	B19
StartBurstIn	A19
STOP0#	AP19
STOP1#	AK01
SysClk	E05
SysRst#	D05
TA#	L31
TBST#	N31
TRDY0#	AM19
TRDY1#	AK03
TS#	N30
TSIZ[0]	N34
TSIZ[1]	N33
TSIZ[2]	N32
TT[0]	M34
TT[1]	M33
TT[2]	M32
TT[3]	M31
TT[4]	M30
TxClk0	P01
TxClk0Out	P04
TxClk1	W02
TxClk1Out	W05
TxD0[0]	P05
TxD0[1]	P06
TxD0[2]	N01
TxD0[3]	N02
TxD0[4]	N03
TxD0[5]	N04
TxD0[6]	N05
TxD0[7]	M01
TxD1[0]	V01
TxD1[1]	V02
TxD1[2]	V03
TxD1[3]	V04
TxD1[4]	V05
TxD1[5]	V06
TxD1[6]	U01
TxD1[7]	U02
TxEn0	P03
TxEn1	W04
TxErr0	P02
TxErr1	W03
VDD Core	F12
VDD Core	F25
VDD Core	K06
VDD Core	L29
VDD Core	AD29
VDD Core	AE06
VDD Core	AJ12
VDD Core	AJ25
VDD CPU	F29
VDD CPU	G29
VDD CPU	K29
VDD CPU	M29
VDD CPU	P29
VDD CPU	T29
VDD CPU	AA29
VDD CPU	AC29
VDD CPU	AE29
VDD CPU	AG29
VDD DRAM	F07
VDD DRAM	F08
VDD DRAM	F10
VDD DRAM	F14
VDD DRAM	F16
VDD DRAM	F19
VDD DRAM	F21
VDD DRAM	F23
VDD DRAM	F27
VDD DRAM	F28
VDD I/O	G06
VDD I/O	H06
VDD I/O	M06
VDD I/O	N06
VDD I/O	R06
VDD I/O	AA06
VDD I/O	AC06
VDD I/O	AG06
VDD I/O	AH06
VDD I/O	AH29
VDD I/O	AJ07
VDD I/O	AJ08
VDD I/O	AJ10
VDD I/O	AJ13
VDD I/O	AJ14
VDD I/O	AJ17
VDD I/O	AJ20
VDD I/O	AJ21
VDD I/O	AJ23
VDD I/O	AJ27
VDD I/O	AJ28
VDD I/O	AJ29
VREF0	AK24
VREF1	AN06
VSS	F18
VSS	P14
VSS	P15
VSS	P16
VSS	P17
VSS	P18
VSS	P19
VSS	P20
VSS	P21
VSS	R14
VSS	R15
VSS	R16
VSS	R17
VSS	R18
VSS	R19
VSS	R20
VSS	R21
VSS	T14
VSS	T15
VSS	T16

Table 18: MV64360 Pinout Sorted by Signal Name (Continued)

Signal Name	Ball#
VSS	T17
VSS	T18
VSS	T19
VSS	T20
VSS	T21
VSS	U14
VSS	U15
VSS	U16
VSS	U17
VSS	U18
VSS	U19
VSS	U20
VSS	U21
VSS	V14
VSS	V15
VSS	V16
VSS	V17
VSS	V18
VSS	V19
VSS	V20
VSS	V21
VSS	V29
VSS	W06
VSS	W14
VSS	W15
VSS	W16
VSS	W17
VSS	W18
VSS	W19
VSS	W20
VSS	W21
VSS	Y14
VSS	Y15
VSS	Y16
VSS	Y17
VSS	Y18
VSS	Y19
VSS	Y20
VSS	Y21
VSS	AA14
VSS	AA15
VSS	AA16
VSS	AA17
VSS	AA18
VSS	AA19

Signal Name	Ball#
VSS	AA20
VSS	AA21
VSS	AJ18
WE#	E20

Section 5. MV64361 Pinout Map and Table, 724 Pin BGA

Figure 7: MV64361 Pinout Map (Top View, Left Side)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
A	NB*	NB	Col1	JTRST#	AVDD1	DRAM_ ACAL	DQ[1]	DQ[6]	DQ[11]	DM[1]	DQS[0]	DQ[17]	DQ[22]	DQ[28]	CB[1]	CB[7]	DA[8]	A
B	NB	Ready#	Col0	JTMS	AVSS1	DRAM_ DCAL	DQ[2]	DQ[7]	DQ[12]	DM[2]	DQS[1]	DQ[18]	DQ[23]	DQ[29]	CB[2]	DA[13]	DA[7]	B
C	DevWE[1]#	DevWE[0]#	JTCLK	JTDI	RES	DQ[0]	DQ[3]	DQ[8]	DQ[13]	DM[3]	DQS[2]	DQ[19]	DQ[24]	DQ[30]	CB[3]	DA[12]	DA[6]	C
D	BAdr[0]	DevWE[3]#	DevWE[2]#	JTDO	SysRst#	AVDD0	DQ[4]	DQ[9]	DQ[14]	DM[8]	DQS[3]	DQ[20]	DQ[25]	DQ[31]	CB[4]	DA[11]	DA[5]	D
E	CSTiming#	ALE	BAdr[2]	BAdr[1]	SysClk	AVSS0	DQ[5]	DQ[10]	DQ[15]	SSTL_ VREF1	DQS[8]	DQ[21]	DQ[26]	CB[0]	CB[5]	DA[9]	DA[4]	E
F	DevAD[4]	DevAD[3]	DevAD[2]	DevAD[1]/ DevRW#	DevAD[0]/ BootCS#	AVSS0_A	VDD DRAM	VDD DRAM	DM[0]	VDD DRAM	DQ[16]	VDD Core	DQ[27]	VDD DRAM	CB[6]	VDD DRAM	DA[3]	F
G	DevAD[8]	DevDP[0]	DevAD[7]	DevAD[6]	DevAD[5]	VDD I/O												G
H	DevAD[13]	DevAD[12]	DevAD[11]	DevAD[10]	DevAD[9]	VDD I/O		*No Ball										H
J	DevAD[18]	DevAD[17]	DevAD[16]	DevDP[1]	DevAD[15]	DevAD[14]												J
K	DevAD[23]	DevAD[22]	DevAD[21]	DevAD[20]	DevAD[19]	VDD Core												K
L	DevAD[28]/ DevCS[0]#	DevAD[27]	DevAD[26]	DevAD[25]	DevAD[24]	DevDP[2]												L
M	TxD0[7]	DevDP[3]	DevAD[31]/ DevCS[3]#	DevAD[30]/ DevCS[2]#	DevAD[29]/ DevCS[1]#	VDD I/O												M
N	TxD0[2]	TxD0[3]	TxD0[4]	TxD0[5]	TxD0[6]	VDD I/O												N
P	TxClk0	TxErr0	TxEn0	TxClk0Out	TxD0[0]	TxD0[1]								VSS	VSS	VSS	VSS	P
R	RxD0[0]	RxDV0	RxClk0	RxErr0	CRS0	VDD I/O								VSS	VSS	VSS	VSS	R
T	RxD0[6]	RxD0[5]	RxD0[4]	RxD0[3]	RxD0[2]	RxD0[1]								VSS	VSS	VSS	VSS	T
U	TxD1[6]	TxD1[7]	CLK_125	MDIO	MDC	RxD0[7]								VSS	VSS	VSS	VSS	U
V	TxD1[0]	TxD1[1]	TxD1[2]	TxD1[3]	TxD1[4]	TxD1[5]								VSS	VSS	VSS	VSS	V
W	CRS1	TxClk1	TxErr1	TxEn1	TxClk1Out	VSS								VSS	VSS	VSS	VSS	W
Y	RxD1[2]	RxD1[1]	RxD1[0]	RxDV1	RxClk1	RxErr1								VSS	VSS	VSS	VSS	Y
AA	RxD1[7]	RxD1[6]	RxD1[5]	RxD1[4]	RxD1[3]	VDD I/O								VSS	VSS	VSS	VSS	AA
AB	MPP[3]	MPP[2]	MPP[1]	MPP[0]	SDA	SCK												AB
AC	MPP[8]	MPP[7]	MPP[6]	MPP[5]	MPP[4]	VDD I/O												AC
AD	MPP[14]	MPP[13]	MPP[12]	MPP[11]	MPP[10]	MPP[9]												AD
AE	GNT1#	RST1#	INT1#	PCLK1	MPP[15]	VDD Core												AE
AF	PAD1[27]	PAD1[28]	PAD1[29]	PAD1[30]	PAD1[31]	REQ1#												AF
AG	IDSEL1	CBE1[3]#	PAD1[24]	PAD1[25]	PAD1[26]	VDD I/O												AG
AH	PAD1[19]	PAD1[20]	PAD1[21]	PAD1[22]	PAD1[23]	VDD I/O												AH
AJ	FRAME1#	CBE1[2]#	PAD1[16]	PAD1[17]	PAD1[18]	PAD1[3]	**VDD I/O**	VDD I/O	NC	VDD I/O	NC	VDD Core	VDD I/O	VDD I/O	INT0#	PAD0[29]	VDD I/O	AJ
AK	STOP1#	DEVSEL1#	TRDY1#	IRDY1#	CBE1[0]#	PAD1[2]	Pull-up	NC	NC	NC	NC	NC	NC	NC	RST0#	PAD0[28]	CBE0[3]#	AK
AL	PAR1	SERR1#	PERR1#	PAD1[10]	PAD1[7]	PAD1[1]	NC	NC	NC	NC	NC	NC	NC	PCI1_CAL	GNT0#	PAD0[27]	IDSEL0	AL
AM	PAD1[15]	CBE1[1]#	PAD1[13]	M66EN1	PAD1[6]	PAD1[0]	NC	NC	NC	NC	NC	NC	NC	PAVDD	REQ0#	PAD0[26]	PAD0[23]	AM
AN	NB	PAD1[14]	PAD1[12]	PAD1[9]	PAD1[5]	VREF1	NC	NC	NC	NC	NC	NC	NC	PAVSS	PAD0[31]	PAD0[25]	PAD0[22]	AN
AP	NB	NB	PAD1[11]	PAD1[8]	PAD1[4]	VDD I/O	NC	NC	NC	NC	NC	NC	NC	PCLK0	PAD0[30]	PAD0[24]	PAD0[21]	AP
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	

CONFIDENTIAL

Document Classification: Proprietary Information

Not Approved by Document Control - For Review Only

Figure 8: MV64361 Pinout Map (Top View, Right Side)

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	
A	DA[2]	StartBurstI n	BA[0]	CS[1]#	DQ[34]	DQ[40]	DQ[45]	DM[6]	DQS[7]	DQ[53]	DQ[58]	DQ[63]	DH[3]	DH[7]	DH[9]	NB	NB	A
B	ClkOut	StartBurst	BA[1]	CS[2]#	DQ[35]	DQ[41]	DQ[46]	DM[7]	DQ[48]	DQ[54]	DQ[59]	DP[0]	DH[4]	DP[1]	DH[10]	DH[11]	NB	B
C	ClkOut#	DA[0]	CAS#	CS[3]#	DQ[36]	DQ[42]	DQ[47]	DQS[4]	DQ[49]	DQ[55]	DQ[60]	DH[0]	DH[5]	DH[8]	DH[14]	DH[13]	DH[12]	C
D	FBClkOut	DA[1]	RAS#	DQ[32]	DQ[37]	DQ[43]	SSTL_ VREF0	DQS[5]	DQ[50]	DQ[56]	DQ[61]	DH[1]	DH[6]	DH[17]	DH[16]	DP[2]	DH[15]	D
E	FBClkIn	DA[10]	WE#	DQ[33]	DQ[38]	DQ[44]	DM[4]	DQS[6]	DQ[51]	DQ[57]	DQ[62]	DH[2]	DH[22]	DH[21]	DH[20]	DH[19]	DH[18]	E
F	VSS	VDD DRAM	CS[0]#	VDD DRAM	DQ[39]	VDD DRAM	DM[5]	VDD Core	DQ[52]	VDD DRAM	VDD DRAM	VDD CPU	DH[26]	DH[25]	DH[24]	DP[3]	DH[23]	F
G												VDD CPU	DH[31]	DH[30]	DH[29]	DH[28]	DH[27]	G
H												DBG0#	BG1#	DBG1#	CPUInt[1]#	CPUInt[0]#	NC	H
J												ABB#	DBB#	DTI[2]	DTI[1]	DTI[0]	BG0#	J
K												VDD CPU	DRDY0#	DRDY1#	BR0#	BR1#	GbI#	K
L												VDD Core	AACK#	TA#	ARTRY#	HIT0#	HIT1#	L
M												VDD CPU	TT[4]	TT[3]	TT[2]	TT[1]	TT[0]	M
N												AP[0]	TS#	TBST#	TSIZ[2]	TSIZ[1]	TSIZ[0]	N
P	VSS	VSS	VSS	VSS								VDD CPU	AP[1]	A[3]	A[2]	A[1]	A[0]	P
R	VSS	VSS	VSS	VSS								A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	R
T	VSS	VSS	VSS	VSS								VDD CPU	A[13]	A[12]	AP[2]	A[11]	A[10]	T
U	VSS	VSS	VSS	VSS								A[19]	A[18]	A[17]	A[16]	A[15]	A[14]	U
V	VSS	VSS	VSS	VSS								VSS	A[23]	A[22]	A[21]	A[20]	AP[3]	V
W	VSS	VSS	VSS	VSS								A[28]	AP[4]	A[27]	A[26]	A[25]	A[24]	W
Y	VSS	VSS	VSS	VSS								A[34]	A[33]	A[32]	A[31]	A[30]	A[29]	Y
AA	VSS	VSS	VSS	VSS								VDD CPU	DL[1]	DL[0]	DP[4]	NC	A[35]	AA
AB												DL[7]	DL[6]	DL[5]	DL[4]	DL[3]	DL[2]	AB
AC												VDD CPU	DL[11]	DL[10]	DL[9]	DL[8]	DP[5]	AC
AD												VDD Core	DP[6]	DL[15]	DL[14]	DL[13]	DL[12]	AD
AE												VDD CPU	DL[20]	DL[19]	DL[18]	DL[17]	DL[16]	AE
AF												DL[25]	DL[24]	DP[7]	DL[23]	DL[22]	DL[21]	AF
AG												VDD CPU	DL[30]	DL[29]	DL[28]	DL[27]	DL[26]	AG
AH												VDD I/O	MPP[29]	MPP[30]	MPP[31]	CPU_CAL	DL[31]	AH
AJ	VSS	CBE0[2]#	VDD I/O	**VDD I/O**	M66EN0	VDD I/O	PAD0[0]	VDD Core	NC	VDD I/O	VDD I/O	VDD I/O	MPP[24]	MPP[25]	MPP[26]	MPP[27]	MPP[28]	AJ
AK	PAD0[20]	FRAME0#	PERR0#	PAD0[14]	PAD0[9]	PAD0[5]	VREF0	NC	NC	NC	NC	NC	NC	MPP[20]	MPP[21]	MPP[22]	MPP[23]	AK
AL	PAD0[19]	IRDY0#	SERR0#	PAD0[13]	PAD0[8]	PAD0[4]	VDD I/O	NC	NC	NC	NC	NC	NC	NC	ENUM#	MPP[18]	MPP[19]	AL
AM	PAD0[18]	TRDY0#	PAR0	PAD0[12]	CBE0[0]#	PAD0[3]	VDD I/O	NC	NC	NC	NC	NC	NC	NC	LED	MPP[16]	MPP[17]	AM
AN	PAD0[17]	DEVSEL0#	CBE0[1]#	PAD0[11]	PAD0[7]	PAD0[2]	NC	NC	NC	NC	NC	NC	NC	NC	HS	PCI0_CAL	NB	AN
AP	PAD0[16]	STOP0#	PAD0[15]	PAD0[10]	PAD0[6]	PAD0[1]	NC	NC	NC	NC	NC	NC	NC	NC	Pull-up	NB	NB	AP
	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	

Table 19: MV64361 Pinout Sorted by Pin Name

Signal Name	Ball#
A[0]	P34
A[1]	P33
A[2]	P32
A[3]	P31
A[4]	R34
A[5]	R33
A[6]	R32
A[7]	R31
A[8]	R30
A[9]	R29
A[10]	T34
A[11]	T33
A[12]	T31
A[13]	T30
A[14]	U34
A[15]	U33
A[16]	U32
A[17]	U31
A[18]	U30
A[19]	U29
A[20]	V33
A[21]	V32
A[22]	V31
A[23]	V30
A[24]	W34
A[25]	W33
A[26]	W32
A[27]	W31
A[28]	W29
A[29]	Y34
A[30]	Y33
A[31]	Y32
A[32]	Y31
A[33]	Y30
A[34]	Y29
A[35]	AA34
AACK#	L30
ABB#	J29
ALE	E02
AP[0]	N29
AP[1]	P30
AP[2]	T32
AP[3]	V34
AP[4]	W30
ARTRY#	L32
AVDD0	D06
AVDD1	A05
AVSS0	E06
AVSS0_A	F06
AVSS1	B05
BA[0]	A20
BA[1]	B20
BAdr[0]	D01
BAdr[1]	E04
BAdr[2]	E03
BG0#	J34
BG1#	H30
BR0#	K32
BR1#	K33
CAS#	C20
CB[0]	E14
CB[1]	A15
CB[2]	B15
CB[3]	C15
CB[4]	D15
CB[5]	E15
CB[6]	F15
CB[7]	A16
CBE0[0]#	AM22
CBE0[1]#	AN20
CBE0[2]#	AJ19
CBE0[3]#	AK17
CBE1[0]#	AK05
CBE1[1]#	AM02
CBE1[2]#	AJ02
CBE1[3]#	AG02
CLK_125	U03
ClkOut	B18
ClkOut#	C18
Col0	B03
Col1	A03
CPU_CAL	AH33
CPUInt[0]#	H33
CPUInt[1]#	H32
CRS0	R05
CRS1	W01
CS[0]#	F20
CS[1]#	A21
CS[2]#	B21
CS[3]#	C21
CSTiming#	E01
DA[0]	C19
DA[1]	D19
DA[2]	A18
DA[3]	F17
DA[4]	E17
DA[5]	D17
DA[6]	C17
DA[7]	B17
DA[8]	A17
DA[9]	E16
DA[10]	E19
DA[11]	D16
DA[12]	C16
DA[13]	B16
DBB#	J30
DBG0#	H29
DBG1#	H31
DevAD[0]/ BootCS	F05
DevAD[1]/ DevRW	F04
DevAD[2]	F03
DevAD[3]	F02
DevAD[4]	F01
DevAD[5]	G05
DevAD[6]	G04
DevAD[7]	G03
DevAD[8]	G01
DevAD[9]	H05
DevAD[10]	H04
DevAD[11]	H03
DevAD[12]	H02
DevAD[13]	H01
DevAD[14]	J06
DevAD[15]	J05
DevAD[16]	J03
DevAD[17]	J02
DevAD[18]	J01
DevAD[19]	K05
DevAD[20]	K04
DevAD[21]	K03
DevAD[22]	K02
DevAD[23]	K01
DevAD[24]	L05
DevAD[25]	L04
DevAD[26]	L03
DevAD[27]	L02
DevAD[28]/ DevCS	L01
DevAD[29]/ DevCS	M05
DevAD[30]/ DevCS	M04
DevAD[31]/ DevCS	M03
DevDP[0]	G02
DevDP[1]	J04
DevDP[2]	L06
DevDP[3]	M02
DEVSEL0#	AN19
DEVSEL1#	AK02
DevWE[0]#	C02
DevWE[1]#	C01
DevWE[2]#	D03
DevWE[3]#	D02
DH[0]	C29
DH[1]	D29
DH[2]	E29
DH[3]	A30
DH[4]	B30
DH[5]	C30
DH[6]	D30
DH[7]	A31
DH[8]	C31
DH[9]	A32
DH[10]	B32
DH[11]	B33
DH[12]	C34
DH[13]	C33
DH[14]	C32
DH[15]	D34
DH[16]	D32
DH[17]	D31
DH[18]	E34
DH[19]	E33
DH[20]	E32
DH[21]	E31
DH[22]	E30
DH[23]	F34
DH[24]	F32
DH[25]	F31
DH[26]	F30
DH[27]	G34
DH[28]	G33
DH[29]	G32
DH[30]	G31
DH[31]	G30
DL[0]	AA31
DL[1]	AA30
DL[2]	AB34
DL[3]	AB33
DL[4]	AB32
DL[5]	AB31
DL[6]	AB30
DL[7]	AB29
DL[8]	AC33
DL[9]	AC32
DL[10]	AC31
DL[11]	AC30
DL[12]	AD34
DL[13]	AD33
DL[14]	AD32
DL[15]	AD31
DL[16]	AE34
DL[17]	AE33
DL[18]	AE32
DL[19]	AE31
DL[20]	AE30
DL[21]	AF34
DL[22]	AF33
DL[23]	AF32
DL[24]	AF30
DL[25]	AF29
DL[26]	AG34
DL[27]	AG33
DL[28]	AG32
DL[29]	AG31
DL[30]	AG30
DL[31]	AH34
DM[0]	F09
DM[1]	A10
DM[2]	B10
DM[3]	C10
DM[4]	E24
DM[5]	F24
DM[6]	A25
DM[7]	B25
DM[8]	D10
DP[0]	B29
DP[1]	B31

Table 19: MV64361 Pinout Sorted by Pin Name (Continued)

Signal Name	Ball#
DP[2]	D33
DP[3]	F33
DP[4]	AA32
DP[5]	AC34
DP[6]	AD30
DP[7]	AF31
DQ[0]	C06
DQ[1]	A07
DQ[2]	B07
DQ[3]	C07
DQ[4]	D07
DQ[5]	E07
DQ[6]	A08
DQ[7]	B08
DQ[8]	C08
DQ[9]	D08
DQ[10]	E08
DQ[11]	A09
DQ[12]	B09
DQ[13]	C09
DQ[14]	D09
DQ[15]	E09
DQ[16]	F11
DQ[17]	A12
DQ[18]	B12
DQ[19]	C12
DQ[20]	D12
DQ[21]	E12
DQ[22]	A13
DQ[23]	B13
DQ[24]	C13
DQ[25]	D13
DQ[26]	E13
DQ[27]	F13
DQ[28]	A14
DQ[29]	B14
DQ[30]	C14
DQ[31]	D14
DQ[32]	D21
DQ[33]	E21
DQ[34]	A22
DQ[35]	B22
DQ[36]	C22
DQ[37]	D22
DQ[38]	E22
DQ[39]	F22
DQ[40]	A23
DQ[41]	B23
DQ[42]	C23
DQ[43]	D23
DQ[44]	E23
DQ[45]	A24
DQ[46]	B24
DQ[47]	C24
DQ[48]	B26
DQ[49]	C26
DQ[50]	D26
DQ[51]	E26
DQ[52]	F26
DQ[53]	A27
DQ[54]	B27
DQ[55]	C27
DQ[56]	D27
DQ[57]	E27
DQ[58]	A28
DQ[59]	B28
DQ[60]	C28
DQ[61]	D28
DQ[62]	E28
DQ[63]	A29
DQS[0]	A11
DQS[1]	B11
DQS[2]	C11
DQS[3]	D11
DQS[4]	C25
DQS[5]	D25
DQS[6]	E25
DQS[7]	A26
DQS[8]	E11
DRAM_ ACAL	A06
DRAM_ DCAL	B06
DRDY0#	K30
DRDY1#	K31
DTI[0]	J33
DTI[1]	J32
DTI[2]	J31
ENUM#	AL32
FBClkIn	E18
FBClkOut	D18
FRAME0#	AK19
FRAME1#	AJ01
GbI#	K34
GNT0#	AL15
GNT1#	AE01
HIT0#	L33
HIT1#	L34
HS	AN32
IDSEL0	AL17
IDSEL1	AG01
INT0#	AJ15
INT1#	AE03
IRDY0#	AL19
IRDY1#	AK04
JTCLK	C03
JTDI	C04
JTDO	D04
JTMS	B04
JTRST#	A04
LED	AM32
M66EN0	AJ22
M66EN1	AM04
MDC	U05
MDIO	U04
MPP[0]	AB04
MPP[1]	AB03
MPP[2]	AB02
MPP[3]	AB01
MPP[4]	AC05
MPP[5]	AC04
MPP[6]	AC03
MPP[7]	AC02
MPP[8]	AC01
MPP[9]	AD06
MPP[10]	AD05
MPP[11]	AD04
MPP[12]	AD03
MPP[13]	AD02
MPP[14]	AD01
MPP[15]	AE05
MPP[16]	AM33
MPP[17]	AM34
MPP[18]	AL33
MPP[19]	AL34
MPP[20]	AK31
MPP[21]	AK32
MPP[22]	AK33
MPP[23]	AK34
MPP[24]	AJ30
MPP[25]	AJ31
MPP[26]	AJ32
MPP[27]	AJ33
MPP[28]	AJ34
MPP[29]	AH30
MPP[30]	AH31
MPP[31]	AH32
NC	H34
NC	AA33
NC	AJ09
NC	AJ11
NC	AJ26
NC	AK08
NC	AK09
NC	AK10
NC	AK11
NC	AK12
NC	AK13
NC	AK14
NC	AK25
NC	AK26
NC	AK27
NC	AK28
NC	AK29
NC	AK30
NC	AL07
NC	AL08
NC	AL09
NC	AL10
NC	AL11
NC	AL12
NC	AL13
NC	AL25
NC	AL26
NC	AL27
NC	AL28
NC	AL29
NC	AL30
NC	AL31
NC	AM07
NC	AM08
NC	AM09
NC	AM10
NC	AM11
NC	AM12
NC	AM13
NC	AM25
NC	AM26
NC	AM27
NC	AM28
NC	AM29
NC	AM30
NC	AM31
NC	AN07
NC	AN08
NC	AN09
NC	AN10
NC	AN11
NC	AN12
NC	AN13
NC	AN24
NC	AN25
NC	AN26
NC	AN27
NC	AN28
NC	AN29
NC	AN30
NC	AN31
NC	AP07
NC	AP08
NC	AP09
NC	AP10
NC	AP11
NC	AP12
NC	AP13
NC	AP24
NC	AP25
NC	AP26
NC	AP27
NC	AP28
NC	AP29
NC	AP30
NC	AP31
PAD0[0]	AJ24
PAD0[1]	AP23
PAD0[2]	AN23
PAD0[3]	AM23

Table 19: MV64361 Pinout Sorted by Pin Name (Continued)

Signal Name	Ball#
PAD0[4]	AL23
PAD0[5]	AK23
PAD0[6]	AP22
PAD0[7]	AN22
PAD0[8]	AL22
PAD0[9]	AK22
PAD0[10]	AP21
PAD0[11]	AN21
PAD0[12]	AM21
PAD0[13]	AL21
PAD0[14]	AK21
PAD0[15]	AP20
PAD0[16]	AP18
PAD0[17]	AN18
PAD0[18]	AM18
PAD0[19]	AL18
PAD0[20]	AK18
PAD0[21]	AP17
PAD0[22]	AN17
PAD0[23]	AM17
PAD0[24]	AP16
PAD0[25]	AN16
PAD0[26]	AM16
PAD0[27]	AL16
PAD0[28]	AK16
PAD0[29]	AJ16
PAD0[30]	AP15
PAD0[31]	AN15
PAD1[0]	AM06
PAD1[1]	AL06
PAD1[2]	AK06
PAD1[3]	AJ06
PAD1[4]	AP05
PAD1[5]	AN05
PAD1[6]	AM05
PAD1[7]	AL05
PAD1[8]	AP04
PAD1[9]	AN04
PAD1[10]	AL04
PAD1[11]	AP03
PAD1[12]	AN03
PAD1[13]	AM03
PAD1[14]	AN02
PAD1[15]	AM01
PAD1[16]	AJ03
PAD1[17]	AJ04
PAD1[18]	AJ05
PAD1[19]	AH01
PAD1[20]	AH02
PAD1[21]	AH03
PAD1[22]	AH04
PAD1[23]	AH05
PAD1[24]	AG03
PAD1[25]	AG04
PAD1[26]	AG05
PAD1[27]	AF01
PAD1[28]	AF02
PAD1[29]	AF03
PAD1[30]	AF04
PAD1[31]	AF05
PAR0	AM20
PAR1	AL01
PAVDD	AM14
PAVSS	AN14
PCI0_CAL	AN33
PCI1_CAL	AL14
PCLK0	AP14
PCLK1	AE04
PERR0#	AK20
PERR1#	AL03
Pull-up	AP32
RAS#	D20
Ready#	B02
REQ0#	AM15
REQ1#	AF06
VDD I/O	AM24
Pull-up	AK07
RES	C05
RST0#	AK15
RST1#	AE02
RxClk0	R03
RxClk1	Y05
RxD0[0]	R01
RxD0[1]	T06
RxD0[2]	T05
RxD0[3]	T04
RxD0[4]	T03
RxD0[5]	T02
RxD0[6]	T01
RxD0[7]	U06
RxD1[0]	Y03
RxD1[1]	Y02
RxD1[2]	Y01
RxD1[3]	AA05
RxD1[4]	AA04
RxD1[5]	AA03
RxD1[6]	AA02
RxD1[7]	AA01
RxDV0	R02
RxDV1	Y04
RxErr0	R04
RxErr1	Y06
SCK	AB06
SDA	AB05
SERR0#	AL20
SERR1#	AL02
SSTL_ VREF0	D24
SSTL_ VREF1	E10
StartBurst	B19
StartBurstIn	A19
STOP0#	AP19
STOP1#	AK01
SysClk	E05
SysRst#	D05
TA#	L31
TBST#	N31
TRDY0#	AM19
TRDY1#	AK03
TS#	N30
TSIZ[0]	N34
TSIZ[1]	N33
TSIZ[2]	N32
TT[0]	M34
TT[1]	M33
TT[2]	M32
TT[3]	M31
TT[4]	M30
TxClk0	P01
TxClk0Out	P04
TxClk1	W02
TxClk1Out	W05
TxD0[0]	P05
TxD0[1]	P06
TxD0[2]	N01
TxD0[3]	N02
TxD0[4]	N03
TxD0[5]	N04
TxD0[6]	N05
TxD0[7]	M01
TxD1[0]	V01
TxD1[1]	V02
TxD1[2]	V03
TxD1[3]	V04
TxD1[4]	V05
TxD1[5]	V06
TxD1[6]	U01
TxD1[7]	U02
TxEn0	P03
TxEn1	W04
TxErr0	P02
TxErr1	W03
VDD Core	F12
VDD Core	F25
VDD Core	K06
VDD Core	L29
VDD Core	AD29
VDD Core	AE06
VDD Core	AJ12
VDD Core	AJ25
VDD CPU	F29
VDD CPU	G29
VDD CPU	K29
VDD CPU	M29
VDD CPU	P29
VDD CPU	T29
VDD CPU	AA29
VDD CPU	AC29
VDD CPU	AE29
VDD CPU	AG29
VDD DRAM	F07
VDD DRAM	F08
VDD DRAM	F10
VDD DRAM	F14
VDD DRAM	F16
VDD DRAM	F19
VDD DRAM	F21
VDD DRAM	F23
VDD DRAM	F27
VDD DRAM	F28
VDD I/O	G06
VDD I/O	H06
VDD I/O	M06
VDD I/O	N06
VDD I/O	R06
VDD I/O	AA06
VDD I/O	AC06
VDD I/O	AG06
VDD I/O	AH06
VDD I/O	AH29
VDD I/O	AJ07
VDD I/O	AJ08
VDD I/O	AJ10
VDD I/O	AJ13
VDD I/O	AJ14
VDD I/O	AJ17
VDD I/O	AJ20
VDD I/O	AJ21
VDD I/O	AJ23
VDD I/O	AJ27
VDD I/O	AJ28
VDD I/O	AJ29
VREF0	AK24
VDD I/O	AL24
VDD I/O	AP06
VREF1	AN06
VSS	F18
VSS	P14
VSS	P15
VSS	P16
VSS	P17
VSS	P18
VSS	P19
VSS	P20
VSS	P21
VSS	R14
VSS	R15
VSS	R16
VSS	R17
VSS	R18
VSS	R19
VSS	R20
VSS	R21
VSS	T14
VSS	T15
VSS	T16

Table 19: MV64361 Pinout Sorted by Pin Name (Continued)

Signal Name	Ball#
VSS	T17
VSS	T18
VSS	T19
VSS	T20
VSS	T21
VSS	U14
VSS	U15
VSS	U16
VSS	U17
VSS	U18
VSS	U19
VSS	U20
VSS	U21
VSS	V14
VSS	V15
VSS	V16
VSS	V17
VSS	V18
VSS	V19
VSS	V20
VSS	V21
VSS	V29
VSS	W06
VSS	W14
VSS	W15
VSS	W16
VSS	W17
VSS	W18
VSS	W19
VSS	W20
VSS	W21
VSS	Y14
VSS	Y15
VSS	Y16
VSS	Y17
VSS	Y18
VSS	Y19
VSS	Y20
VSS	Y21
VSS	AA14
VSS	AA15
VSS	AA16
VSS	AA17
VSS	AA18
VSS	AA19
VSS	AA20
VSS	AA21
VSS	AJ18
WE#	E20

Section 6. MV64362 Pinout Map and Table, 724 Pin BGA

Figure 9: MV64362 Pinout Map (Top View, Left Side)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
A	NB*	NB	Col1	JTRST#	AVDD1	DRAM_ ACAL	DQ[1]	DQ[6]	DQ[11]	DM[1]	DQS[0]	DQ[17]	DQ[22]	DQ[28]	CB[1]	CB[7]	DA[8]	A
B	NB	Ready#	Col0	JTMS	AVSS1	DRAM_ DCAL	DQ[2]	DQ[7]	DQ[12]	DM[2]	DQS[1]	DQ[18]	DQ[23]	DQ[29]	CB[2]	DA[13]	DA[7]	B
C	DevWE[1]#	DevWE[0]#	JTCLK	JTDI	RES	DQ[0]	DQ[3]	DQ[8]	DQ[13]	DM[3]	DQS[2]	DQ[19]	DQ[24]	DQ[30]	CB[3]	DA[12]	DA[6]	C
D	BAdr[0]	DevWE[3]#	DevWE[2]#	JTDO	SysRst#	AVDD0	DQ[4]	DQ[9]	DQ[14]	DM[8]	DQS[3]	DQ[20]	DQ[25]	DQ[31]	CB[4]	DA[11]	DA[5]	D
E	CSTiming#	ALE	BAdr[2]	BAdr[1]	SysClk	AVSS0	DQ[5]	DQ[10]	DQ[15]	SSTL_ VREF1	DQS[8]	DQ[21]	DQ[26]	CB[0]	CB[5]	DA[9]	DA[4]	E
F	DevAD[4]	DevAD[3]	DevAD[2]	DevAD[1]/ DevRW#	DevAD[0]/ BootCS#	AVSS0_A	VDD DRAM	VDD DRAM	DM[0]	VDD DRAM	DQ[16]	VDD Core	DQ[27]	VDD DRAM	CB[6]	VDD DRAM	DA[3]	F
G	DevAD[8]	DevDP[0]	DevAD[7]	DevAD[6]	DevAD[5]	VDD I/O												G
H	DevAD[13]	DevAD[12]	DevAD[11]	DevAD[10]	DevAD[9]	VDD I/O		* No Ball										H
J	DevAD[18]	DevAD[17]	DevAD[16]	DevDP[1]	DevAD[15]	DevAD[14]												J
K	DevAD[23]	DevAD[22]	DevAD[21]	DevAD[20]	DevAD[19]	VDD Core												K
L	DevAD[28]/ DevCS[0]#	DevAD[27]	DevAD[26]	DevAD[25]	DevAD[24]	DevDP[2]												L
M	TxD0[7]	DevDP[3]	DevAD[31]/ DevCS[3]#	DevAD[30]/ DevCS[2]#	DevAD[29]/ DevCS[1]#	VDD I/O												M
N	TxD[2]	TxD[3]	TxD[4]	TxD[5]	TxD0[6]	VDD I/O												N
P	TxClk	TxErr	TxEn	TxClkOut	TxD[0]	TxD[1]								VSS	VSS	VSS	VSS	P
R	RxD[0]	RxDV	RxClk	RxErr	CRS	VDD I/O								VSS	VSS	VSS	VSS	R
T	RxD[6]	RxD[5]	RxD[4]	RxD[3]	RxD[2]	RxD[1]								VSS	VSS	VSS	VSS	T
U	NC	NC	CLK_125	MDIO	MDC	RxD0[7]								VSS	VSS	VSS	VSS	U
V	NC	RstCfg[0]	RstCfg[1]	RstCfg[2]	RstCfg[3]	NC								VSS	VSS	VSS	VSS	V
W	VSS	VSS	NC	NC	NC	VSS								VSS	VSS	VSS	VSS	W
Y	VSS	VSS	VSS	VSS	VSS	VSS								VSS	VSS	VSS	VSS	Y
AA	VSS	VSS	VSS	VSS	VSS	VDD I/O								VSS	VSS	VSS	VSS	AA
AB	MPP[3]	MPP[2]	MPP[1]	MPP[0]	SDA	SCK												AB
AC	MPP[8]	MPP[7]	MPP[6]	MPP[5]	MPP[4]	VDD I/O												AC
AD	MPP[14]	MPP[13]	MPP[12]	MPP[11]	MPP[10]	MPP[9]												AD
AE	VSS	RST#	NC	PCLK	MPP[15]	VDD Core												AE
AF	NC	NC	NC	NC	NC	NC												AF
AG	VSS	NC	NC	NC	NC	VDD I/O												AG
AH	NC	NC	NC	NC	NC	VDD I/O												AH
AJ	Pull-up	NC	NC	NC	NC	NC	**VDD I/O**	VDD I/O	NC	VDD I/O	NC	VDD Core	VDD I/O	VDD I/O	INT#	PAD[29]	VDD I/O	AJ
AK	VDD I/O	VDD I/O	VDD I/O	Pull-up	NC	NC	Pull-up	NC	NC	NC	NC	NC	NC	NC	RST#	PAD[28]	CBE[3]#	AK
AL	NC	Pull-up	VDD I/O	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	Pull-up	GNT#	PAD[27]	IDSEL	AL
AM	NC	NC	NC	M66EN	NC	NC	NC	NC	NC	NC	NC	NC	NC	PAVDD	REQ0#	PAD[26]	PAD[23]	AM
AN	NB	NC	NC	NC	NC	VREF	NC	NC	NC	NC	NC	NC	NC	PAVSS	PAD[31]	PAD[25]	PAD[22]	AN
AP	NB	NB	NC	NC	NC	VDD I/O	NC	NC	NC	NC	NC	NC	NC	PCLK	PAD[30]	PAD[24]	PAD[21]	AP
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	

Figure 10: MV64362 Pinout Map (Top View, Right Side)

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	
A	DA[2]	StartBurstIn	BA[0]	CS[1]#	DQ[34]	DQ[40]	DQ[45]	DM[6]	DQS[7]	DQ[53]	DQ[58]	DQ[63]	DH[3]	DH[7]	DH[9]	NB	NB	A
B	ClkOut	StartBurst	BA[1]	CS[2]#	DQ[35]	DQ[41]	DQ[46]	DM[7]	DQ[48]	DQ[54]	DQ[59]	DP[0]	DH[4]	DP[1]	DH[10]	DH[11]	NB	B
C	ClkOut#	DA[0]	CAS#	CS[3]#	DQ[36]	DQ[42]	DQ[47]	DQS[4]	DQ[49]	DQ[55]	DQ[60]	DH[0]	DH[5]	DH[8]	DH[14]	DH[13]	DH[12]	C
D	FBClkOut	DA[1]	RAS#	DQ[32]	DQ[37]	DQ[43]	SSTL_ VREF0	DQS[5]	DQ[50]	DQ[56]	DQ[61]	DH[1]	DH[6]	DH[17]	DH[16]	DP[2]	DH[15]	D
E	FBClkIn	DA[10]	WE#	DQ[33]	DQ[38]	DQ[44]	DM[4]	DQS[6]	DQ[51]	DQ[57]	DQ[62]	DH[2]	DH[22]	DH[21]	DH[20]	DH[19]	DH[18]	E
F	VSS	VDD DRAM	CS[0]#	VDD DRAM	DQ[39]	VDD DRAM	DM[5]	VDD Core	DQ[52]	VDD DRAM	VDD DRAM	VDD CPU	DH[26]	DH[25]	DH[24]	DP[3]	DH[23]	F
G												VDD CPU	DH[31]	DH[30]	DH[29]	DH[28]	DH[27]	G
H												DBG0#	NC	NC	NC	CPUInt[0]#	NC	H
J												ABB#	DBB#	DTI[2]	DTI[1]	DTI[0]	BG0#	J
K												VDD CPU	DRDY0#	VDD CPU	BR0#	VDD CPU	Gbl#	K
L												VDD Core	AACK#	TA#	ARTRY#	HIT0#	VDD CPU	L
M												VDD CPU	TT[4]	TT[3]	TT[2]	TT[1]	TT[0]	M
N												AP[0]	TS#	TBST#	TSIZ[2]	TSIZ[1]	TSIZ[0]	N
P	VSS	VSS	VSS	VSS								VDD CPU	AP[1]	A[3]	A[2]	A[1]	A[0]	P
R	VSS	VSS	VSS	VSS								A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	R
T	VSS	VSS	VSS	VSS								VDD CPU	A[13]	A[12]	AP[2]	A[11]	A[10]	T
U	VSS	VSS	VSS	VSS								A[19]	A[18]	A[17]	A[16]	A[15]	A[14]	U
V	VSS	VSS	VSS	VSS								VSS	A[23]	A[22]	A[21]	A[20]	AP[3]	V
W	VSS	VSS	VSS	VSS								A[28]	AP[4]	A[27]	A[26]	A[25]	A[24]	W
Y	VSS	VSS	VSS	VSS								A[34]	A[33]	A[32]	A[31]	A[30]	A[29]	Y
AA	VSS	VSS	VSS	VSS								VDD CPU	DL[1]	DL[0]	DP[4]	NC	A[35]	AA
AB												DL[7]	DL[6]	DL[5]	DL[4]	DL[3]	DL[2]	AB
AC												VDD CPU	DL[11]	DL[10]	DL[9]	DL[8]	DP[5]	AC
AD												VDD Core	DP[6]	DL[15]	DL[14]	DL[13]	DL[12]	AD
AE												VDD CPU	DL[20]	DL[19]	DL[18]	DL[17]	DL[16]	AE
AF												DL[25]	DL[24]	DP[7]	DL[23]	DL[22]	DL[21]	AF
AG												VDD CPU	DL[30]	DL[29]	DL[28]	DL[27]	DL[26]	AG
AH												VDD I/O	MPP[29]	MPP[30]	MPP[31]	CPU_CAL	DL[31]	AH
AJ	VSS	CBE[2]#	VDD I/O	**VDD I/O**	M66EN	VDD I/O	PAD[0]	VDD Core	PAD[61]	VDD I/O	VDD I/O	VDD I/O	MPP[24]	MPP[25]	MPP[26]	MPP[27]	MPP[28]	AJ
AK	PAD[20]	FRAME#	PERR#	PAD[14]	PAD[9]	PAD[5]	VREF	CBE[5]#	PAD[60]	PAD[55]	PAD[50]	PAD[45]	PAD[40]	MPP[20]	MPP[21]	MPP[22]	MPP[23]	AK
AL	PAD[19]	IRDY#	SERR#	PAD[13]	PAD[8]	PAD[4]	ACK64#	CBE[4]#	PAD[59]	PAD[54]	PAD[49]	PAD[44]	PAD[39]	PAD[35]	ENUM#	MPP[18]	MPP[19]	AL
AM	PAD[18]	TRDY#	PAR	PAD[12]	CBE[0]#	PAD[3]	REQ64#	PAR64	PAD[58]	PAD[53]	PAD[48]	PAD[43]	PAD[38]	PAD[34]	LED	MPP[16]	MPP[17]	AM
AN	PAD[17]	DEVSEL#	CBE[1]#	PAD[11]	PAD[7]	PAD[2]	CBE[7]#	PAD[63]	PAD[57]	PAD[52]	PAD[47]	PAD[42]	PAD[37]	PAD[33]	HS	PCI_CAL	NB	AN
AP	PAD[16]	STOP#	PAD[15]	PAD[10]	PAD[6]	PAD[1]	CBE[6]#	PAD[62]	PAD[56]	PAD[51]	PAD[46]	PAD[41]	PAD[36]	PAD[32]	64En	NB	NB	AP
	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	

Table 20: MV64362 Pinout Sorted by Pin Name

Signal Name	Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name	Ball#
64En	AP32	AP[4]	W30	DA[1]	D19	DevAD[30]/ DevCS	M04	DL[2]	AB34
A[0]	P34	ARTRY#	L32	DA[2]	A18	DevAD[31]/ DevCS	M03	DL[3]	AB33
A[1]	P33	AVDD0	D06	DA[3]	F17	DevDP[0]	G02	DL[4]	AB32
A[2]	P32	AVDD1	A05	DA[4]	E17	DevDP[1]	J04	DL[5]	AB31
A[3]	P31	AVSS0	E06	DA[5]	D17	DevDP[2]	L06	DL[6]	AB30
A[4]	R34	AVSS0_A	F06	DA[6]	C17	DevDP[3]	M02	DL[7]	AB29
A[5]	R33	AVSS1	B05	DA[7]	B17	DEVSEL#	AN19	DL[8]	AC33
A[6]	R32	BA[0]	A20	DA[8]	A17	DevWE[0]#	C02	DL[9]	AC32
A[7]	R31	BA[1]	B20	DA[9]	E16	DevWE[1]#	C01	DL[10]	AC31
A[8]	R30	BAdr[0]	D01	DA[10]	E19	DevWE[2]#	D03	DL[11]	AC30
A[9]	R29	BAdr[1]	E04	DA[11]	D16	DevWE[3]#	D02	DL[12]	AD34
A[10]	T34	BAdr[2]	E03	DA[12]	C16	DH[0]	C29	DL[13]	AD33
A[11]	T33	BG0#	J34	DA[13]	B16	DH[1]	D29	DL[14]	AD32
A[12]	T31	BR0#	K32	DBB#	J30	DH[2]	E29	DL[15]	AD31
A[13]	T30	CAS#	C20	DBG0#	H29	DH[3]	A30	DL[16]	AE34
A[14]	U34	CB[0]	E14	DevAD[0]/ BootCS	F05	DH[4]	B30	DL[17]	AE33
A[15]	U33	CB[1]	A15	DevAD[1]/ DevRW	F04	DH[5]	C30	DL[18]	AE32
A[16]	U32	CB[2]	B15	DevAD[2]	F03	DH[6]	D30	DL[19]	AE31
A[17]	U31	CB[3]	C15	DevAD[3]	F02	DH[7]	A31	DL[20]	AE30
A[18]	U30	CB[4]	D15	DevAD[4]	F01	DH[8]	C31	DL[21]	AF34
A[19]	U29	CB[5]	E15	DevAD[5]	G05	DH[9]	A32	DL[22]	AF33
A[20]	V33	CB[6]	F15	DevAD[6]	G04	DH[10]	B32	DL[23]	AF32
A[21]	V32	CB[7]	A16	DevAD[7]	G03	DH[11]	B33	DL[24]	AF30
A[22]	V31	CBE[0]#	AM22	DevAD[8]	G01	DH[12]	C34	DL[25]	AF29
A[23]	V30	CBE[1]#	AN20	DevAD[9]	H05	DH[13]	C33	DL[26]	AG34
A[24]	W34	CBE[2]#	AJ19	DevAD[10]	H04	DH[14]	C32	DL[27]	AG33
A[25]	W33	CBE[3]#	AK17	DevAD[11]	H03	DH[15]	D34	DL[28]	AG32
A[26]	W32	CBE[4]#	AL25	DevAD[12]	H02	DH[16]	D32	DL[29]	AG31
A[27]	W31	CBE[5]#	AK25	DevAD[13]	H01	DH[17]	D31	DL[30]	AG30
A[28]	W29	CBE[6]#	AP24	DevAD[14]	J06	DH[18]	E34	DL[31]	AH34
A[29]	Y34	CBE[7]#	AN24	DevAD[15]	J05	DH[19]	E33	DM[0]	F09
A[30]	Y33	CLK_125	U03	DevAD[16]	J03	DH[20]	E32	DM[1]	A10
A[31]	Y32	ClkOut	B18	DevAD[17]	J02	DH[21]	E31	DM[2]	B10
A[32]	Y31	ClkOut#	C18	DevAD[18]	J01	DH[22]	E30	DM[3]	C10
A[33]	Y30	Col0	B03	DevAD[19]	K05	DH[23]	F34	DM[4]	E24
A[34]	Y29	Col1	A03	DevAD[20]	K04	DH[24]	F32	DM[5]	F24
A[35]	AA34	CPU_CAL	AH33	DevAD[21]	K03	DH[25]	F31	DM[6]	A25
AACK#	L30	CPUInt[0]#	H33	DevAD[22]	K02	DH[26]	F30	DM[7]	B25
ACK64#	AL24	CRS	R05	DevAD[23]	K01	DH[27]	G34	DM[8]	D10
ABB#	J29	CS[0]#	F20	DevAD[24]	L05	DH[28]	G33	DP[0]	B29
ALE	E02	CS[1]#	A21	DevAD[25]	L04	DH[29]	G32	DP[1]	B31
AP[0]	N29	CS[2]#	B21	DevAD[26]	L03	DH[30]	G31	DP[2]	D33
AP[1]	P30	CS[3]#	C21	DevAD[27]	L02	DH[31]	G30	DP[3]	F33
AP[2]	T32	CSTiming#	E01	DevAD[28]/ DevCS	L01	DL[0]	AA31	DP[4]	AA32
AP[3]	V34	DA[0]	C19	DevAD[29]/ DevCS	M05	DL[1]	AA30	DP[5]	AC34

Table 20: MV64362 Pinout Sorted by Pin Name (Continued)

Signal Name	Ball#
DP[6]	AD30
DP[7]	AF31
DQ[0]	C06
DQ[1]	A07
DQ[2]	B07
DQ[3]	C07
DQ[4]	D07
DQ[5]	E07
DQ[6]	A08
DQ[7]	B08
DQ[8]	C08
DQ[9]	D08
DQ[10]	E08
DQ[11]	A09
DQ[12]	B09
DQ[13]	C09
DQ[14]	D09
DQ[15]	E09
DQ[16]	F11
DQ[17]	A12
DQ[18]	B12
DQ[19]	C12
DQ[20]	D12
DQ[21]	E12
DQ[22]	A13
DQ[23]	B13
DQ[24]	C13
DQ[25]	D13
DQ[26]	E13
DQ[27]	F13
DQ[28]	A14
DQ[29]	B14
DQ[30]	C14
DQ[31]	D14
DQ[32]	D21
DQ[33]	E21
DQ[34]	A22
DQ[35]	B22
DQ[36]	C22
DQ[37]	D22
DQ[38]	E22
DQ[39]	F22
DQ[40]	A23
DQ[41]	B23
DQ[42]	C23
DQ[43]	D23
DQ[44]	E23
DQ[45]	A24
DQ[46]	B24
DQ[47]	C24
DQ[48]	B26
DQ[49]	C26
DQ[50]	D26
DQ[51]	E26
DQ[52]	F26
DQ[53]	A27
DQ[54]	B27
DQ[55]	C27
DQ[56]	D27
DQ[57]	E27
DQ[58]	A28
DQ[59]	B28
DQ[60]	C28
DQ[61]	D28
DQ[62]	E28
DQ[63]	A29
DQS[0]	A11
DQS[1]	B11
DQS[2]	C11
DQS[3]	D11
DQS[4]	C25
DQS[5]	D25
DQS[6]	E25
DQS[7]	A26
DQS[8]	E11
DRAM_ ACAL	A06
DRAM_ DCAL	B06
DRDY0#	K30
DTI[0]	J33
DTI[1]	J32
DTI[2]	J31
ENUM#	AL32
FBClkIn	E18
FBClkOut	D18
FRAME#	AK19
GbI#	K34
GNT#	AL15
HIT0#	L33
HS	AN32
IDSEL	AL17
INT#	AJ15
IRDY#	AL19
JTCLK	C03
JTDI	C04
JTDO	D04
JTMS	B04
JTRST#	A04
LED	AM32
M66EN	AJ22
M66EN	AM04
MDC	U05
MDIO	U04
MPP[0]	AB04
MPP[1]	AB03
MPP[2]	AB02
MPP[3]	AB01
MPP[4]	AC05
MPP[5]	AC04
MPP[6]	AC03
MPP[7]	AC02
MPP[8]	AC01
MPP[9]	AD06
MPP[10]	AD05
MPP[11]	AD04
MPP[12]	AD03
MPP[13]	AD02
MPP[14]	AD01
MPP[15]	AE05
MPP[16]	AM33
MPP[17]	AM34
MPP[18]	AL33
MPP[19]	AL34
MPP[20]	AK31
MPP[21]	AK32
MPP[22]	AK33
MPP[23]	AK34
MPP[24]	AJ30
MPP[25]	AJ31
MPP[26]	AJ32
MPP[27]	AJ33
MPP[28]	AJ34
MPP[29]	AH30
MPP[30]	AH31
MPP[31]	AH32
NC	H30
NC	H31
NC	H32
NC	H34
NC	U01
NC	U02
NC	V01
NC	V06
NC	W03
NC	W04
NC	W05
NC	AA33
NC	AE03
NC	AF01
NC	AF02
NC	AF03
NC	AF04
NC	AF05
NC	AF06
NC	AG02
NC	AG03
NC	AG04
NC	AG05
NC	AH01
NC	AH02
NC	AH03
NC	AH04
NC	AH05
NC	AJ02
NC	AJ03
NC	AJ04
NC	AJ05
NC	AJ06
NC	AJ09
NC	AJ11
NC	AK05
NC	AK06
NC	AK08
NC	AK09
NC	AK10
NC	AK11
NC	AK12
NC	AK13
NC	AK14
NC	AL01
NC	AL04
NC	AL05
NC	AL06
NC	AL07
NC	AL08
NC	AL09
NC	AL10
NC	AL11
NC	AL12
NC	AL13
NC	AM01
NC	AM02
NC	AM03
NC	AM05
NC	AM06
NC	AM07
NC	AM08
NC	AM09
NC	AM10
NC	AM11
NC	AM12
NC	AM13
NC	AN02
NC	AN03
NC	AN04
NC	AN05
NC	AN07
NC	AN08
NC	AN09
NC	AN10
NC	AN11
NC	AN12
NC	AN13
NC	AP03
NC	AP04
NC	AP05
NC	AP07
NC	AP08
NC	AP09
NC	AP10
NC	AP11
NC	AP12
NC	AP13
PAD[0]	AJ24
PAD[1]	AP23
PAD[2]	AN23

Table 20: MV64362 Pinout Sorted by Pin Name (Continued)

Signal Name	Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name	Ball#
PAD[3]	AM23	PAD[48]	AM28	RxD[4]	T03	VDD Core	AE06	VDD I/O	AJ27
PAD[4]	AL23	PAD[49]	AL28	RxD[5]	T02	VDD Core	AJ12	VDD I/O	AJ28
PAD[5]	AK23	PAD[50]	AK28	RxD[6]	T01	VDD Core	AJ25	VDD I/O	AJ29
PAD[6]	AP22	PAD[51]	AP27	RxD[7]	U06	VDD CPU	F29	VDD I/O	AK01
PAD[7]	AN22	PAD[52]	AN27	RxDV	R02	VDD CPU	G29	VDD I/O	AK02
PAD[8]	AL22	PAD[53]	AM27	RxErr	R04	VDD CPU	K29	VDD I/O	AK03
PAD[9]	AK22	PAD[54]	AL27	SCK	AB06	VDD CPU	K31	VDD I/O	AL03
PAD[10]	AP21	PAD[55]	AK27	SDA	AB05	VDD CPU	K33	VDD I/O	AP06
PAD[11]	AN21	PAD[56]	AP26	SERR#	AL20	VDD CPU	L34	VREF	AK24
PAD[12]	AM21	PAD[57]	AN26	SSTL_ VREF0	D24	VDD CPU	M29	VREF	AN06
PAD[13]	AL21	PAD[58]	AM26	SSTL_ VREF1	E10	VDD CPU	P29	VSS	F18
PAD[14]	AK21	PAD[59]	AL26	StartBurst	B19	VDD CPU	T29	VSS	P14
PAD[15]	AP20	PAD[60]	AK26	StartBurstIn	A19	VDD CPU	AA29	VSS	P15
PAD[16]	AP18	PAD[61]	AJ26	STOP#	AP19	VDD CPU	AC29	VSS	P16
PAD[17]	AN18	PAD[62]	AP25	SysClk	E05	VDD CPU	AE29	VSS	P17
PAD[18]	AM18	PAD[63]	AN25	SysRst#	D05	VDD CPU	AG29	VSS	P18
PAD[19]	AL18	PAR	AM20	TA#	L31	VDD DRAM	F07	VSS	P19
PAD[20]	AK18	PAR64	AM25	TBST#	N31	VDD DRAM	F08	VSS	P20
PAD[21]	AP17	PAVDD	AM14	TRDY#	AM19	VDD DRAM	F10	VSS	P21
PAD[22]	AN17	PAVSS	AN14	TS#	N30	VDD DRAM	F14	VSS	R14
PAD[23]	AM17	PCI_CAL	AN33	TSIZ[0]	N34	VDD DRAM	F16	VSS	R15
PAD[24]	AP16	PCLK	AP14	TSIZ[1]	N33	VDD DRAM	F19	VSS	R16
PAD[25]	AN16	PCLK	AE04	TSIZ[2]	N32	VDD DRAM	F21	VSS	R17
PAD[26]	AM16	PERR#	AK20	TT[0]	M34	VDD DRAM	F23	VSS	R18
PAD[27]	AL16	Pull-up	AJ01	TT[1]	M33	VDD DRAM	F27	VSS	R19
PAD[28]	AK16	Pull-up	AK04	TT[2]	M32	VDD DRAM	F28	VSS	R20
PAD[29]	AJ16	Pull-up	AK07	TT[3]	M31	VDD I/O	G06	VSS	R21
PAD[30]	AP15	Pull-up	AL02	TT[4]	M30	VDD I/O	H06	VSS	T14
PAD[31]	AN15	Pull-up	AL14	TxClk	P01	VDD I/O	M06	VSS	T15
PAD[32]	AP31	RAS#	D20	TxClkOut	P04	VDD I/O	N06	VSS	T16
PAD[33]	AN31	Ready#	B02	TxD[0]	P05	VDD I/O	R06	VSS	T17
PAD[34]	AM31	REQ0#	AM15	TxD[1]	P06	VDD I/O	AA06	VSS	T18
PAD[35]	AL31	REQ64#	AM24	TxD[2]	N01	VDD I/O	AC06	VSS	T19
PAD[36]	AP30	RES	C05	TxD[3]	N02	VDD I/O	AG06	VSS	T20
PAD[37]	AN30	RST#	AK15	TxD[4]	N03	VDD I/O	AH06	VSS	T21
PAD[38]	AM30	RST#	AE02	TxD[5]	N04	VDD I/O	AH29	VSS	U14
PAD[39]	AL30	RstCfg[0]	V02	TxD[6]	N05	VDD I/O	AJ07	VSS	U15
PAD[40]	AK30	RstCfg[1]	V03	TxD[7]	M01	VDD I/O	AJ08	VSS	U16
PAD[41]	AP29	RstCfg[2]	V04	TxEn	P03	VDD I/O	AJ10	VSS	U17
PAD[42]	AN29	RstCfg[3]	V05	TxErr	P02	VDD I/O	AJ13	VSS	U18
PAD[43]	AM29	RxClk	R03	VDD Core	F12	VDD I/O	AJ14	VSS	U19
PAD[44]	AL29	RxD[0]	R01	VDD Core	F25	VDD I/O	AJ17	VSS	U20
PAD[45]	AK29	RxD[1]	T06	VDD Core	K06	VDD I/O	AJ20	VSS	U21
PAD[46]	AP28	RxD[2]	T05	VDD Core	L29	VDD I/O	AJ21	VSS	V14
PAD[47]	AN28	RxD[3]	T04	VDD Core	AD29	VDD I/O	AJ23	VSS	V15

Table 20: MV64362 Pinout Sorted by Pin Name (Continued)

Signal Name	Ball#
VSS	V16
VSS	V17
VSS	V18
VSS	V19
VSS	V20
VSS	V21
VSS	V29
VSS	W01
VSS	W02
VSS	W06
VSS	W14
VSS	W15
VSS	W16
VSS	W17
VSS	W18
VSS	W19
VSS	W20
VSS	W21
VSS	Y01
VSS	Y02
VSS	Y03
VSS	Y04
VSS	Y05
VSS	Y06
VSS	Y14
VSS	Y15
VSS	Y16
VSS	Y17
VSS	Y18
VSS	Y19
VSS	Y20
VSS	Y21
VSS	AA01
VSS	AA02
VSS	AA03
VSS	AA04
VSS	AA05
VSS	AA14
VSS	AA15
VSS	AA16
VSS	AA17
VSS	AA18
VSS	AA19
VSS	AA20
VSS	AA21
VSS	AE01
VSS	AG01
VSS	AJ18
WE#	E20

Section 7. Address Space Decoding

The MV64360/1/2 has a fully programmable address map. There is a separate address space to each of the device interfaces:

- CPU address space.
- PCI_0 address space.
- PCI_1 address space.
- Ethernet Controller address space.
- MPSCs address space.
- IDMAs address space.

Note

The PCI_1 interface is only available in the MV64360 and MV64361 devices.

Each interface includes programmable address windows that allows it to access any of the MV64360/1/2 resources. Each window can map up to 4GB of address space.

Note

The MV64360/1/2 address decoding is NOT software compatible with GT-64240/GT-64260 address decoding scheme. The address windows are no longer defined by base and top registers. Instead, the address windows are defined by base and size registers, similar to the PCI address decoding scheme. Moreover, the Ethernet Controller, the MPSCs, and the IDMAs have their own address decoding registers. This means they no longer use the CPU interface address decoders.

7.1 CPU Address Decoding

In the MV64360 and MV64361devices, the CPU interface address decoding map consists of 21 address windows for the different devices, as shown in Table 21.

Note

The MV64362 device only has 16 address windows, since it does not use the PCI_1 interface or integrated SRAM.

Each window can have a minimum of 64 KB of address space, and up to 4GB.

Table 21: CPU Interface Address Decoder Mappings
NOTE: The PCI_1 interface and integrated SRAM are only available in the MV64360 and MV64361 devices.

CPU Decoder	Associated Target
CS[3:0]*	DDR SDRAM chip selects.

Table 21: CPU Interface Address Decoder Mappings
NOTE: The PCI_1 interface and integrated SRAM are only available in the MV64360 and MV64361 devices.

CPU Decoder	Associated Target
DevCS[3:0]#, BootCS#	Devices chip selects.
PCI_0 I/O	PCI_0 I/O space.
PCI_0 Mem 0/1/2/3	PCI_0 Memory space.
PCI_1 I/O	PCI_1 I/O space.
PCI_1 Mem 0/1/2/3	PCI_1 Memory space.
Internal	MV64360/1/2 internal registers.
Integrated SRAM	Integrated SRAM

Each address window is defined by a Base and a Size register, see A.3 "CPU Address Decode Registers" on page 443. The CPU address is compared with the values in the various CPU Base and Size registers.

Note

In the following section, bit[35] of the CPU address refers to the Most Significant Bit (MSB) address. This is opposite of the PowerPC convention, in which A[0] is the Most Significant Bit (MSB) address and A[35] is the Least Significant bit address.

The CPU address windows are defined in 64 KB granularity. The Base Address is 20-bits wide, corresponding to CPU address bits[35:16], and the Size is 16-bits wide, corresponds to address bits[31:16]. The Size register must be programmed as a set of '1's (staring from the LSB) followed by a set of '0's. The set of '1's defines the size. For example, if Size[15:0] is set to 0x03ff, it defines a size of 64 MB (number of '1's is 10, 2^{10} x 64 KB = 64 MB).

Address decoding starts with the CPU address being compared with the values in the various Base Address registers. The Size register sets which address bits are significant for the comparison. In the previous example of a 64 MB size, the CPU address bits[35:26] are compared against Base Address bits[19:10] (the Size register masks address bits[25:0]). An address is considered as a window hit if it matches the Base Address register bits (the bits which are not masked by the Size register)

Based on the hit indication, the CPU interface transfers the transaction to the required target interface (DRAM, Device...).

Note

Never program the Base and Size registers so that they result in an address windows overlap.

The CPU address decoding scheme restricts the address window to a size of 2^n and to a start address aligned to the window size.

Address bits[35:32] are only relevant for the MPC7450 extended address mode.

The Base Address Enable register's `En` bits [20:0] (Table 383 on page 540) allows for the selective disabling of each window, thus preventing waste of address space for non used resources. The register has one enable bit per window.

- To enable a window, set the corresponding bit to '0'.
- To disable a window, set the corresponding bit 1.

Table 22 shows the default CPU memory map following RESET de-assertion.

Table 22: CPU Default Address Mapping
NOTE: The PCI_1 interface and integrated SRAM are only available in the MV64360 and MV64361 devices.

Decoder	Address Range
CS[0]#	0x0 to 0x007F.FFFF 8 MB
CS[1]#	0x0080.0000 to 0x00FF.FFFF 8 MB
CS[2]#	0x0100.0000 to 0x017F.FFFF 8 MB
CS[3]#	0x0180.0000 to 0x01FF.FFFF 8 MB
DevCS[0]#	0x1C00.0000 to 0x1C7F.FFFF 8 MB
DevCS[1]#	0x1C80.0000 to 0x1CFF.FFFF 8 MB
DevCS[2]#	0x1D00.0000 to 0x1DFF.FFFF 16 MB
DevCS[3]#	0xFF00.0000 to 0xFF7F.FFFF 8 MB
BootCS#	0xFF80.0000 to 0xFFFF.FFFF 8 MB
Internal Registers	0x1400.0000 to 0x1400.FFFF or F100.0000 to 0xF100.FFFF (Reset configuration) 64 KB
PCI_0 Mem0	0x1200.0000 to 0x13FF.FFFF 32 MB
PCI_0 Mem1	0xF200.0000 to 0xF3FF.FFFF 32 MB
PCI_0 Mem2	0xF400.0000 to 0xF5FF.FFFF 32 MB
PCI_0 Mem3	0xF600.0000 to 0xF7FF.FFFF 32 MB

Table 22: CPU Default Address Mapping (Continued)
NOTE: The PCI_1 interface and integrated SRAM are only available in the MV64360 and MV64361 devices.

Decoder	Address Range
PCI_0 I/O	0x1000.0000 to 0x11FF.FFFF 32 MB
PCI_1 Mem0	0x2200.0000 to 0x23FF.FFFF 32 MB
PCI_1 Mem1	0x2400.0000 to 0x25FF.FFFF 32 MB
PCI_1 Mem2	Ox2600.0000 to 0x27FF.FFFF 32MB
PCI_1 Mem3	0x2800.0000 to 0x29FF.FFFF 32MB
PCI_1 I/O	0x2000.0000 to 0x21FF.FFFF 32 MB
Integrated SRAM	0x4200.0000 to 0x4203.FFFF 256 KB

7.1.1 CPU to PCI Address Remapping

The MV64360/1/2 supports CPU address remapping on CPU accesses to the PCI interface. This enables relocating a CPU-to-PCI address window to a new location in the PCI address space, de-coupling the CPU and the PCI memory allocation.

Each of the CPU-to-PCI address windows has an associated 16-bit wide Remap Register (Table 207 on page 456). The upper bits of the CPU address, that are found to be a hit in one of the PCI windows, are replaced by the corresponding bits of the Remap register, before being transferred to the PCI interface unit. The number of bits to be replaced is determined according to the Size register. For the example of 64MB window, CPU address bits[31:26] are replaced with bits[15:10] of the Remap register

Each of the CPU-to-PCI memory windows also has a 32-bit High Remap register (Table 208 on page 456). The High Remap register may be used for 64-bit addressing on the PCI bus. When this register is not set to ‘0’, a CPU address hit in this window results in the MV64360/1/2 PCI master generating a PCI DAC transaction (13.12 “64-bit Addressing” on page 175).

When a Base Address register (A.3 “CPU Address Decode Registers” on page 443) is written to, the associated Remap register bits are simultaneously updated. This effectively sets a 1:1 mapping that means the address transferred to the PCI interface is the same address presented on the CPU bus. For users that do not need this address remapping feature, This feature allows the user to change the CPU interface address decoding windows without working with the associated remap registers.

When a Remap register is written to, only its contents are affected. Following reset, the default value of a Remap register is equal to its associated Base Address register. Unless a specific write operation to a Remap register takes place, a 1:1 mapping is maintained.

When setting the CPU Configuration register’s `RemapWrDis` bit [27] (Table 232 on page 462) to ‘1’, writing to the Base Address register does not result in a simultaneous write to the corresponding Remap registers.

7.1.2 CPU Access Protection Windows

In addition to the address decoding windows, the CPU interface includes four access protection windows. These are used to prevent un-desired CPU accesses to a specific address space.

Each window can be defined to one of the following attributes:

- Write protect: CPU access restricted to reads.
- Access protect: CPU access (read or write) forbidden.
- Cache protect: CPU caching (block read) forbidden.
- No protection: Any CPU access allowed.

Each access window is defined by a Base and a Size registers (see A.3 “CPU Address Decode Registers” on page 443). The CPU address is compared against these registers. If the CPU attempts to access a protected address space (e.g. block read from a cache protect region), the following occurs:

1. The MV64360/1/2 latches the address into the CPU Error Address registers.
2. The CPU AccErr bit in the interrupt cause register is set.
3. An interrupt is asserted (if not masked).

If an access protection is violated, the transaction is not transferred to the target interface. If the violation occurs during a write transaction, the transaction on the CPU bus completes but the write data is lost. If the violation occurs during a read transaction, the MV64360/1/2 completes the transaction and drives data of 0xFFFF.FFFF back to the CPU.

7.1.3 CPU Address Decoding Errors

When the CPU tries to access an unmapped address:

- The MV64360/1/2 latches the address into the CPU Error Address registers.
- The CPU AddrErr bit in the interrupt cause register is set.
- An interrupt is asserted (if not masked).

This feature is especially useful during software debug, when errant code causes fetches from unsupported addresses.

If an address decoding error happens, the transaction is not transferred to the target interface. If the error occurs during a write transaction, the transaction on the CPU bus completes but the write data is lost. If the error occurs during a read transaction, the MV64360/1/2 completes the transaction and drives data of 0xFFFF.FFFF back to the CPU.

7.1.4 CPU Programming of Address Windows Registers

The CPU setting of the CPU interface address decoding registers requires special care, especially if changing the mapping of the MV64360/1/2 internal registers space. If, for example, the CPU changes the Internal Register Base Address register and accesses the internal registers based on the new address, the CPU might get an address mismatch, since the register is not updated yet.

To change the Internal Register Base Address register, perform the following steps:

1. If the required new value overlaps another address window, disable this address decoder through the Address Window Enable register’s.
2. Read the Internal Register Base Address register. This guarantees that all previous transaction in the CPU interface pipe are flushed.

3. Program the register to its new value.
4. Read polling of the register. If the new value is not updated, there is an address mismatch and data of 0xFFFFFFFF is returned.

Note

The Address mismatch interrupt must be masked to prevent a CPU interrupt.

5. Once a valid data is being read, the software continues to program the MV64360/1/2 registers using the internal registers window new base address.

Notes

- Instead of step 4, it is possible to use a wait loop of eight SysClk cycles.

7.2 PCI Address Decoding

In the MV64360 and MV64361 devices, the PCI slave interface address decoding map consists of 16 address windows for the different devices, as shown in Table 23.

Note

The MV64362 only has 12 address windows, since it does not support P2P or integrated SRAM.

Table 23: PCI Interface Address Decoder Mappings
NOTE: The P2P feature and integrated SRAM are only available in the MV64360 and MV64361 devices.

PCI_0 Slave Decoder	Associated Target
CS[3:0]#	SDRAM chip selects.
DevCS[3:0]#, BootCS#	Devices chip selects.
P2P Mem 0/1	Second PCI bus memory space.
P2P I/O	Second PCI bus I/O space.
Internal Mem	Memory mapped internal registers.
Internal I/O	I/O mapped internal registers.
Integrated SRAM	Integrated SRAM
CPU[1]	CPU bus.

1. Used for PCI mastering on the 60x bus (PCI to CPU bridging).

Each window can have a minimum of 4 KB of address space and up to 2 GB space. Each window is defined by a Base and a Size register. The PCI address window is defined in 64 KB granularity. The PCI address bits[31:16] are compared against the values in the various Base and Size registers.

PCI address decoding is similar to the CPU address decoding scheme. An address is considered as window hit if it matches the Base Address register bits (the bits which are not masked by the Size register)

The MV64360/1/2 PCI slave transfers the transaction to the required target interface (DRAM, Device...), based on the hit indication.

Each of the Base Address registers (except for the I/O windows) are 64-bit wide, to support 64-bit addressing (PCI DAC transactions). This does not necessarily mean that the MV64360/1/2 resources can be accessed from PCI only by using DAC transactions. If the upper 32-bits of the Base Address register is set to ‘0’ (which is the default), the window is used for 32-bit addressing accessed via PCI Single Address Cycle (SAC) transactions (13.12 “64-bit Addressing” on page 175).

Notes

- 64-bit addressing is only supported by the MV64360 and MV64362 devices. The MV64361 uses a 32-bit PCI bus.
- Never program the Base and Size registers so that they result in an address windows overlap.
- The PCI address decoding scheme restricts the address window to a size of 2^n and to a start address aligned to the window size.

The PCI Base Address Registers Enable register (Table 353 on page 526) allows disabling address windows, thus preventing waste of PCI address space for unused resources. The register has one enable bit per window. To disable a window, set the corresponding bit in the Base Address Enable register to ’1’.

Table 24 shows the default PCI memory map following RESET de-assertion.

Table 24: PCI Default Address Mapping
NOTE: The P2P feature and integrated SRAM are only available in the MV64360 and MV64361 devices.

Decoder	Address Range
CS[0]#	0x0 to 0x007F.FFFF 8 MB
CS[1]#	0x0080.0000 to 0x00FF.FFFF 8 MB
CS[2]#	0x0100.0000 to 0x017F.FFFF 8 MB
CS[3]#	0x0180.0000 to 0x01FF.FFFF 8 MB
DevCS[0]#	0x1C00.0000 to 0x1C7F.FFFF 8 MB
DevCS[1]#	0x1C80.0000 to 0x1CFF.FFFF 8 MB
DevCS[2]#	0x1D00.0000 to 0x1DFF.FFFF 16 MB
DevCS[3]#	0xFF00.0000 to 0xFF7F.FFFF 8 Megabyte
BootCS#	0xFF800000 to 0xFFFF.FFFF 8 MB

Table 24: PCI Default Address Mapping (Continued)
NOTE: The P2P feature and integrated SRAM are only available in the MV64360 and MV64361 devices.

Decoder	Address Range
Internal Mem	0x1400.0000 to 0x1400.0FFF 4 KB
Internal I/O	0x1400.0000 to 0x1400.0FFF 4 KB
P2P Mem0	PCI_0: 0x2200.0000 to 0x23FF.FFFF PCI_1: 0x1200.0000 to 0x13FF.FFFF 32 MB
P2P Mem1	PCI_0: 0x2400.0000 to 0x25FF.FFFF PCI_1: 0xF200.0000 to 0xF3FF.FFFF 32 MB
P2P I/O	PCI_0: 0x2000.0000 to 0x21FF.FFFF PCI_1: 0x1000.0000 to 0x11FF.FFFF 32 MB
Integrated SRAM	0x4200.0000 to 0x4203.FFFF 256 KB
CPU	0x4000.0000 to 0x41FF.FFFF 32 MB

7.2.1 PCI to Memory Address Remapping

The MV64360/1/2 supports PCI address remapping on PCI accesses to the DRAM, Device, or integrated SRAM. This enables relocating a PCI address window to a new location in the MV64360/1/2 address space (de-couple CPU and PCI memory allocation).

Note

The integrated SRAM support is only available in the MV64360 and MV64361 devices.

Each of the PCI address windows has a Remap Register associated with it. The upper bits of the PCI address, that is found to be a hit in one of the PCI windows, are replaced by the corresponding bits of the remap register, before transferred to the target interface. The number of bits to be replaced is determined according to the Size register

On P2P transactions, the High Remap register (Table 364 on page 529) is used for 64-bit addressing on the PCI bus. It allows for converting a SAC transaction responded on one PCI interface to a Double Address Cycle (DAC) transaction on the other PCI interface, or even DAC-to-DAC bridging to different offsets in the PCI memory space.

Note

The P2P feature is only available in the MV64360 and MV64361 devices. The MV64362 only has one PCI interface.

When a BAR register is written to, a write transaction simultaneously takes place in the associated Remap register. This effectively sets a 1:1 mapping (meaning the address transferred to the target interface is the same address presented on the PCI bus). When a Remap register is written to, only its contents are affected.

Following reset, the default value of a Remap register is equal to its associated Base Address register. Unless a specific write operation to a Remap register takes place, a 1:1 mapping is maintained.

In some applications, the operating system might re-program the Base Address registers after the Remap registers have been programed by the local driver. In such cases, the 1:1 mapping due to the BARs re-programing is not desired.

If the PCI Address Decode Control register's `RemapWrDis` bit [0] in (Table 372 on page 531) is set to '1', writing to the BARs does not result in a simultaneous write to the corresponding Remap registers.

7.2.2 PCI Access Protection Windows

In addition to the address decoding windows, the PCI interface includes six access protection windows. These are used to prevent undesired PCI accesses to a specific address space.

Each window can be defined to one of the following attributes:

- Write protect: PCI access restricted to reads.
- Access protect: PCI access (read or write) forbidden.
- No protection: Any PCI access allowed.

Each access window is defined by Base and Size registers. The PCI address is compared against these registers. In case the PCI attempts to access a protected address space (e.g. write to a read only region), the following occurs:

1. The MV64360/1/2 latches the address into the PCI Error Address registers.
2. The PCI STabort bit in the interrupt cause register is set.
3. An interrupt is asserted (if not masked).

If an access protection is violated, the MV64360/1/2 PCI slave terminates the transaction with Target Abort (.

7.2.3 PCI Base Address Registers Programming

PCI Base Address and Size registers can be programmed by CPU, or by some host from the PCI bus.

PCI write access to the MV64360/1/2 PCI configuration registers is non-posted. However, write access to the internal registers are treated as posted writes. This means that when the transaction is completed on the PCI bus there is no guarantee that the register is updated with its new value.

The PCI Base Address registers are located in the MV64360/1/2 configuration space. However, the Size and Remap registers are part of the internal registers space. This means, that there is a potential race between PCI programing of these registers, and consecutive access to the MV64360/1/2 memory space, based on the new programing.

Note

For further details about PCI address decoding, see 13.4 "PCI Target Address Decoding" on page 165.

7.3 IDMA Address Decoding

The four IDMA channels share a single address decoding logic. This logic consists of eight address windows. Whenever a DMA is activated, the address generated by the DMA controller is compared against these address windows to determine which interface must be accessed.

Note

Unlike the GT-6424x\6x devices, in which the IDMA shared the CPU address decoding windows, the MV64360/1/2 IDMAs have their own dedicated address decoding windows. It is possible to have the same IDMA and CPU address map by programing the IDMA address decoding registers to the same values of the CPU address decoders.

Since the IDMAs address decoding is not coupled with CPU address decoding, the PCI override feature that was implemented in the GT-6424x\6x devices, is no longer supported in MV64360/1/2.

Each address window is defined by a Base register (Table 596 on page 664) and a Size register (Table 597 on page 666). The IDMA address is compared with the values in the various DMA Base and Size registers. The address comparison mechanism is the same one used by the CPU and PCI address decoders, see 7.1 “CPU Address Decoding” on page 82).

Unlike the CPU or PCI interfaces, in which each window is coupled to a specific target, the IDMAs address windows targets can be programmed. Each of the eight windows can be configured to a specific target interface, via the Base Address register’s `Target` bits[3:0].

Unlike the CPU or PCI interfaces that might need access to all of the MV64360/1/2 resources, IDMAs typically work against very specific interfaces and, therefore, do not need as many address windows. On the other hand, the programmable target per window feature, allows allocating multiple, non-consecutive address windows to the same target interface.

Four out of the eight windows have also a High Address Remap register (Table 598 on page 666). Use these registers to generate an address beyond the standard 4GB space. This is useful for 64-bit addressing when accessing the PCI or for mastering the CPU 60x bus in case of interfacing the MPC7450 in extended address mode.

Note

For further details about IDMA address decoding, see 18.3 “IDMA Address Decoding” on page 302.

7.3.1 IDMA Access Protection

Each of the four IDMA channels has a Access Protect register (Table 600 on page 667). This register defines which of the eight address decoding windows is accessible for this channel. Used this feature to prevent incorrect IDMA accesses to CPU or PCI data structures. It can also be used for preventing one channel interfering with the data structure of another channel.

When an IDMA channel attempts to access a protected region, the following occurs:

1. The MV64360/1/2 latches the address into the DMA Error Address register, including failing DMA channel indication.
2. The DMA AccErr bit in the Interrupt Cause register is set.
3. An interrupt is asserted (if not masked).
4. The channel halts.

7.4 Gigabit Ethernet Address Decoding

The MV64360 has two GbE ports, plus a third port that is multiplexed on the upper pins of PCI_1 interface. The MV64361 has only two GbE ports. And, the MV64362 has one GbE port.

The GbE ports share a single address decoding logic consisting of six address windows. Whenever one of the ports generates a read or a write transaction (e.g. fetch descriptor), the address is compared against these address windows, to determine which interface must be accessed.

The address decoding scheme is the same as the IDMA logic with one exception. In case of a read from an erroneous address, the Ethernet controller SDMA operation is not terminated. The read is retargeted to a pre-defined address. The GbE controller regards the data read from this pre-defined address as bad data and terminates the transaction as if it detected a parity or ECC error.

Note

For further details about GbE address decoding, see 15.4.1 “Address Decoding” on page 213.

7.5 MPSCs Address Decoding

The two MPSCs share a single address decoding logic. This logic consists of four address windows. Whenever one of the ports generates a read or a write transaction (e.g. fetch descriptor), the address is compared against these address windows to determine which interface must be accessed.

The address decoding scheme is the same as the IDMA logic.

Note

For further details about MPSC address decoding, see 16.2.1 “Data Encoding/Decoding” on page 253.

7.6 Headers Retarget

In the typical application, Rx packets (coming from PCI agents or from the MV64360/1/2 GbE ports) are placed in DRAM, manipulated by the CPU, and then placed in Tx queues in DRAM for transmit.

The CPU handles the Rx and Tx descriptors (which can reside in DRAM or integrated SRAM). Also, the CPU examines the packet header and performs an action based on the header content. Typically, the CPU only needs to read the header. It does not need to read and manipulate the entire packet.

Since DRAM read latency is often a performance bottle neck, optimal system performance is achieved by placing packet headers in the low latency integrated SRAM and placing the rest of the packet in DRAM. The headers retarget scheme enables re-directing of the headers to the integrated SRAM, in a way, which is transparent to the application software. From the software point of view, the whole packet goes to DRAM.

Note

The headers retarget scheme supports a fixed size header of 64-bytes, meaning, the first 64 bytes of the buffer, are retargeted to the integrated SRAM

The headers retarget scheme can only work with fixed size aligned buffers.

7.6.1 Headers Retarget Registers

Headers retarget is supported on the PCI_0, PCI_1, GbE, IDMA, and CPU Interface units. The PCI0, PCI1, GbE, and IDMA units use these registers to identify the headers space.

Notes

- The PCI_1 interface unit is only supported in the MV64360 and MV64361 devices. The MV64362 only has one PCI interface.

Table 25: Headers Retarget Control

Bits	Field	Type	Description
0	En	0x0	Headers Retarget Enable 0 = Disable 1 = Enable
3:1	BSize	0x0	Buffer Size 0 = 256 bytes 1 = 512 bytes 2 = 1 KB 3 = 2 KB 4 = 4 KB 5 = 8 KB 6 - 7= Reserved
15:4	Reserved	0x0	Reserved.
31:16	Mask1	0x0	In 64 KB granularity, defines the total space of the buffers to be manipulated. Size must be set from LSB to MSB as a sequence of 1's, followed by sequence of 0's. For example, to retarget the headers of 1K buffers of 1 KB size, which means 1MB of buffers space, Mask1 must be set to 0x000f. **NOTE:** The total address space of retargeted headers must not exceed the integrated SRAM size (256 KB). The minimum buffers space to be manipulated is 64 KB.

Table 26: Header Retarget Base

Bits	Field	Type	Description
15:0	Reserved	0x0	Reserved.
31:16	Base	0x0	Base Address Retarget is executed if the address bits, which are not masked by Mask1, match the corresponding bits of the base. For example, if Mask1 is set to 0x000f (1MB of buffers), retarget is executed if address bits[31:20] are equal to Base[31:20].

The two PCI units, use a third register, which defines the upper 32-bit base address for the case the buffers are mapped above the 4GB space (DAC transactions).

Table 27: Header Retarget Base High

Bits	Field	Type	Description
31:0	BaseH	0x0	Base address Corresponds to address bits[63:32].

In the CPU interface unit, there are four pairs of registers. Each register pair corresponds to one of the PCI0, PCI1, GbE or IDMA units. These registers are used by the CPU interface to identify the headers space for transactions coming from the CPU, but also used by the integrated SRAM controller for remapping of the addresses (placing the headers consecutively).

Table 28: CPU Header Retarget Base and Remap

Bits	Field	Type	Description
3:0	Remap	0x0	Remap Address Defines the upper bits of headers space in SRAM.
7:4	Mask2	0x0	Defines how many of the integrated SRAM address upper bits must be remapped. If for example, the headers space is 64 KB (which is one quarter of the total SRAM space), Mask2 can be set to b'0011. This means that the two MSB bits of SRAM address (address bits[17:16]) are replaced with the corresponding Remap bits.
15:12	BaseH	0x0	High Base Address Corresponds to address bits[35:32].
31:16	Base	0x0	Base Address Retarget is executed if the address bits, unmasked by Mask1, matches the corresponding Base bits. Corresponds to CPU address bits[31:16].

Notes

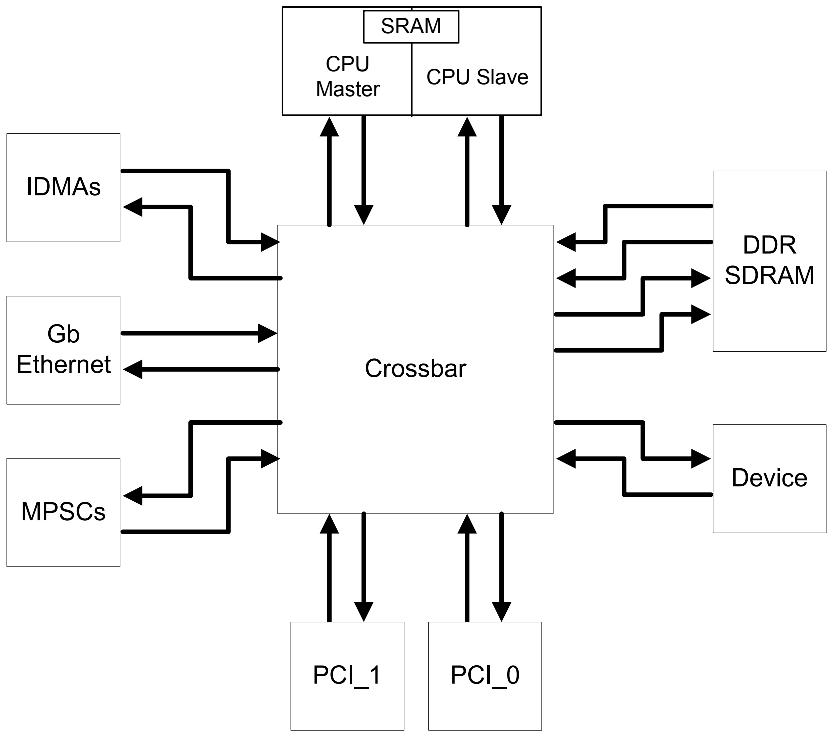
- For the retarget scheme to work properly, the programing of the CPU interface registers must be the same as the programing in the corresponding unit register. Both registers must have same BSize and Mask1 settings, and should map the same DRAM space.
- It is the user responsibility to have the same cache coherency policy for DRAM and SRAM. If for example, the packets are placed in a cache coherent WB DRAM region, also configure the integrated SRAM to WB cache coherency policy.
- When using Headers Retarget, the internal burst size of the units participating in this scheme must not exceed 64 bytes. When using cache coherency, the burst is limited to 32 bytes.
- If the integrated SRAM is also used for other tasks (such as descriptors memory), the retargeting scheme must not cause headers to override other data in SRAM.
- If the retargeting scheme is used for transferring headers between multiple agents (e.g. GE MAC Rx headers written to SRAM, processed by CPU, and being read by a PCI adapter), set the multiple CPU Header Retarget windows to the same setting (PCI and GE windows for the above example) and set En bit only to one of the windows (if setting En to multiple windows that have the same setting, an error occurs).

Section 8. Internal Crossbar

The MV64360/1/2 internal architecture is based on a 64-bit full duplex data path connecting the different units. This internal architecture allows concurrent data transfers between different interfaces (for example, CPU read from DRAM, PCI_0 read from device and IDMA write to PCI_1 at the same time) and transaction pipelining (issue multiple transactions in parallel between the same source and destination).

Figure 11 shows how the data path routing is controlled via the MV64360/1/2 crossbar.

Figure 11: MV64360/1/2 Crossbar

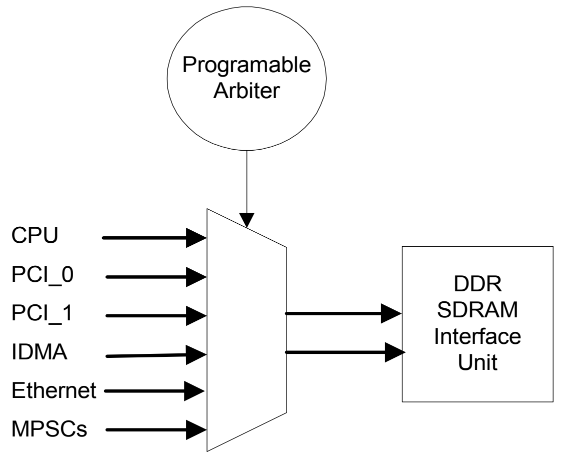


Note

The PCI_1 interface only applies to the MV64360 and MV64361 devices.

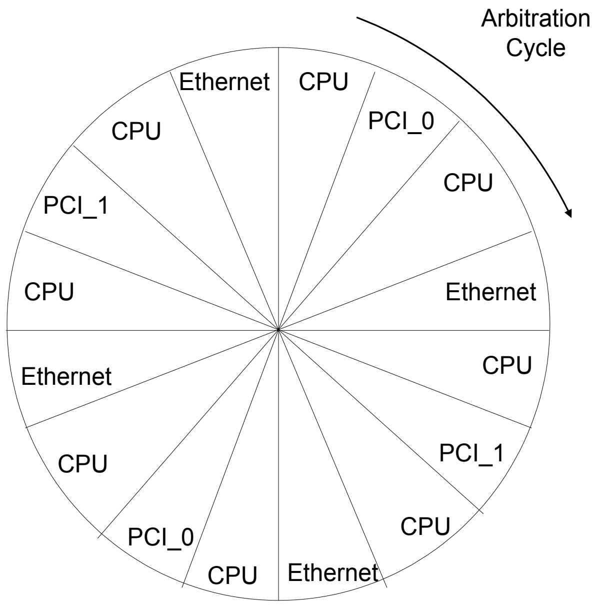
Sometimes conflicts may occur over resources. For example, if the CPU, PCI_0, and GbE unit request access to the DRAM simultaneously, these requests cannot be served at the same time. The crossbar contains programmable arbitration mechanisms to optimize device performance, according to the system requirements, as shown in Figure 12.

Figure 12: SDRAM Interface Arbitration



Each arbiter is a user defined round-robin arbiter (called a “pizza arbiter”). Figure 13 shows an example of the Device interface arbiter setting.

Figure 13: Configurable Weights Arbiter



It is possible to define each of the 16 slices of this “pizza arbiter”. This arbiter is working on a transaction basis (a transaction can vary from one up to 16 64-bit words). In the above example, if all units are requesting constantly access to the device bus, out of N transactions running on the device bus, 50% will be CPU transactions, 25% will be Gb Ethernet controller transactions, 12.5% will be PCI_0 transactions and 12.5% will be PCI_1 transactions.

This “pizza” configuration also allows the user to guarantee minimum latency. Even if the CPU does not require 50% transactions allocation, the above configuration guarantees that in the worst case, the CPU request needs to wait for one access of another unit before being served.

At each clock cycle, the crossbar arbiter samples all requests and gives the bus to the next agent according to the “pizza”. It is parked on the last access.

The exact registers settings can be found in the CPU, PCI, DRAM, Device units registers sections.

An arbiter slice can also be marked as NULL. If marked as NULL, the arbiter works as if the NULL slice does not exist. For example, if only two requests are used that need same bandwidth, it is possible to specify the first slice per one request, the second slice per the other request, and all of the other slices as NULL. This is equivalent to specifying half of the slices for one request and the other half for the other request.

Note

Once a unit is removed from an interface’s “pizza” arbiter control register, this unit has no access to this interface. If for example, the MPSCs unit is removed from the DRAM interface “pizza” arbiter, this unit can no longer accesses the DRAM. If it attempts to access the DRAM, the unit hangs.

Section 9. CPU Interface

The MV64360/1/2 supports PowerPC 64-bit bus CPUs. These include:

- Motorola MPC603e, MPC604e, MPC740/750/755
- Motorola MPC7400/7410/7450
- IBM PPC603e, PPC750/750Cx/750Fx
- Motorola PowerQUICC II (MPC8260)
- Any 64-bit 60x or MPX bus compatible CPU

The CPU interface can work as a slave interface, responding to CPU transactions, or as a master interface, generating CPU like transactions. The master interface is used for PowerPC snoops generation. It also allows data transfers to/from other slaves on the bus when running in 60x bus mode

9.1 CPU Address Decoding

This section summarizes CPU address decoding. For full details, see 7.1 “CPU Address Decoding” on page 82.

In the MV64360 and MV64361 devices, the CPU interface supports 21 address windows for the different devices. The MV64362 device supports 16 address windows, since it does not use the PCI_1 interface or integrated SRAM.

The CPU interface supports 21 address windows.

- Four for SDRAM chip selects
- Five for device chip selects
- Five for PCI_0 interface (4 memory + one I/O)
- Five for PCI_1 interface (4 memory + one I/O) (MV64360 and MV64361 only)
- One for the MV64360/1/2 integrated SRAM (MV64360 and MV64361 only)
- One for the MV64360/1/2 internal registers space

Each window is defined by a Base and a Size register (A.3 “CPU Address Decode Registers” on page 443). Each window has a minimum of 64 KB of address space, and up to 4 GB (except of the integrated SRAM which is fixed 256 KB).

The CPU interface also supports address remapping to the PCI bus. This is useful when a CPU address range must be reallocated to a different location on the PCI bus. Also, it enables a CPU access to a PCI agent located above the 4GB space.

The CPU interface contains High PCI Remap registers that define the upper 32-bit PCI address. If the register is set to ‘0’, the CPU access to PCI results in a SAC transaction. If it is set to a value other than ‘0’, the PCI master issues a DAC transaction with the high 32 address bits set according to the High PCI Remap register’s value.

The CPU interface supports configuring the access protection. This includes up to four address ranges defined to a different protection type - whether the address range is cacheable or not, whether it is writable or not, or whether it is accessible or not.

If there is an access violation, the CPU interface completes the transaction against the CPU but ignores the transaction internally. The transaction latches the address in the CPU Error Address register and sets the Interrupt Cause register’s `AddrErr` bit.

9.2 CPU Slave Operation

The CPU slave interface contains 256 bytes of posted write data buffer and 256 bytes of read data buffer. It can absorb up to eight write transactions and up to eight outstanding read transactions. In dual CPU configuration, it supports up to 16 outstanding read transactions.

Note

Dual CPU configurations are only supported by the MV64360 and MV64361 devices.

CPU writes are posted. They are written into the write buffer and only then driven to the target. If the target device is busy and cannot accept the transaction, the write buffer can still accept new CPU write transactions, with zero wait states.

The MV64360/1/2 supports split read transactions. The CPU interface pipelines up to 8 transactions to target devices. In this case, data may be returned out of order. For example, if the first read transaction is directed to the PCI and the second is directed to SDRAM, data from SDRAM is returned first.

The CPU interface tries to drive read data to the CPU when data arrives from the target device. If the CPU supports out of order completion (e.g. MPC7400), the first read data coming from the target interface is the first one to be driven on the CPU bus (SDRAM read data in the above example). If the CPU doesn't support out of order completion (e.g. MPC750), data may be temporarily placed in the read buffer while waiting for read data from a slower interface to complete first. In the above example, the SDRAM data is placed in a read buffer until the read data from PCI arrives and is driven on the CPU bus.

The CPU transactions are issued to the target device in order. The first transaction appearing on the CPU bus is the first one to be issued towards the target device. There is no transaction bypassing. However, each of the MV64360/1/2 interfaces contains large read and write buffers that allow the CPU interface pipe to keep pushing transactions to the different interfaces (e.g. It is unlikely that a CPU transaction to DRAM is stalled, due to a previous CPU transaction to PCI. The PCI master interface absorbs the CPU transaction in its buffers and allows the CPU interface to continue with the DRAM access).

9.3 PowerPC 60x Bus Interface

The MV64360/1/2 supports 64-bit PowerPC CPUs non-multiplexed address/data bus protocol. It supports partial read/writes of one byte up to eight bytes as well as 32-byte cache line reads/writes.

Notes

- According to the PowerPC convention, each signal's Most Significant Bit (MSB) is bit[0] and the Least Significant bit is bit[n]. This convention is used throughout this section.
- In the following sections, it is assumed that the address is 36-bit wide, meaning A[0] is the MSB and A[35] is the LSB. When interfacing a CPU with a 32-bit wide address, A[4] is the MSB and A[35] is the LSB.

9.3.1 Transaction Attributes

On Transfer Start, the CPU drives an address on A[0-35] and transaction attributes on TBST#, TSIZ, and TT, as shown in Table 29.

Table 29: Transfer Size Summary

TBST#	TSIZ[0-2]	Transfer Size
0	010	32 bytes burst
1	000	8 bytes
1	001	1 byte
1	010	2 bytes
1	011	3 bytes
1	100	4 bytes
1	101	5 bytes
1	110	6 bytes
1	111	7 bytes

Table 30: Data Bus Bytes Lane

Transfer Size	A [33-35]	Data Bus Byte Lanes							
		DH [0-7]	DH [8-15]	DH [16-23]	DH [24-31]	DL [0-7]	DL [8-15]	DL [16-23]	DL [24-31]
1 byte	000	A	-	-	-	-	-	-	-
	001	-	A	-	-	-	-	-	-
	010	-	-	A	-	-	-	-	-
	011	-	-	-	A	-	-	-	-
	100	-	-	-	-	A	-	-	-
	101	-	-	-	-	-	A	-	-
	110	-	-	-	-	-	-	A	-
	111	-	-	-	-	-	-	-	A

Table 30: Data Bus Bytes Lane (Continued)

Transfer Size	A [33-35]	Data Bus Byte Lanes							
		DH [0-7]	DH [8-15]	DH [16-23]	DH [24-31]	DL [0-7]	DL [8-15]	DL [16-23]	DL [24-31]
2 bytes	000	A	A	-	-	-	-	-	-
	001	-	A	A	-	-	-	-	
	010	-	-	A	A	-	-	-	-
	011	-	-	-	A	A	-	-	-
	100	-	-	-	-	A	A	-	-
	101	-	-	-	-	-	A	A	-
	110	-	-	-	-	-	-	A	A
3 bytes	000	A	A	A	-	-	-	-	-
	001	-	A	A	A	-	-	-	-
	010	-	-	A	A	A	-	-	-
	011	-	-	-	A	A	A	-	-
	100	-	-	-	-	A	A	A	-
	101	-	-	-	-	-	A	A	A
4 bytes	000	A	A	A	A	-	-	-	-
	001	-	A	A	A	A	-	-	-
	010	-	-	A	A	A	A	-	-
	011	-	-	-	A	A	A	A	-
	100	-	-	-	-	A	A	A	A
5 bytes	000	A	A	A	A	A	-	-	-
	001	-	A	A	A	A	A	-	-
	010	-	-	A	A	A	A	A	-
	011	-	-	-	A	A	A	A	A
6 bytes	000	A	A	A	A	A	A	-	-
	001	-	A	A	A	A	A	A	-
	010	-	-	A	A	A	A	A	A
7 bytes	000	A	A	A	A	A	A	A	-
	001	-	A	A	A	A	A	A	A
8 bytes	000	A	A	A	A	A	A	A	A

Table 31: Transfer Type (TT[0-4]) Encoding

TT[0-4]	Transaction	Bus Cycle	MV64360/1/2 Response
00000	Clean block	Address Only	Ignores
00100	Flush block	Address Only	Ignores
01000	SYNC	Address Only	Ignores
01100	Kill block	Address Only	Ignores
10000	Ordered I/O operation (eieio)	Address Only	Ignores
10100	External control word write (ecowx)	Single-beat write	Not Supported
11000	TLB invalidate (tlbie)	Address Only	Ignores
11100	External control word read (eciwx)	Single-beat read	Not Supported
00001	lwarx reservation set	Address Only	Ignores
00101	Reserved	----	Not Supported
01001	TLB sync	Address Only	Ignores
01101	Invalidate Icache copy (icbi)	Address Only	Ignores
00010	Write with flush	Single beat or burst write	Write
00110	Write with kill	Burst write	Write
01010	Read	Single beat or burst read	Read
01110	Read with intent to modify	Burst read	Read
10010	Write with flush atomic (stwcx)	Single beat write	Write
10110	Reserved	----	Not Supported
11010	Read atomic (lwarx)	Single beat or burst read	Read
11110	Read with intent to modify atomic	Burst read	Read
00X11	Reserved	----	Not Supported
01011	Read with no intent to cache	Single beat or burst read	Read
01111	Read claim	Burst read	Read
1XXX1	Reserved	----	Not Supported

The only transactions the MV64360/1/2 does not support are ECIWX and ECOWX. These transactions are optional in the PowerPC architecture.

Note

An attempt to access the MV64360/1/2 (address match) with these transactions results in unpredictable behavior and the CPU might hang.

Address only transactions targeted to the MV64360/1/2 are completed with AACK#. No further action is taken. No internal transaction is taken nor internal state is changed.

Note

The MV64360/1/2 does NOT support direct store operations. It does not interface XATS# signals and does not support the special decoding of TT, TBST, and TSIZ during these transactions.

9.3.2 60x Read Protocol

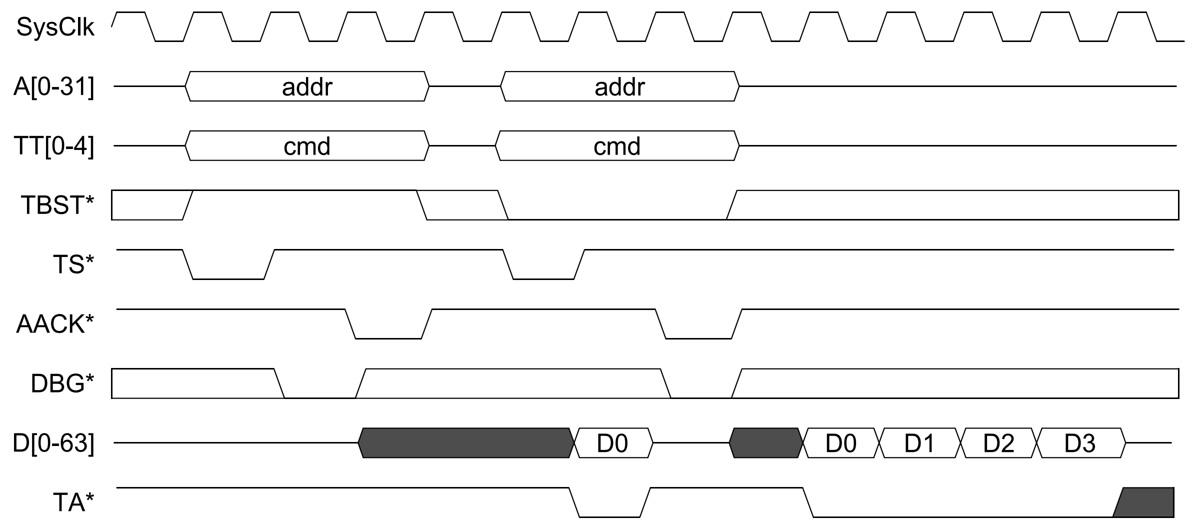
To issue a new transaction, the CPU must first gain bus ownership. It asserts BR# and waits for BG# from the bus arbiter. It may start a new transaction on the cycle after a qualified bus grant. This means BG# is asserted and the address bus is not busy. ABB# is not asserted.

The CPU starts a transaction with by asserting TS# for one cycle. During the same cycle, it asserts ABB# and drives address and attributes. The CPU keeps driving address and attributes until the MV64360/1/2 asserts AACK#. On the next cycle, it floats address bus.

The MV64360/1/2 starts driving the data bus as soon as the bus is granted to the initiating CPU. DBB# is also asserted. As soon as read data is available, the MV64360/1/2 asserts TA# and drives valid data on the data bus (DH, DL).

An example of two consecutive reads is shown in Figure 14.

Figure 14: PowerPC Read Protocol



Note

Figure 14 only demonstrates 60x bus read transactions. It does not reflect the actual read latency of the MV64360/1/2.

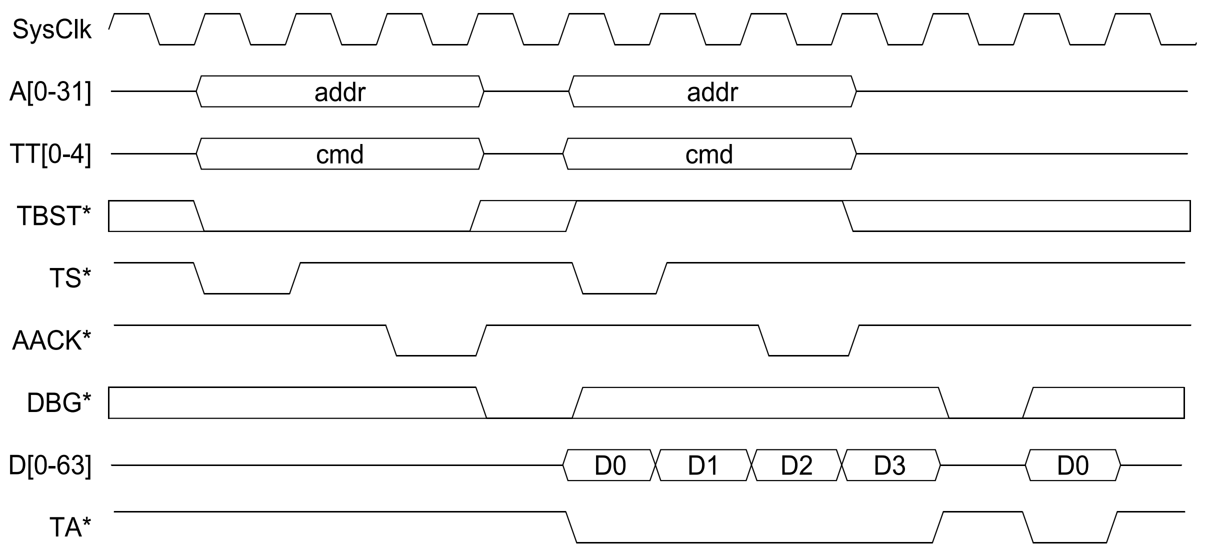
9.3.3 Write Protocol

Similar to a read transaction, the CPU must first gain bus ownership and then initiate an address phase.

As soon as the CPU also gains data bus ownership, the bus arbiter asserts CPU DBG# and the CPU drives valid data on data bus (DH, DL) and asserts DBB#. It keeps driving the bus until the last data.

An example of two consecutive writes is shown in Figure 15.

Figure 15: PowerPC Write Protocol



9.4 Address Pipelining Support

The PowerPC bus protocol supports address pipelining.

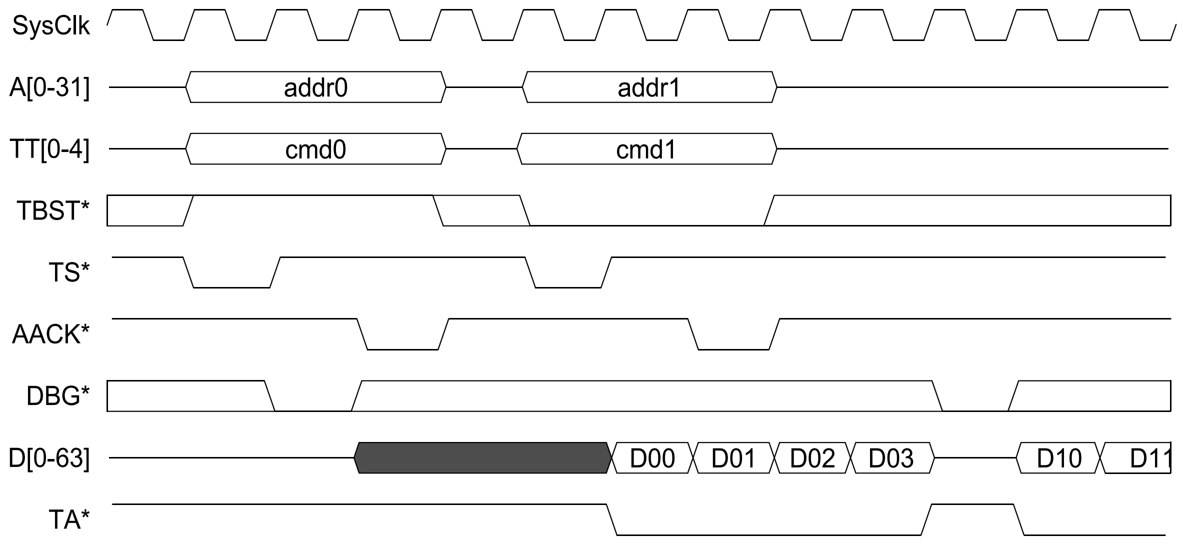
This means that a CPU initiating a transaction is allowed to issue a new transaction address phase before a completion of a previous transaction data phase. Additionally, in a multi-CPU configuration, one CPU can issue a new transaction between the address and data phases of a previous transaction of another CPU. In any case, data flow is always in order.

Note

The MV64360/1/2 does not support the PowerPC DBWO# feature that enables write data tenure to bypass previous transaction read data tenure.

Although the PowerPC bus protocol does not limit pipeline depth, the MV64360/1/2 only supports a pipeline depth of up to eight write transactions and eight read transactions. This is useful in multi-CPU configurations, or when interfacing MPC74xx processors, when it is possible to reach this pipeline depth. An example of two pipeline reads is shown in Figure 16.

Figure 16: PowerPC Pipeline Reads Example



Note

Figure 16 only demonstrates 60x bus pipelined transactions. It does not reflect the actual read latency of the MV64360/1/2.

Internally, the CPU interface unit can handle two transactions in parallel.

In Figure 16, if the two reads are targeted to different SDRAM banks, the MV64360/1/2 takes advantage of the SDRAM interleave feature and data returns to the CPU with minimum latency. There is no wait states between the last data of the first transaction and the first data of the second transaction.

Additionally, if the first read is targeted to a slow device and the second read is targeted to a fast device, the CPU interface pipelines the two requests. Since the 60x bus protocol restricts in order data phases, the MV64360/1/2 must drive the data of the first transaction on the bus and then the data of the second transaction. This means that the data coming from the fast device is temporarily stored in a read buffer. As soon as the data from the slow device is received, it is driven on the data bus. Then, the data temporarily stored in the read buffer is driven on the data bus.

9.5 Burst Support

CPU cache line read or write results in 32 byte burst read/write transaction on the bus.

Note

The MV64360/1/2 also supports MPC74xx 16 byte burst read/write transactions.

Cache line write addresses are always aligned to the cache line (A[32-35] are 0).

Cache line read addresses might point to any of the four double-words of the cache line. The burst read order is linear wrap-around, as shown in Table 32.

Table 32: 64-bit Linear Wrap-Around Burst Order

Data Transfer	Start Address A[31-32]			
	00	01	10	11
1st data beat	DW0	DW1	DW2	DW3
2nd data beat	DW1	DW2	DW3	DW0
3rd data beat	DW2	DW3	DW0	DW1
4th data beat	DW3	DW0	DW1	DW2

Note

Burst transactions to internal space are not supported. This type of transaction results in an interrupt assertion (if not masked) and the CPU Error Cause register setting `TTErr` bit [2] to '1'. The transaction completes normally on the CPU bus. However, on writes, the data is discarded. On reads, random data is returned.

9.6 Transactions Flow Control

The MV64360/1/2 controls the CPU transactions rate using the AACK# signal.

When the CPU interface transaction queue is full, the MV64360/1/2 keeps AACK# de-asserted in response to a new transaction start. AACK# is kept de-asserted until there is "room" for a new transaction in the queue.

Notes

- The MV64360/1/2 fastest AACK# response (when the transaction queue is not full) is programmable to be two or three cycles after TS# assertion. Program the response via the CPU Configuration register's `AACKDelay2` bit [25] (Table 232 on page 462).
- The MV64360/1/2 does not support AACK# delay, in MPX bus mode.

The CPU interface implementation guarantees that once there is room in the transaction queue there is also room in the write buffer to absorb write data. This implies that MV64360/1/2 will never insert wait states (de-assert TA#) during a CPU burst write transaction.

The MV64360/1/2 uses TA# to insert wait states during read transaction. If the CPU accesses a slow device, the MV64360/1/2 keeps TA# de-asserted until read data arrives from the target device. In case of burst read from a slow device, the MV64360/1/2 might insert wait states between data beats by de-asserting TA#.

Note

The MV64360/1/2 fastest TA# response to a write transaction is two cycles after AACK* assertion.

The MV64360/1/2 does not support DRTRY# (it never terminates an outstanding transaction, nor tolerate DRTRY# asserted by other slave device on the bus). The CPU can be configured to no-DRTRY mode, which improves its read latency by one cycle (see your specific CPU Users Manual for more information).

9.7 PowerPC ARTRY#

The MV64360/1/2 never asserts ARTRY#. It only samples ARTRY# assertion by the CPU.

The CPU snoops every address that is driven on the bus, and marked as global (GBL# asserted). This address might be driven by the MV64360/1/2 generating a transaction on the bus or by another CPU in multi-CPU configuration. In multi-CPU configurations, if one CPU responds with an ARTRY# to a transaction targeted to the MV64360/1/2 from another CPU, the MV64360/1/2 ignores the transaction.

If TA# is asserted in the same cycle or one cycle before ARTRY#, the 60x bus definition allows termination of data tenure via ARTRY# assertion. When the CPU transaction is targeted to the MV64360/1/2, the CPU interface never asserts TA# before the ARTRY# window. However, it can tolerate other devices on the bus, such as PowerQUICC II, asserting TA# on ARTRY# window.

Note

The MV64360/1/2 samples ARTRY# three or four cycles after TS# assertion. This depends on the CPU Configuration register's `AACKDelay2` bit [25] setting (see Table 232 on page 462). Setting AACKDelay2 to four cycles is useful for CPUs that have a late ARTRY# response window.

In multi-CPU configuration, where ARTRY# assertion might terminate a transaction that is targeted to one of the MV64360/1/2 resources (e.g. CPU0 performs cache line read from DRAM, that happen to be modified in CPU1 cache), the MV64360/1/2 must wait for the ARTRY# window, to decide whether to execute the transaction. This impacts the CPU read latency.

With a single CPU configuration, there is no latency impact. The MV64360/1/2 does not have to wait for ARTRY# window. To achieve minimal read latency, set CPU Configuration register's `SingleCPU` bit [11] to '1'.

Notes

- The MPC7450 might generate a self-ARTRY# (ARTRY# to its own initiated transaction), in case of a STWCX instruction that lost its reservation. If using the MPC7450 with this instruction, the MV64360/1/2 must be used in multi-CPU mode, even if there is only a single CPU.
- If enabling CPU data intervention, the MV64360/1/2 must be used in multi-CPU mode, even if interfacing a single CPU, since the MV64360/1/2 must wait for the HIT# decide window. This decides whether a data only transaction should be executed or not.
- See 9.10.4 "Multi-CPU Support and Data Intervention" on page 111.

9.8 PowerPC Cache Coherency

The MV64360/1/2 supports full cache coherency between the CPU caches, and the SDRAM and integrated SRAM.

Note

Each PCI, IDMA, MPSC or Ethernet controller access to the integrated SRAM or to the SDRAM might result in a snoop transaction on the CPU bus.

Maintaining cache coherency might cause some performance loss. To minimize the snoop penalty, the MV64360/1/2 supports configuring DRAM address windows to maintain cache coherency. Each region can be defined as a WB or WT region.

Note

The cache coherency on a WT region has less performance penalty than a WB defined region.

For full description of the snoop process, see Section 20. "PowerPC Cache Coherency" on page 322.

9.9 PowerQUICC II Support

The MPC8260 (PowerQUICC II) is a 60x bus compliant CPU.

The only difference from the other PowerPC CPUs is the additional PSDVAL# (partial data valid) signal. This signal is used by the 8260 when it accesses a non 64-bit wide device. For example, when it's memory controller performs a 64-bit access to a 16-bit device, it asserts PSDVAL# with each 16-bit data transfer and asserts both PSDVAL# and TA# with the fourth data.

The MV64360/1/2 always acts as a 64-bit device when interfacing the 8260. Therefore never drives nor samples the PSDVAL# signal. When interfacing with the 8260, PSDVAL# should be pulled up to VDD.

When the MPC8260 is configured to work with its own memory controller (in addition to the MV64360/1/2), there are effectively two slaves on the bus - the MV64360/1/2 and the QUICC itself. In this case, configure MV64360/1/2 to multi-MV mode, see 9.14 "PowerPC Multi-MV Mode" on page 120. If the QUICC initiated transaction is targeted to the MV64360/1/2, the MV64360/1/2 responds with AACK# and completes the transaction. If the transaction is targeted to its own memory controller, the QUICC completes the transaction. The QUICC acts as master and slave.

The MPC8260 supports up to two pipelined transactions. The MV64360/1/2 supports any combinations of pipelined transactions. These includes both:

- Transactions targeted to the QUICC.
- Transactions targeted to the MV64360/1/2.
- One transaction targeted to the QUICC and one to the MV64360/1/2.

Note

Although the MV64360/1/2 never asserts TA# before AACK# when it acts as the target of a transaction, it tolerates this MPC8260 behavior.

9.10 PowerPC MPX Bus Support

Note

All references in this section to MPC74xx CPU applies to all MPC74xx CPUs.

Motorola MPC74xx CPUs support an enhanced bus protocol, named MPX. The MPX bus protocol includes the following enhanced bus features:

- Address streaming: No dead cycle between consecutive address tenures.
- Data Streaming: No dead cycles between consecutive data phases.
- Out of Order Data Phases: Out of order execution for data phases of multiple outstanding transactions (not in the same order of the address phases).
- Data Intervention: Cache to cache data transfers using data only transactions. This is in contrast to the conventional method of ARTRY# and cache line push back to main memory.
- 16 Bytes Burst Read/Write - G4 AltiVec load/store.

Note

Unlike the GT–64260 that is limited to a single CPU running in MPX mode, the MV64360 and MV64361 supports dual CPU configuration. There is no support for this in the MV64362

9.10.1 MPX Bus Signaling

The MPX bus interface contains the following additional signals:

Table 33: MPX Bus Additional Signals

Signal	Description
HIT0#	Snoop Hit Indicates to the MV64360/1/2 that: • The snooped address hit an exclusive or modified line in the CPU cache. • The CPU supplies the data later, using a data only transaction.
HIT1#	Snoop Hit Indicates to the MV64360/1/2 that: • The snooped address hit an exclusive or modified line in the CPU cache. • The CPU will supply the data later, using a data only transaction.
DRDY0#	Data Ready Indicates that the CPU has data ready for intervention. Used as an implicit CPU0 data bus request to perform data only transactions.

Table 33: MPX Bus Additional Signals (Continued)

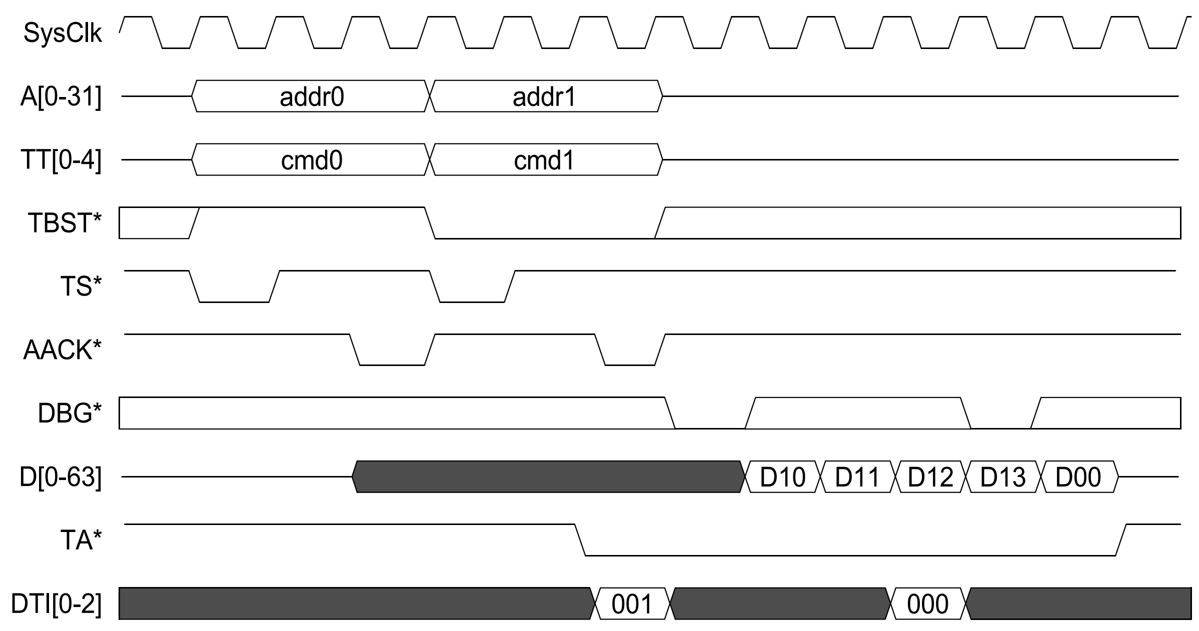
Signal	Description
DRDY1#	Data Ready Indicates that the CPU has data ready for intervention. Used as an implicit CPU1 data bus request to perform data only transactions.
DTI[0-2]	Data Transfer Index Indicates the index of the next data phase, with respect to the total number of outstanding transactions. **NOTE:** When interfacing MPC7450 that has a 4-bit DTI interface, connect MV64360/1/2 DTI[0-2] to CPU DTI[1-3]. Connect CPU DTI[0] to VSS.

9.10.2 MPX Bus Transactions

An example of pipelining two read transactions is shown in Figure 17.

In the example, there is an address streaming. The address tenure of the second transaction is driven on the cycle following the AACK# assertion for the first transaction. Since the second read's data is ready first, DTI is driven to 0x1 one cycle before DBG#. This indicates to the CPU that the next data tenure corresponds to the second transaction, rather than the first one (out of order read completion). By the time the second read's last data beat is driven on the bus, the first read's data is ready and driven on the bus with no wait states to the previous data (data streaming).

Figure 17: MPX Bus Read Example



Note

Figure 17 only demonstrates MPX bus read transactions. It does not reflect the actual read latency of the MV64360/1/2.

9.10.3 DTI[0-2]

DTI[0-2] is used as the pointer to the CPU transaction queue.

In instances of multiple outstanding transactions in which one transaction is already completed, all transactions that were queued after the completed transaction are pushed in the queue and receive new DTI values. For example, three outstanding read transactions - T1, T2, and T3- exist. T1 is coupled to DTI value of ‘000’, T1 to DTI value of ‘001’, and T2 to DTI value of ‘010’. If T2 finishes first (with DTI value of ‘001), T3 is pushed in the queue and receives a new DTI value of ‘001’. Similar to the CPU queue, the MV64360/1/2 maintains its own pending reads queue and completes them with the appropriate DTI value.

The MPX bus protocol is based on the concept that the system controller also acts as the bus arbiter. The protocol requires that DTI[0-2] must be asserted one cycle before a qualified DBG#. Since transaction ordering is determined by the system controller (based on available read data), the system controller must be the one to drive DBG#.

This bus protocol forces a read latency penalty of two cycles in comparison to the read latency in the 60x bus compatible mode. In MPX mode, only when read data arrives from the target device, the system controller knows which of the multiple, outstanding reads is about to be completed. It drives DTI[0-2] accordingly and in the next cycle it asserts DBG#. Only in the next cycle it asserts TA# and drives valid data.

The MV64360/1/2 drives a default DTI[0-2] value (without asserting DBG#), assuming the first transaction being issued is the first one completed. If this speculation succeeds, the read latency penalty is only one cycle. If it is unsuccessful, it is two cycles as previously explained.

Note

MPC7450 has four bits DTI. Connect MV64360/1/2 DTI[0-2] to the processor DTI[1-3] inputs, respectively, and connect the processor DTI[0] to VSS.

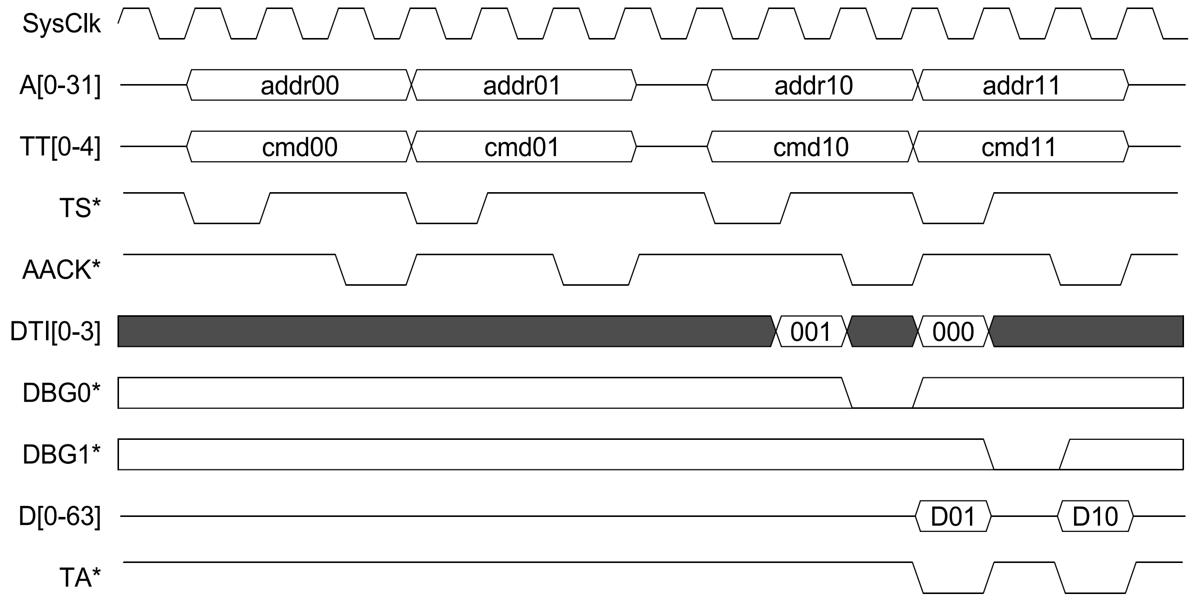
9.10.4 Multi-CPU Support and Data Intervention

As explained in section 9.10.3, the MV64360/1/2 CPU interface maintains an ordering queue. The ordering queue controls the data phases being driven on the bus and their corresponding DTIs. When interfacing with two CPUs, the MV64360/1/2 must maintain a separate queue for each CPU. Now, data phases can run on the bus in out of order data completion and the data phases of the two CPUs can mix with one another.

Each CPU has its own queue, with its own DTIs. However, there is only one DTI[0-2] output from the MV64360/1/2 to both CPUs. The CPU qualifies its DTI input with DBG#. Since each CPU has its own DBG#, a CPU cannot encounter a DTI that belongs to the other CPU.

Figure 18 shows an example of multiple outstanding reads from two CPUs. In this example, CPU0 issues two read transactions and then CPU1 issues two reads. The MV64360/1/2 CPU interface pipes all four reads to the targeted interfaces.When the first read data is available, the MV64360/1/2 drives the data on the bus. In this example, the first available data belongs to the second transaction of CPU0. Therefore, theMV64360/1/2 drives DTI value of 0x1. This is followed by DBG0# in the next cycle. And, followed by the data in the next cycle. The next available read data belongs to the first transaction of CPU1. Therefore, the MV64360/1/2 drives DTI value of 0x0. This is followed by DBG1# in the next cycle. And, followed by the data in the next cycle.

Figure 18: Multi CPU MPX Mode Read Transactions Example



Note

Figure 18 only demonstrates multi-CPU reads in the MPX bus mode. It does not reflect the actual read latency of the MV64360/1/2.

The MPX bus also supports a new concept of data intervention. Data intervention is a fast cache-to-cache data transfer that increases system performance in a multi-CPU environment.

In the conventional 60x bus cache coherency protocol, the following data fetching pattern applies. One CPU attempts to read a cache line that is present in the cache of a second CPU and is modified. Only after the second CPU has pushed the cache line back to main memory in the window of opportunity, the first CPU can fetch the data from main memory. Even if the data is not modified in the second CPU cache, the first CPU must fetch the data from main memory. This data fetching pattern results in a long read latency cycle.

When CPU data intervention is enabled, an alternative data fetching pattern takes place. When one CPU attempts to read a cache line that is present in the cache of a second CPU, it receives the data directly from the second CPU cache. The data is not pushed back to main memory by the second CPU, as in the conventional 60x bus cache coherency protocol. Directly fetching the data from the CPU results in much lower latency.

Since the data must also be updated in the main memory theMV64360/1/2 performs an action called "data snarf-ing". This means the MV64360/1/2 samples the data being transferred between the two CPUs and pushes it back to memory.

Data intervention involves two new signals - HIT# and DRDY#. These signals are CPU outputs to the MV64360/1/2 (point-to-point signals). When the CPU attempts to read a cache line that is present in the cache of another CPU, the second CPU may assert HIT# in the ARTRY# window, instead of the traditional ARTRY#. Asserting HIT# indicates that the second CPU will perform data intervention and that it intends to issue later a Data Only transaction, to drive the cache line to the requesting CPU.

Note

The requesting CPU has no knowledge that data intervention is about to occur. Only the system controller is aware of the event. It appears to the requesting CPU as a normal cache line fill.

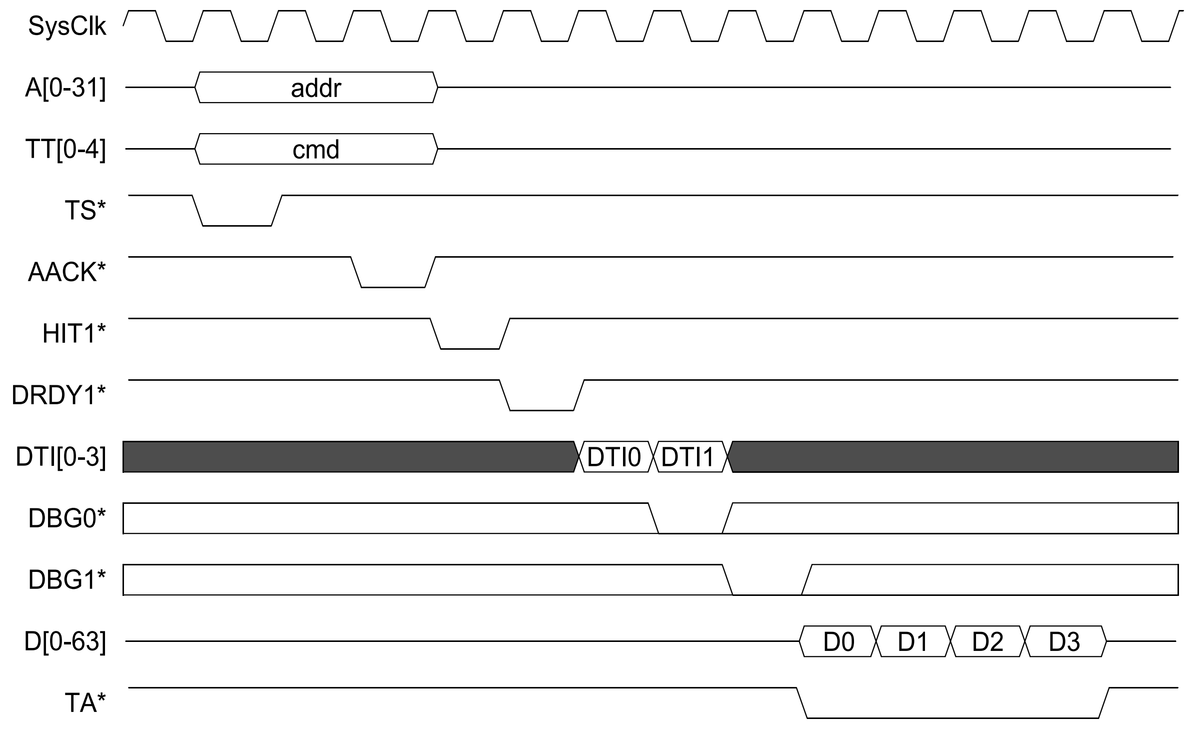
When the second CPU has the cache line data available, it asserts DRDY#. This indicates the second CPU is going to push the cache line to the requesting CPU. DRDY# is used by the MV64360/1/2 as an implicit data bus request. At the earliest window, the MV64360/1/2 grants the data bus to both CPUs by asserting both DBG0# and DBG1#. This duplicate data bus mastership is required because this is a cache-to-cache transaction. The requesting CPU expects data bus mastership to complete its cache line read data phase. Meanwhile, the second CPU expects data bus mastership so it can drive the data intervention (which is a type of “write back” transaction).

Note

Since the requesting CPU is unaware of the data intervention and is expecting to receive the cache line data in linear wrap around ordering (as in any cache line fill), the second CPU must drive the data intervention in linear wrap around order, with critical word first.

An example of data intervention between two CPUs is shown in Figure 19.

Figure 19: Data Intervention Example



In this example, CPU0 issues a cache line read request. Two cycles after TS# assertion in the ARTRY# window CPU1 asserts HIT1#. This indicates CPU1’s intent to perform data intervention. Later, CPU1 asserts DRDY1# to signal the MV64360/1/2 that it will generate a Data Only transaction and transfer the cache line to CPU0. Since each CPU has a pipe of pending transactions, each CPU needs to get the appropriate DTI value for the pending data only transaction data phase. The MV64360/1/2 drives a DTI value to each CPU followed by DBG# assertion.

Note

The MV64360/1/2 can drive a different DTI value to each CPU.

Once both CPUs are granted, the MV64360/1/2 asserts TA# for four cycles. This allows for the transfer of the cache line between the two CPUs. The MV64360/1/2 samples the data being transferred from one CPU to the other for every TA# signal it asserts. This data is pushed back to main memory, as any other write back transaction with one exception – the burst order is linear wrap around (as in the case of cache line fill).

Note

Data Intervention concept, is based on the fact that WIMG bits of the two processor are set properly (see PowerPC Architecture spec). This means that a cache lines shared between two processors has the same setting. If the cache lines do not have the same setting (programing error), a CPU read request of a single data might result in a full cache line data intervention, which might result in system hang.

If the data being pushed from one CPU to the other is in Shared or Exclusive state, a HIT signal is kept asserted for a second cycle after the ARTRY# window. This indicates that the data does not need to be pushed to main memory. If the data is in Modified state, it must be forwarded to main memory (snarfed) because it is placed in the cache of the requesting CPU in an Exclusive state and needs to be coherent with main memory.

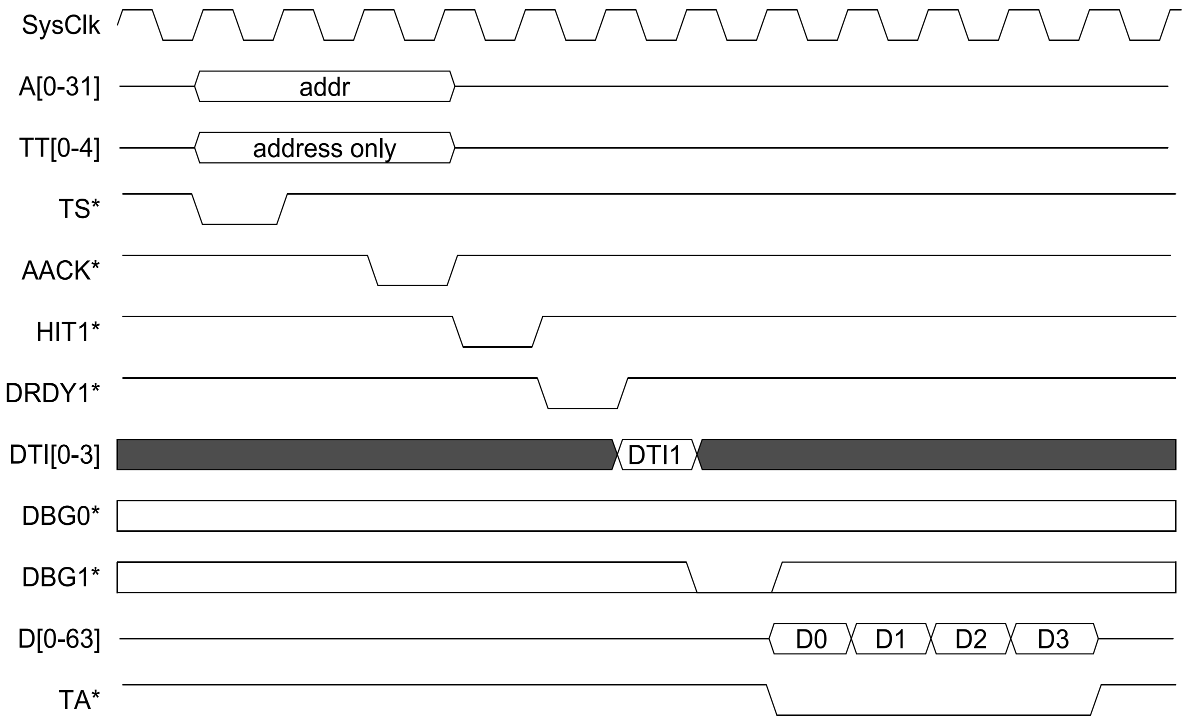
Note

The MPX bus protocol distinguishes between shared, exclusive, and modified cache line intervention. The MV64360/1/2 does not. The MV64360/1/2 always perform data snarfing during cache to cache transfer, even if the data is shared or exclusive.

A CPU is allowed to have multiple pending data only transactions. The MPX bus protocol restricts all data only transactions of the same CPU to be held in order.

A HIT# response does not necessarily means a future cache to cache transfer. A CPU might response with HIT# to an address only transaction. Figure 20 shows an example.

Figure 20: Address Only Transaction



In this example, CPU0 issues a FLUSH transaction. This forces the cache line to be flushed back to main memory. The address matches a Modified cache line in CPU1. Therefore, CPU1 responds with HIT# in the ARTRY# window. CPU1 then issues the data only transaction and pushes the cache line back to main memory. This behavior is much more efficient than the conventional ARTRY# mechanism.

Note

The Hit# signal response to an address only transaction also applies to address only transactions issued by the MV64360/1/2 (snoop transactions). For more details, see Section 20. "PowerPC Cache Coherency" on page 322.

9.11 Multi-CPU and Symmetric Multi Processing (SMP) Support

The MV64360/1/2 includes special features to support multi-CPU configurations. These features are summarized below.

Note

Most of the features rely on the usage of the MV64360/1/2 internal bus arbiter. An internal bus arbiter is required because the MV64360/1/2 needs it to distinguish between a CPU0 and a CPU1 access. If using an external bus arbiter in 60x bus mode, there is no way for the MV64360/1/2 to distinguish if the current CPU access is driven by CPU0 or CPU1.

9.11.1 Bus Features

As already mentioned, the MV64360 and MV64361 support dual CPUs on its MPX bus. It supports a pipe depth of eight transactions per CPU (total pipe depth of 16 transactions). See9.10 "PowerPC MPX Bus Support" on page 109 for full details.

Note

There is no support for dual CPUs in the MV64362.

The internal arbiter supports a dual CPUs reset sequence. After reset, CPU1 arbitration is disabled. This means CPU0 boots first and initializes the system. After the system is initialized, CPU0 enables CPU1 arbitration. See 9.13 "PowerPC Bus Arbitration" on page 119for full details.

After reset, CPU1 arbitration is disabled. This means CPU0 boots first and initializes the system. After the system is initialized, CPU0 enables CPU1 arbitration. See for full details.

The MV64360/1/2 supports the traditional ARTRY# method of cache coherency, as well as data intervention. With data intervention, cache lines are transferred directly from one CPU cache to the other CPU, making the cache line fill latency smaller. More over, the CPU can respond to a MV64360/1/2 snoop transaction with an intervention, rather than ARTRY#. This results in better MPX bus utilization.

9.11.2 Interrupts

The MV64360/1/2 has two interrupt pins per the two CPUs. Each interrupt has its own mask register. For full details on the MV64360/1/2 Interrupt Controller, see Section 25. "Interrupt Controller" on page 340.

The MV64360/1/2 also supports external interrupts driven from external resources, via it's GPP port. It supports up to 32 external interrupts. It has two GPP Mask registers, one per each CPU. The separate Mask registers, enables each CPU to deal with it's own set of GPP interrupts. For more details on the GPP interrupts, see Section 24. "General Purpose Port" on page 338.

9.11.3 Doorbell Interrupts

The MV64360/1/2 has two 8-bit wide Doorbell Interrupt registers, one per each CPU. The Doorbell Interrupt register can be used for CPU to CPU interrupt generation, for external PCI device to CPU interrupt generation, or even for CPU to interrupt itself.

To set the interrupt, write a value of '1' to a Doorbell Interrupt register bit. A '0' value has no affect. This scheme enables distributing the eight doorbell interrupts between multiple interrupt requesters, without the interrupts interfering with one another.

There is a separate pair of Doorbell Interrupt Clear registers, one per each CPU, that are used by the CPUs to clear doorbell interrupts. A CPU writing a value of '1' to clear the register bit, clears the corresponding interrupt. A value of '0 has no affect.

Note

For further information, see A.5 "SMP Registers" on page 472.

9.11.4 Who Am I register

The MV64360/1/2 includes a “Who Am I” register. This is a read only register, that may help the software running on the CPU, to identify itself.

CPU0 read from this register results in a value of 0x0, CPU1 read results in a value of 0x1, and external PCI agent read results in a value of 0x2.

9.11.5 Semaphores

The MV64360/1/2 supports eight semaphore registers. These registers can be used as a lock mechanism between the two CPUs, or even between CPU and external PCI agent.

Each possible owner has it’s own ID:

- CPU0 ID = 0.
- CPU1 ID = 1.
- External agent ID = 2.

After wake up, all eight semaphores are unlocked.

The first CPU or PCI agent to read a semaphore register, receives it’s own ID. This ID indicates that interface owns this semaphore. Once locked by one of the three (CPU0, CPU1, PCI) possible owners, a read of the semaphore register by any of the other two, will return the owner’s ID. For example, if Semaphore0 is owned by CPU1, a CPU0 read will result in a value of 0x1, indicating to CPU0 that this semaphore is currently owned by CPU1.

To unlock a semaphore, the owner writes back a value of 0xFF.

9.11.6 Sync Barrier

The MV64360/1/2 supports a CPU Sync Barrier mechanism that enables handshaking between PCI agents pushing data to the MV64360/1/2 DRAM or integrated SRAM and the CPU reading this data. This Sync Barrier mechanism allows the CPU software to confirm that the data received from a PCI has been placed in memory.

In the MV64360 and MV64361 devices, the Sync Barrier implementation enables dual CPU configuration, where both CPUs handle data received from a PCI. Each CPU can trigger its own sync barrier operation. This implementation guarantees that the two sync barrier actions do not interfere with one another.

Note

For full details on CPU sync barrier implementation, see 9.17 “CPU Synchronization Barrier” on page 123.

9.11.7 Boot Up in Multi-CPU Configuration

In a multi-CPU configuration, each CPU has it’s own boot code placed in a different memory location. Assuming that CPU0 boot code is placed in BootCS# and CPU1 code is placed in DevCS[3]#, the following boot up sequence is recommended:

1. Both CPUs assert their bus request signals. Initially, BR1# is masked and the MV64360/1/2 gives bus mastership only to CPU0.
2. CPU0 reads the boot code from BootCS#. The BootCS# default address space is 0xFFC0.0000 up to ffff.ffff. PowerPC boot vector 0xFFF0.0100 resides within this range.
3. CPU0 initializes the MV64360/1/2 internal registers, and specifically the MV64360/1/2 address map.
4. At some point, CPU0 stops executing from the BootCS# and starts executing code from DRAM.

5. CPU0 disables the BootCS# address window. Then, CPU0 re-allocates the DevCS[3]# window to match CPU1 boot vector.
6. CPU0 clears MaskBR1 bit. This enables CPU1 arbitration.
7. CPU1 is now able to read it's boot code from DevCS[3]#.

Note

PowerPC boot vector can be configured to 0xFFF0.0100 or 0x0000.0100. The MV64360/1/2 BootCS# default map matches boot vector 0xFFF0.0100. However, for CPU1, boot vector 0x0000.0100 is also useful.

9.12 CPU Bus Mastering

In addition to the CPU interface snoop capability, the MV64360/1/2 supports master capability for data transfers between the MV64360/1/2 and other agents on the 60x bus, such as another MV64360/1/2 device or PowerQUICC II local memory.

Access to the CPU bus is established through the different MV64360/1/2 units address decoding mechanisms, see 7.1 "CPU Address Decoding" on page 82. If a CPU bus access is required, the MV64360/1/2 initiates a CPU-like transaction. Transaction attributes are determined according to the required amount of data.

When configured to interface with the PowerPC CPU, the MV64360/1/2 is limited to cache line bursts as defined by the 60x bus protocol. In this case, any PCI, IDMA, MPSC, or Ethernet controller access to the CPU bus, which is not fully cache line aligned, results in the CPU interface splitting the burst to a single beat bus transaction.

Note

The MV64360/1/2 data mastering on the CPU bus is not supported when running in MPX mode.

9.12.1 CPU Master Interface Operation

The CPU master interface consists of 512 bytes of posted write data buffer and 512 bytes of read data buffer.

The Write buffer can absorb up to four 128-bytes transactions. The bus master interface tries to access the CPU bus as soon as the first write data is placed in write buffer or only when the whole burst is placed in the write buffer, depending on the setting of the CPU Master Control register's `MWrTrig` bit [10] (Table 232 on page 462). If the bus is occupied and transaction cannot be driven, the write buffer can still absorb new DMA/PCI write transactions.

The Read buffer can absorb up to four 128-byte transactions. The bus master interface drives the read data to the initiating unit as soon as data arrives from the CPU bus or only when the whole burst read is placed in the read buffer. This depends on the setting of `MRdTrig` bit [11] in the CPU Master Control register.

Note

If using the MV64360/1/2 integrated SRAM, `MRdTrig` must be set to '1'.

To comply with the 60x bus specification, the only burst transactions the master interface can generate are cache line bursts. Therefore, the master splits a burst request received from one of the MV64360/1/2 units to multiple 60x bus transactions, according to the cache line alignment. If, for example, the master interface is requested for a read of 64 bytes from address offset 0x8. The master generates three 8-bytes read transactions from offsets 0x8, 0x10, and 0x18. Then, it generates a cache line read (32 bytes) from offset 0x20. This is followed by an 8-byte read from offset 0x40.

The Bus master interface can pipeline up to four transactions on the CPU bus. It supports only in-order completion (as required by 60x bus protocol). Transactions are issued to the CPU bus in order.

The CPU master interface also generates snoop transactions for PowerPC cache coherency support. During snoop transactions, write and read data buffers are not used because snoops are generated via address only transactions.

9.13 PowerPC Bus Arbitration

PowerPC bus protocol supports separate arbitration on address and data busses.

The MV64360/1/2 supports both external arbiter or internal arbiter configurations. Setting CPU Master Control register's `IntArb` bit [8] to '1' enables the MV64360/1/2 internal CPU bus arbiter.

Note

Using an external arbiter is only relevant to the 60x bus mode. When configured to the MPX bus mode, bus arbitration is maintained by the MV64360/1/2 MPX bus arbiter, regardless of `IntArb` bit setting.

9.13.1 MV64360/1/2 Internal 60x Bus Arbiter

MV64360/1/2 60x bus arbiter supports arbitration of two external bus masters (CPUs) plus internal bus requests (snoop requests or data transactions).

If the internal arbiter is enabled, the BR0# signal is used as the CPU bus request input and BG0# and DBG0# are used as the CPU bus grant and data bus grant outputs. In this configuration, the arbiter bus requests are BR0#, BR1#, and the CPU interface internal request (for bus mastering or snooping). The arbiter bus outputs are BG0# and DBG0# to one master, BG1# and DBG1# to the second bus master, and internal bus grants and data bus grants for the internal CPU interface.

Notes

- In the MV64360 and MV64361, by default, the BR1# input is masked, enabling CPU0 to boot first. To enable CPU1 arbitration, set the CPU Master Control register's MaskBR1 bit to '0', see Table 110 on page 138.
- If the MV64360 and MV64361 are using a single CPU system, only the BR0*, BG0*, and DBG0* signals are used as arbitration signals. The CPU Master Control register's MaskBR1 bit must be set to '1' and the BR1* input requires a pull-up.

The arbiter works in a fixed round robin scheme, giving fair arbitration to the tree requests sources. Since 60x bus restricts in order data phases, the arbiter keeps a queue of pending data phases. The queue maintains the DBG assertions in the same order the BGs were asserted.

In MPX bus mode, arbitration ordering is mixed up with the data phases of read transactions coming back from the MV64360/1/2 interfaces. As soon as read data is available, the arbiter assigns DBG# (with the appropriate DTI value) to the CPU that is the source for the data. In case of data intervention, the arbiter assigns DBG# to both CPUs at the earliest window following DRDY# assertion.

On boot up, one CPU is typically required to initialize the system. The second CPU only starts working afterwards. The MV64360/1/2 internal arbiter supports “starvation” of one CPU until it is enabled by the master CPU. It ignores BR1# assertion and keeps BG1# de-asserted. This state is maintained until the CPU Master Interface Control register’s `MaskBR1` bit [9] is cleared (Table 237 on page 470). When the master CPU completes system initialization, it clears this bit. This allows the second CPU to start booting.

9.13.2 External Bus Arbiter

If the MV64360/1/2 60x bus arbiter is not good enough for a given system configuration (more than two external bus masters, for example), an external arbiter is required.

In this mode:

- The BG1#/GT_BR# signal is used as the MV64360/1/2 bus request output
- The BR0#/GT_BG# signal is used as the MV64360/1/2 bus grant.
- The BR1#/GT_DBG# signal is used as the MV64360/1/2 data bus grant.

9.14 PowerPC Multi-MV Mode

Notes

- Multi-MV is only supported in 60x bus mode.
- Operating in multi-MV mode affects the AC Timing. Before implementing multi-GT support, consult with your local FAE.

It is possible to connect up to four MV64360/1/2 devices to the CPU bus without the need for any glue logic. This capability adds significant flexibility for system design. Multiple MV64360/1/2 are enabled through reset configuration, see Section 27. “Reset Configuration” on page 347.

9.14.1 Hardware Connections

In multi-MV configuration, the AACK# and TA# signals function as sustained tri-state outputs requiring 4.7 KOhm pull up resistors. All TA# outputs from the MV64360/1/2 devices must be tied together to drive the CPU TA# input. All AACK# outputs from the MV64360/1/2 devices must be tied together to drive the CPU AACK# input.

AACK# and TA# are only driven by the target MV64360/1/2. After last TA# and AACK# assertion, the MV64360/1/2 drives these signals HIGH for another half cycle and then tri-states them.

In case of a bad CPU address that misses all address windows in all of the MV64360/1/2 devices, no device will assert AACK# and the system might hang. If `NoMatchCntEn` bit [8]in the CPU Configuration Register is set to ‘1’, the boot MV64360/1/2 responds after a time out period defined in the NoMatchCnt field of the CPU Configuration Register and completes the transaction.

Note

The NoMatch counter is only applicable to the boot device (the one with Multi-MV ID of ‘11). In case the boot ROM is connected to some other slave device other than MV64360/1/2 (e.g. MPC8260 boots from a local ROM), the system might hang in case of address mismatch. To avoid a system hang, the non-MV64360/1/2 slave device must have some address mismatch protection mechanism).

9.14.2 Multi-MV Mode Address Decoding

Note

The following section assumes a 36-bit CPU address (A[0] is the MSB, A[35] is the LSB).

In multi-MV mode, each MV64360/1/2 device has a two bit ID. This ID distinguish between the devices - each device responds to transaction address that matches it's ID, as shown in Multi-MV ID Encoding.

Table 34: Multi-MV ID Encoding

ID	Multi-MV64360/1/2 Address ID
00	MV64360/1/2 responds to A[9-10]='00'
01	MV64360/1/2 responds to A[9-10]='01'
10	MV64360/1/2 responds to A[9-10]='10'
11	MV64360/1/2 responds to A[9-10]='11'
	NOTE: The boot MV64360/1/2 ID must be programmed to '11'.

If the MV64360/1/2 is configured to the multi-MV mode during reset, the `MultiMVDec` bit [18] in CPU Configuration register is SET, indicating that the CPU Interface address decoding is reduced to:

1. If A[9-10] == ID AND it's a WRITE, the access is directed to the internal space of the CPU Interface registers with A[20-31] defining the specific register offset.
2. If A[9-10] == ID AND it's a READ AND A[8] == 0, the access is directed to the internal space of the CPU Interface registers with A[20-31] defining the specific register offset.
3. If A[9-10] == ID AND it's a READ AND A[8] == 1, the access is directed to BootCS#.

Notes

- Since 0xFFF0.0100 (PowerPC boot address) implies A[9-10] == 0x3, the MV64360/1/2 holding the boot device must be programed to ID = 3.
- If the CPU attempts to access an address that is outside of the address ranges defined above (even with an address-only transaction), the system may hang. Before accessing such an address, set the boot CPU Configuration register's `NoMatchCntEn` bit [8] to '1'.

4. When the `MultiMVDec` bit is CLEARED, the CPU Interface resumes normal address decoding.

9.14.3 Initializing a Multi-MV64360/1/2 System

The following procedure is recommended to initialize a system with two MV64360/1/2 devices attached to the same CPU.

Note

For this example, the two MV64360/1/2 devices are called MV-1 and MV-2, MV-1 ID is '11' (boot MV64360/1/2) and MV-2 ID is '00'. After reset, the processor executes from the BootROM on MV-1 because the address on A[0-35] is 0x0.FFF00100 where A[8-10] = '111' and it's a read cycle. Registers on MV-1 are accessible via address A[9-10]='11'. Registers on MV-2 are accessible via address A[9-10]='00'. Registers offsets are determined via A[20-31].

1. Reconfigure MV-2's CPU Interface address space registers.
2. Reconfigure MV-1's CPU Interface address space registers.

Note

The address mapping for each MV64360/1/2 must be unique. There must not be any address decoding range in one device that overlaps any part of the other device address mapping.

3. Disable the MV-2 Internal Space window (clear Internal Space Base Address register's En bit).
4. Clear MV-2 `MultiMVDec` bit.
5. Clear MV-1 `MultiMVDec` bit.
6. Enable MV-2 Internal Space window.

Both MV64360/1/2 devices resume NORMAL operation with USUAL address decoding.

Note

In the presence of multiple MV64360/1/2 devices, the CPU Configuration Register for each one of them must be programed to the same value.

9.15 PowerPC Parity Support

The MV64360/1/2 supports odd data parity driven on DP[0-7] and odd address parity driven on AP[0-4].

The MV64360/1/2 samples address parity with each CPU transaction and drives parity as CPU bus master. It samples data parity on write transactions and drives parity on reads. It also propagates bad parity between the CPU bus and the other interfaces (SDRAM, PCI). In case of bad parity detection, it also asserts an interrupt.

Note

IBM 750Cx does not support address and data parity. When interfacing this CPU, CPU Configuration register's bits APVal and DPVal bits must be set to '0'.

For full description of parity support, see 19.1 "CPU Parity Support" on page 313.

9.16 Big/Little Endian Support

The MV64360/1/2 supports only Big Endian CPU bus. The MV64360/1/2 provides the capability to swap the byte order of data that enables endianess conversion between the CPU interface and some other interfaces.

The endianess convention of the local memory attached to the MV64360/1/2 (SDRAM, integrated SRAM devices) is assumed to be the same one as the CPU. This means data transferred to/from the local memory is NEVER swapped.

The internal registers of the MV64360/1/2 are always programed in Little Endian. On a CPU access to the internal registers,data is byte swapped.

For example, a CPU write of data 0xaa to an internal register at offset 0x2. The data appears on the {DH,DL} bus as - 0x????aa?????????? and byte swap is required.

Note

The location of the data on the CPU bus depends on the size of the access. For example, a CPU write of 32-bit word 0xaabbccdd to offset 0x0 appears on the bus as 0xaabbccdd???????? where byte 0xdd is the least significant byte. However, a write of a single byte 0xdd to offset 0x0 appears on the bus as 0xdd??????????????. The MV64360/1/2 byte swap on access to the internal registers is based on a byte access convention. On CPU half-word or word access, software needs to perform further byte swapping.

Data swapping on a CPU access to the PCI is controlled via PCISwap bits of each PCI Low Address register. This configuration setting allows a CPU access to PCI agents with a different endianess convention.

For software compatibility with the GT-64120/130 devices, the MV64360/1/2 maintains MByteSwap and MWordSwap bits in the PCI Command register. If the PCI Command register's `MSwapEn` bit is set to '1' (Table 378 on page 535), the MV64360/1/2 PCI master performs data swapping according to PCISwap bits setting. If set to '0' (default), it works according to MByteSwap and MWordSwap bits setting, as in the GT-64120/130 devices.

Note

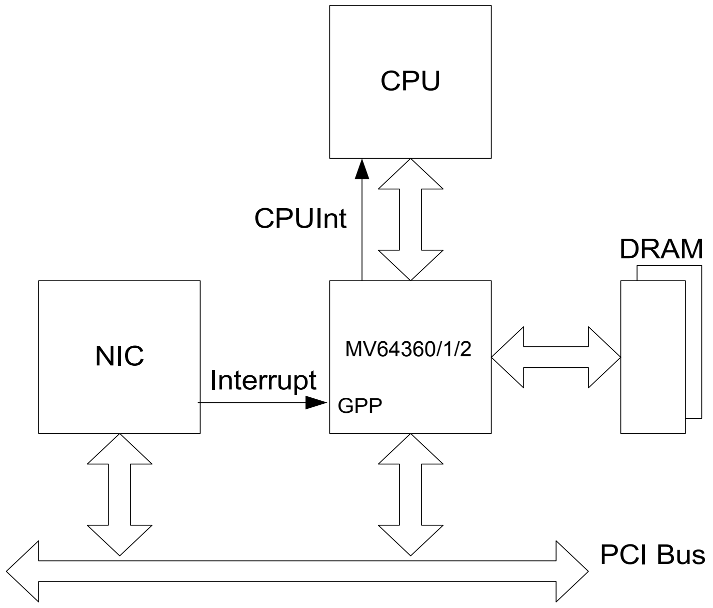
See 13.10 "Data Endianess" on page 172 for more information on data swapping.

9.17 CPU Synchronization Barrier

The MV64360/1/2 supports a sync barrier mechanism. This mechanism is a hardware hook to help software synchronize between the CPU and PCI activities. The MV64360/1/2 supports sync barrier in both directions - CPU to PCI and PCI to CPU.

Figure 21 shows an example of a CPU sync barrier application.

Figure 21: CPU Sync Barrier Example



Assume the NIC sends a packet over the PCI to the DRAM and then interrupts the CPU (using the MV64360/1/2 GPP interrupt). Since the packet might still reside in the MV64360/1/2 PCI slave write buffer rather than DRAM, the CPU interrupt handler must first perform a sync barrier action, before reading the packet, to make sure the packet is flushed to DRAM.

Additionally, if the packet goes to cache coherent region, the transaction might be placed in the snoop queue, waiting for snoop to be resolved (see Section 20. “PowerPC Cache Coherency” on page 322 for more information on cache coherency implementation). The sync barrier must guarantee that the snoop completes, before the CPU access the DRAM and reads the packet.

Notes

- The MV64360/1/2 CPU sync barrier implementation is different than the GT-6424x/6x implementation. It is a register polling based, rather than a single read waiting for resolution.
- If the packets and/or descriptors are placed in cache coherent regions, sync barrier is required also in case of handshake between the CPU and the MV64360/1/2 GbE ports. Once the CPU is interrupted by the Ethernet controller, the software must make sure that the transaction is not placed in snoop queue, before it accesses memory and reads the data.

The MV64360/1/2 implements two sets of sync barrier register. Each set contains a CPU Sync Barrier Trigger register and a CPU Sync Barrier Virtual register (A.6 “CPU Sync Barrier Registers” on page 477). The two sets can be used by two CPUs (one per each CPU) in multi-CPU configuration. For the CPU to activate a sync barrier, it must first write to the Trigger register and then perform read polling on the virtual register.

The trigger register is 4-bits wide. Each bit defines which buffers must be flushed. If for example, the sync barrier is used to flush the write path from PCI_0 to DRAM, the PCI_0 bit must be set. If it goes to a DRAM cache coherent region, the DRAM bit must be set. Another example, if the GbE ports descriptors are located in the integrated SRAM, and the SRAM is marked as cache coherent, the SRAM bit needs to be set, to flush the path from GbE port to the SRAM.

A write to the CPU Sync Barrier Trigger register, triggers the sync barrier state machine. A subsequent read polling on the CPU Sync Barrier Virtual register results in value of 0xFFFF.FFFF, until the sync action completes. Once the buffers are flushed, a read of the Virtual register results in a value of 0x0.

The MV64360/1/2 also supports nesting of sync barriers. For example, there are two NIC cards set on the two PCI interfaces. Each card has a dedicated GPP interrupt. The NIC on PCI_0 interrupts the CPU first, indicating to the CPU that there is a pending buffer to handle in DRAM. CPU interrupt handler, trigger in response, a sync barrier to flush the path from PCI_0 to DRAM. Then it starts polling on the CPU Sync Barrier Virtual register. However, before sync barrier is resolved, the NIC on PCI_1 also interrupts the CPU. If the interrupt handler, decides that this is a higher priority task, it can re-trigger the sync barrier, setting the PCI_1 bit in the CPU Sync Barrier Trigger register. The MV64360/1/2 sync barrier implementation guarantees that a subsequent read polling on the CPU Sync Barrier Virtual register will result in a value of 0x0, only after the path from PCI_1 to DRAM is flushed.

In multi-CPU configuration, where every CPU manages it's own traffic from PCI (or GbE ports) to DRAM or integrated SRAM, each CPU handles its own sync barrier activity using its own set of sync barrier registers, without interfering with the activity of the peer CPU. This implementation guarantees, that the two sync barriers do not mix with each other, and that the CPU sync barrier is marked as resolved (read of the Virtual register results in 0x0), only when its specific path is flushed.

9.18 CPU Interface Clocking

The CPU interface is driven from the core clock (SysClk). All CPU interface inputs are sampled on the rising edge of SysClk. All output are driven with the rising edge of SysClk.

Note

Unlike the GT-6424x/6x, the MV64360/1/2 no longer supports a separate slower CPU interface clock.

9.19 CPU Interface I/O Signaling

For improved signal integrity and board timing design, the CPU interface pads have a calibration mechanism to control pad drive and impedance. Connect the CPU_CAL pin to V_{DD} CPU via a resistor. The resistor size must be in the range of 50 ohms. After reset de-assertion, the calibration logic tunes the CPU interface output drivers.

9.20 Programing the CPU Configuration Register

Note

The CPU interface registers are located in Appendix A. "CPU Interface Registers" on page 438.

The CPU setting of the CPU Configuration register requires special care, since it affects the MV64360/1/2 behavior on consecutive CPU accesses.

To change the register, the following steps are recommended:

1. Read the CPU Configuration register. This guarantees that all previous transactions in the CPU interface pipe are flushed.
2. Program the register to its new value.

3. Read polling of the register until the new data is being read.

Note

The CPU Configuration register wakes up with transactions pipeline disable. It is recommended to change this default in order gain the maximum CPU interface performance.

Setting the CPU Configuration register must be done once. For example, if the CPU interface is configured to support out of order read completion, changing the register to not support OOO read completion is fatal.

Section 10. Integrated SRAM

Note

This section only applies to the MV64360 and MV64361 devices. The MV64362 does not include integrated SRAM.

The MV64360 and MV64361devices integrate a 2 Mb SRAM.

The SRAM is a general purpose memory and accessible from any of the device units. Since it is an internal resource, it's typical access time is much faster than access to external resources, such as DRAM. Moreover, to achieve low CPU read latency, the SRAM is integrated as part of the CPU interface unit. This results in a CPU read latency as fast as six cycles.

The integrated SRAM may be used for many applications. It is especially useful when the DDR SDRAM bandwidth is critical and the SRAM can off load some of the DRAM traffic or as a descriptor memory, allowing DMAs to simultaneously run the data from DRAM and descriptors from the integrated SRAM.

The integrated SRAM also supports parity generation and checking. It also supports cache coherency per cache coherent regions basis, see Section 20. “PowerPC Cache Coherency” on page 322.

Note

Following reset de-assertion, do not access the integrated SRAM for 450 cycles.

10.1 Implementation

The SRAM supports accesses from 1byte up to bursts of 128 bytes. It runs at the SysClk domain (up to 133MHz), resulting in throughput of up to 1 GBps.

The SRAM is accessible through the regular MV64360 and MV64361 address decoding logic. The CPU and PCI interfaces have a dedicated fixed size (256 KB) address windows per the integrated SRAM. All other units can assign one of their address decoding windows to be targeted to the integrated SRAM.

The SRAM controller contains a transaction queue, read and write buffers (128 bytes each). The transaction queue guarantees in order execution of the incoming transactions. This maintains data coherency between consecutive accesses to the same location.

The SRAM controller contains a fixed round robin arbiter, to arbitrate between simultaneous requests from CPU and some other unit.

Note

Arbitration between different units requesting access to the SRAM banks is performed in the MV64360 and MV64361crossbar, see Section 8. “Internal Crossbar” on page 95.

10.1.1 Parity Support

During a write access, the SRAM controller calculates even parity for the incoming data, and update the appropriate parity bits along with the data.

During a read access, the SRAM controller calculates even parity for the data being read and compares it to the parity being read from the SRAM. If parity is enabled (the SRAM Configuration register's `ParEn` bit [4] is set to '1') and the calculated parity does not match the parity being read from the SRAM, the SRAM controller set's the SRAM cause bit in the Main Interrupt Cause register and the interrupt is set, if not masked.

The integrated SRAM also supports propagation of parity errors. If the SRAM Configuration register's `PErrProp` bit [5] is set to '1' (Table 276 on page 484) and a write transaction to SRAM is marked with erroneous data (meaning, errors were detected in the originating interface), the SRAM controller forces a bad parity in the SRAM.

The SRAM controller also supports forcing of specific parity byte during write access for system debug purposes. If the SRAM Configuration register's `ForceParEn` bit [6] is set to '1', the SRAM controller, instead of writing a correct even parity to SRAM, writes a fixed parity byte as defined in SRAM Configuration register's ForcePar bits [15:8].

10.1.2 Cache Coherency

The MV64360 and MV64361 also supports cache coherency between the integrated SRAM and CPU caches. If the SRAM Configuration register's `CCEn` bits [1:0] are not '0' (see Table 276 on page 484), access from any of the units to the SRAM results in a snoop on the CPU bus. If set to 0x2 (WB mode), the access to SRAM is delayed and, only after the snoop is resolved, is the access driven to the SRAM.

Note

For more details, see Section 20. "PowerPC Cache Coherency" on page 322.

Section 11. DDR SDRAM Controller

The DDR SDRAM (Double Data Rate-Synchronous DRAM) controller supports up to four DRAM banks (four DRAM chip selects). It has a 16-bit address bus (DA[13:0] and BA[1:0]) and a 72-bit data bus (DQ[63:0], CB[7:0]). It supports 64, 128, 256, 512 Mb and 1 Gb DDR SDRAM devices. Up to 2 GB address space per DRAM bank, which results in 8 GB maximum DRAM space.

Note

The 8 GB DRAM address space is only reached when using 1 Gb x4 DRAM devices.

The DRAM can be accessed from any of the MV64360/1/2 interfaces. The DRAM controller supports up to 128byte burst per a single transaction and can run up to a 183 MHz clock frequency (366MHz data rate). With it's enhanced bank interleaving mechanism, it can reach the maximum DDR bandwidth of 23.4 Gbps.

Note

The 183 MHz clock frequency is only achievable when interfacing single load DRAM (e.g. single 512 MB DRAM bank, built up of nine 512 Mb x 8 devices). With higher loads, the frequency is limited to 133 MHz.

The DRAM controller supports DDR DRAM DIMMs - both registered and unbuffered.

It is also possible to configure the DRAM controller to keep pages open. This eliminates the need to close a page (pre-charge cycle) and re-open it (activate cycle) in case of consecutive accesses to the same page. This is typically useful on long DMA bursts to/from DRAM. Up to 16 pages can be opened simultaneously.

The DRAM controller also supports ECC (Error Checking and Correction) that allows it to detect and correct single bit error and detect two bit errors. It supports RMW (Read-Modify-Write) for the case of partial writes to DRAM (write of data which is smaller than 64-bit).

The DRAM controller also supports a wide range of DRAM timing parameters to meet current and future DDR DRAM Characteristics.

11.1 SDRAM Controller Implementation

The DRAM controller receives read and write requests from any of the other interfaces through the MV64360/1/2 crossbar (see section Section 8. "Internal Crossbar" on page 95) and translates these requests to DDR SDRAM transactions.

The DRAM controller contains four transaction queues – two write buffers and two read buffers. It can absorb up to 16 write and 16 read transactions with up to 128 bytes per transaction.

Once a DRAM access is requested, the address is pushed into one of the four transaction queues. The DRAM controller arbitrates between the four transactions queues. It takes the transaction from the top of the selected queue and drives part of the address bits on DA[13:0] and BA[1:0] during the activate cycle (RAS#) and the remaining bits during the command cycle (CAS#).

Note

The DRAM controller does not necessarily issue DRAM transactions in the same order that it receives the transactions. See 11.1.1 "Arbitration" on page 131 for more information on the queues arbitration scheme and DRAM interleaving.

For a write transaction, write data coming from the requesting unit is placed in the write buffer. The DRAM write buffer allows the requesting unit to complete a write transaction, even if the DRAM controller is currently busy in serving a previous transaction.

For a read transaction, after the command cycle (CAS#), the SDRAM controller samples read data driven by the DRAM (sample window depends on CL parameter), pushes the data into the read buffer, and drives it back to the requesting unit.

The DRAM controller attempts to drive read data back to the requesting unit, as soon as data is available, to gain minimum read latency. However, it doesn't always succeed.

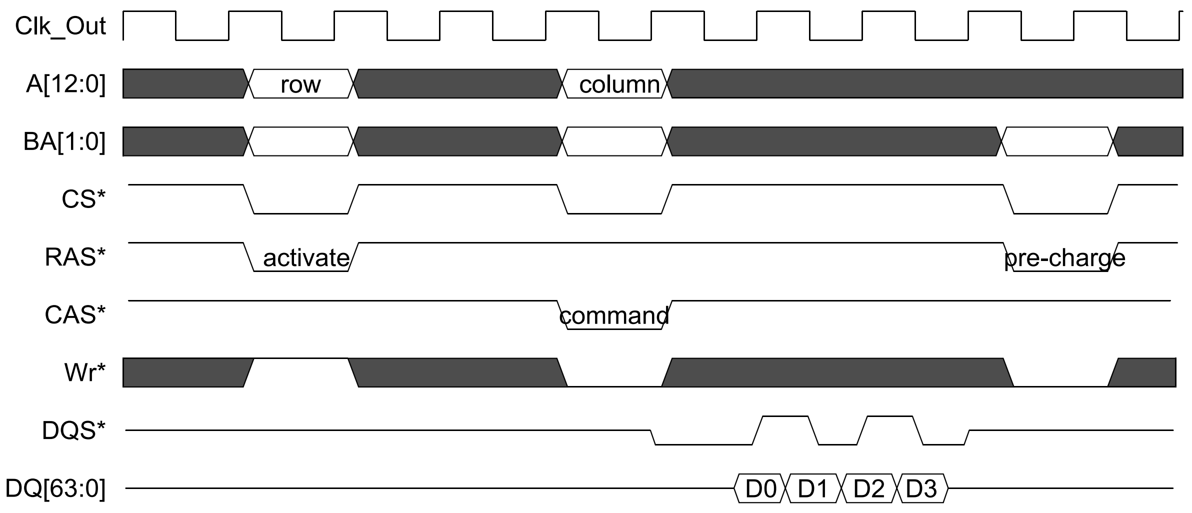
For example, if the CPU interface issues a read from the PCI and then issues another read from DRAM, by the time the DRAM controller is able to return read data, the CPU interface unit might not be able to absorb the data. The CPU interface is busy in receiving read data from the PCI. In this case, read data from DRAM is placed in the read buffer and only pushed to the CPU interface unit when it is ready to receive the data.

The two DRAM controller read buffers are also used for de-coupling reads to different resources. Via the DRAM Configuration register's `RdBuff` bits [31:26] (Table 287 on page 489), each requesting interface (CPU, PCI, IDMA, MPSCs, Ethernet) can be assigned to use one of the two buffers. For example, if the CPU read latency is important and shouldn't be delayed due to some PCI read data waiting in the top of read buffer, assigning one buffer for the CPU interface and the other buffer to the other interfaces guarantees a minimum CPU read latency.

Both read and write buffers are also used as pack/unpack fabrics. The data path from/to the requesting units is 64-bits wide. The DRAM data path is 64-bits wide running at double data rate, equivalent to 128-bit wide. Write data is pushed to the write buffer in 64-bit granularity, and pulled out of the buffer towards the DRAM in 128-bit granularity. Read data coming from DRAM is pushed into the read buffer at 128-bit granularity, and pulled out of the buffer in 64-bit.

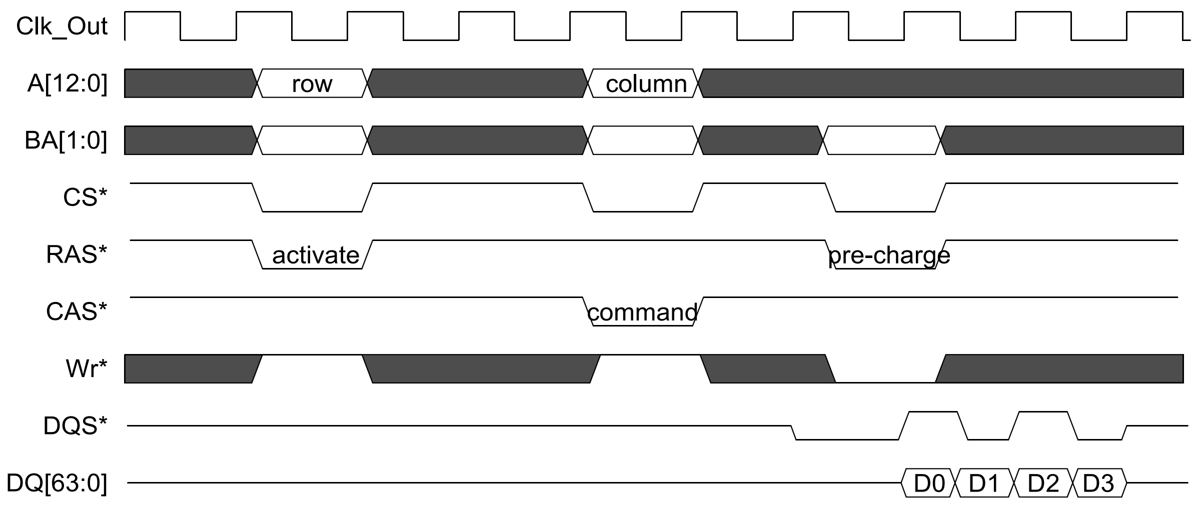
An example of write transactions are shown in Figure 22. Basic DRAM controller access to DRAM consists of an activate cycle (row address), a command (column address), and a pre-charge at the end of transaction. Write data is driven with each of the clock's rising and falling edges, along with DQS. The SDRAM controller also inserts the required preamble and post-amble.

Figure 22: DDR Burst Write Example



Example of a read transaction is shown in Figure 23. The DRAM controller latches the incoming data with each rising and falling of DQS input.

Figure 23: DDR Burst Read Example



11.1.1 Arbitration

Transactions coming from the MV64360/1/2 core are pushed into one of four transaction queues. The DRAM controller selects which queue is used to achieve maximum DRAM bank interleaving, see section 11.4.1 “SDRAM Bank Interleaving” on page 135.

The DRAM controller arbitrates between the four queues, and selects which of the queues to serve next. The arbitration is a basic round robin scheme with some priority mechanism. Each requesting unit can be assigned to high priority using DUnit Control Low register's `Prio` bits [15:10] (Table 288 on page 490). The DRAM controller serves the high priority requests first.

11.1.2 Cache Coherency

The MV64360/1/2 supports full PowerPC cache coherency between CPU L1/L2 caches and DRAM. Each access to the DRAM may result in snoop transaction initiated by the MV64360/1/2 on the CPU bus.

An access to DRAM that requires snoop action is placed in the snoop queue. The access waits for snoop resolution. Once the snoop is resolved, the transaction moves to the regular transaction queue and is issued to DRAM.

For full description of snoop process, see Section 20. "PowerPC Cache Coherency" on page 322.

11.2 DRAM Size

The MV64360/1/2 supports 64, 128, 256, and 512 Mb DDR DRAM devices as well as 1 Gb devices. The different DRAM devices differ in the usage of DA[13:0] and BA[1:0] lines, as described in Table 35.

Table 35: DRAM Addressing

DRAM Type		Bank Address	Row Address	Column Address	Auto Precharge
64 Mb	16Mx4	BA[1:0]	DA[11:0]	DA[9:0]	DA[10]
	8Mx8	BA[1:0]	DA[11:0]	DA[8:0]	DA[10]
	4Mx16	BA[1:0]	DA[11:0]	DA[7:0]	DA[10]
	2Mx32	BA[1:0]	DA[10:0]	DA[7:0]	DA[8]
128 Mb	32Mx4	BA[1:0]	DA[11:0]	DA[11], DA[9:0]	A[10]
	16Mx8	BA[1:0]	DA[11:0]	DA[9:0]	DA[10]
	8Mx16	BA[1:0]	DA[11:0]	DA[8:0]	DA[10]
	4Mx32	BA[1:0]	DA[11:0]	DA[7:0]	DA[8]
256 Mb	64Mx4	BA[1:0]	DA[12:0]	DA[11], DA[9:0]	DA[10]
	32Mx8	BA[1:0]	DA[12:0]	DA[9:0]	DA[10]
	16Mx16	BA[1:0]	DA[12:0]	DA[8:0]	DA[10]
	8Mx32	BA[1:0]	DA[12:0]	DA[7:0]	DA[8]

Table 35: DRAM Addressing (Continued)

DRAM Type		Bank Address	Row Address	Column Address	Auto Precharge
512 Mb	128Mx4	BA[1:0]	DA[12:0]	DA[12:11], DA[9:0]	DA[10]
	64Mx8	BA[1:0]	DA[12:0]	DA[11], DA[9:0]	DA[10]
	32Mx16	BA[1:0]	DA[12:0]	DA[9:0]	DA[10]
	16Mx32	BA[1:0]	DA[12:0]	DA[9], DA[7:0]	DA[8]
1 Gb	256Mx4	BA[1:0]	DA[13:0]	DA[12:11], DA[9:0]	DA[10]
	128Mx8	BA[1:0]	DA[13:0]	DA[11], DA[9:0]	DA[10]
	64Mx16	BA[1:0]	DA[13:0]	DA[9:0]	DA[10]
	32Mx32	BA[1:0]	DA[13:0]	DA[9], DA[7:0]	DA[8]

Note

Bank Address is the same in both RAS and CAS cycles.

Auto Pre-charge indication during the CAS cycle is A10 in x4,x8 and x16 devices. It is A8 in x32 devices.

The DRAM controller supports up to four DRAM banks (DRAM chip selects). The total DRAM bank address space is determined by the nature of the DDR DRAM devices. If for example using 256 Mb x8 devices (32Mx8), the bank, built up of eight such devices, has 256 MB address space.

11.3 SDRAM Timing Parameters

The SDRAM controller supports a wide range of SDRAM timing parameters. These parameters can be configured through the SDRAM Mode register (see Table 295 on page 495) and SDRAM Timing Parameters register (see Table 290 on page 493 and Table 291 on page 493):

Table 36: SDRAM Timing Parameters

SDRAM Timing Parameters	Description
CAS Latency (CL)	The number of cycles from CAS# assertion to the sampling of the first read data. The SDRAM controller supports CL of 1.5, 2, 2.5, 3, 3.5 or 4 cycles.
RAS Precharge (Trp)	The minimum number of cycles between a precharge cycle and the following activate cycle. The SDRAM controller supports Trp of 2, 3 or 4 cycles.

Table 36: SDRAM Timing Parameters

SDRAM Timing Parameters	Description
RAS# to CAS# (Trcd)	The minimum number of cycles between the assertion of RAS# with a valid row address (activate) to the assertion of CAS# with a valid column address (command). The SDRAM controller supports Trcd of 2, 3 or 4 cycles.
Row Active Time (Tras)	The minimum number of cycles between activate cycle to precharge cycle. The SDRAM controller supports Tras of 5, 6, 7, 8 or 9 cycles.
Write to DQS (Tdqss)	The minimum number of cycles between write command and DQS assertion. The SDRAM controller supports Tdqss of 1 cycle.
Write to Precharge (Twr)	The minimum number of cycles between write command and pre-charge. The SDRAM controller supports Twr of 2 or 3 cycles.
Write to Read (Twtr)	The minimum number of cycles between write command and read command. The SDRAM controller supports Twtr of 1 or 2 cycles.
Active to Active (Trrd)	The minimum number of cycles between activate bank A to activate bank B. The SDRAM controller supports Trrd of 1,2 or 3 cycles.
Refresh Command (Trfc)	The minimum number of cycles between refresh command and new activate command. The SDRAM controller supports Trfc between eight to 16 cycles.
Read to Read (Trtr)	The minimum number of cycles between consecutive read com-mands. The SDRAM controller supports Trtr of 1 or 2 cycles.
Read to Write (Trtw)	The minimum number of cycles between read command to write command. The SDRAM controller supports Trtw of 1 or 2 cycles.

11.4 DRAM Burst

A DDR SDRAM device can be configured to different burst lengths and burst ordering. The MV64360/1/2 DRAM controller supports only BL (Burst Length) setting of four. It only supports linear wrap around burst type (DRAM Mode register's BT must be set to '0').

A single DRAM access request in MV64360/1/2 can vary from a single byte up to 128byte burst (burst of 16 64-bit words). Also, the burst ordering can vary depending on the requesting unit. A CPU cache line read requires linear wrap around ordering around the 32-byte cache line boundary, while access from any other interface requires straight linear burst order. The DRAM controller drives the DRAM address and control signals at the appropriate time windows. This supports the different bursts size and ordering required by the different units.

When the required DRAM access is not a full multiple of the DRAM burst lengths (BL), the burst need to be terminated. The DRAM controller terminates the burst by driving a pre-charge cycle and asserting the DM signals.

Note

In case of open pages, the burst is not terminated with a pre-charge. This is done to keep the page open (see 11.5 “SDRAM Open Pages” on page 138).

Since the DRAM controller effectively accesses 128-bits for each cycle, it always accesses the DRAM to 128-bit aligned addresses and uses DM to mask non-desired writes. For example, a three 64-bit word burst write to offset 0x1 is executed as a write of four 64-bit words to offset 0x0, with DM masking the first 64-bit word. Similarly, in case of a read from a non 128-bit aligned address, the DRAM controller reads the whole 128-bit data from DRAM and ignores the first 64-bit word.

A CPU cache line read requires linear wrap around ordering on a 32-byte cache line basis. If first qword offset is 0x0 or 0x1, the DRAM controller read four qwords from DRAM offset 0x0 (0x0,0x1,0x2,0x3), and drives in the appropriate order to the CPU interface. If first qword offset is 0x2 or 0x3, it reads four qwords from DRAM offset 0x2 (0x2,0x3,0x0,0x1). The DRAM wraps around the BL of four.

All other unit accesses to DRAM, requires straight linear ordering. If the required access crosses the BL boundary (e.g. PCI read of 5 qwords from offset 0x2), the DRAM controller drives a new column address (asserting a new CAS) when crossing BL and prevents the DRAM from wrapping around the address.

11.4.1 SDRAM Bank Interleaving

The MV64360/1/2 supports both physical banks (CS[3:0]) interleaving and virtual banks (BA[1:0]) interleaving.

Interleaving provides higher system performance by hiding a new transaction’s active cycles during a previous transaction’s data cycles. This technique gains maximum utilization of the DRAM bus bandwidth.

If the first transaction does not require early pre-charge, the DRAM controller may drive a command cycle of the second transaction and delay the pre-charge cycle of the first cycle, to gain maximum bus utilization.

Note

A pre-charge is required to each bank at the end of the burst, unless the page is kept open, see 11.5 “SDRAM Open Pages” on page 138.

Interleaving occurs when there are multiple pending accesses to different DRAM banks, whether they are virtual banks distinguished by different BA[1:0] values or physical banks distinguished by different CS[3:0].

To achieve maximum bank interleave, the DRAM controller contains four transactions queue. Transactions received from the different units are pushed into the queues, based on their target DRAM bank. Transactions targeted to different banks may be placed in different queues. The DRAM controller arbitrates between the four queues to select the next transaction to be driven to the DRAM.

For example, the DRAM controller receives four consecutive requests (0, 1, 2, 3) to DRAM banks (A,A,B,B). The DRAM controller issues the transactions to the DRAM in the order of ‘0, 2, 1, 3’ to gain full bank interleaving.

Note

Transactions are not necessarily issued to the DRAM in the order they have been received from the requesting units. Even multiple transactions from the same requester (e.g. CPU) might be executed out of order, if they target different banks.

Disable virtual bank interleaving and physical bank interleaving via the SDRAM Configuration register's `VInter` bit [15] and `PInter` bit [14], see Table 287 on page 489.

11.4.2 SDRAM Address Control

The SDRAM Address Control register's `AddrSel` bits [3:0] (see Table 299 on page 497) defines how address bits driven by the requesting unit to the DRAM controller are translated to row and column address bits on DA[13:0] and BA[1:0]. This flexibility lets the designer choose the address setting that gives the software the best chance for virtual banks interleaving.

For example, the CPU and PCI access the same physical bank, and each of them is using a different 16 MB slice of the DRAM, configuration of address bits[25:24] to map BA[1:0] results in bank interleaving whenever there are consecutive CPU and PCI accesses to DRAM.

The row and column address translation is different for 64/128 Mb, 256/512 Mb or 1 Gb DDR DRAM devices, as well as for x4,x8,x16 or x32 devices, as shown in Table 37 through Table 39. DRAM density (64/128 Mb, 256/512 Mb or 1 Gb) is configured in SDRAM Address Control register's `DCfg` bits[5:4], and DRAM organization (x4,x8,x16 or x32) is configured in SDRAM Configuration register's `DQS` bits[21:20].

Table 37: Address Control for 64/128 Mb DDR SDRAM Devices

AddSel	BA[1:0]	Row DA[11:0]	Column DA[11:0] x4,x8,x16 Devices	Column DA[11:0] x32 Devices
0x0	11,7	24-13	27, "0", 26-25, 12, 10-8, 6-3	27-25, "0", 12, 10-8, 6-3
0x1	8-7	24-13	27, “0”, 26-25, 12-9, 6-3	27-25, “0”, 12-9, 6-3
0x2	12-11	24-13	27, “0”, 26-25, 10-3	27-25, “0”, 10-3
0x3	14-13	24-15, 12-11	27, “0”, 26-25, 10-3	27-25, “0”, 10-3
0x4	22-21	24-23, 20-11	27, “0”, 26-25, 10-3	27-25, “0”, 10-3
0x5	24-23	22-11	27, “0”, 26-25, 10-3	27-25, “0”, 10-3
0x6[1]	25-24	22-11	27, “0”, 26, 23, 10-3	NA
0x7[2]	26-25	22-11	27, “0”, 24-23, 10-3	NA

1. Only used for x4 or x8 or 8Mx16 devices.
2. Only used for x4 or 16Mx8 devices.

Table 38: Address Control for 256/512 Mb DDR SDRAM Devices

AddSel	BA[1:0]	Row DA[12:0]	Column DA[12:0] x4,x8,x16 Devices	Column DA[12:0] x32 Devices
0x0	11,7	25-13	29-28, "0", 27-26, 12, 10-8, 6-3	29-26, "0", 12, 10-8, 6-3

Table 38: Address Control for 256/512 Mb DDR SDRAM Devices

AddSel	BA[1:0]	Row DA[12:0]	Column DA[12:0] x4,x8,x16 Devices	Column DA[12:0] x32 Devices
0x1	8-7	25-13	29-28, "0", 27-26, 12-9, 6-3	29-26, "0", 12-9, 6-3
0x2	12-11	25-13	29-28, "0", 27-26, 10-3	29-26, "0", 10-3
0x3	14-13	25-15, 12-11	29-28, "0", 27-26, 10-3	29-26, "0", 10-3
0x4	22-21	25-23, 20-11	29-28, "0", 27-26, 10-3	29-26, "0", 10-3
0x5	24-23	25, 22-11	29-28, "0", 27-26, 10-3	29-26, "0", 10-3
0x6	25-24	23-11	29-28, "0", 27-26, 10-3	29-26, "0", 10-3
0x7[1]	26-25	24, 22-11	29-28, "0", 27, 23, 10-3	29-27, 23, "0", 10-3
0x8[2]	27-26	25, 22-11	29-28, "0", 24-23, 10-3	NA
0x9[3]	28-27	25, 22-11	29,26, "0", 24-23, 10-3	NA

1. Only used for x4 or x8 or x16 or 16Mx32 devices.
2. Only used for x4 or x8 or 32Mx16 devices.
3. Only used for x4 or 64Mx8 devices.

Table 39: Address Control for 1 Gb DDR SDRAM Devices

AddSel	BA[1:0]	Row DA[13:0]	Column DA[13:0] x4,x8,x16 Devices	Column DA[13:0] x32 Devices
0x0	11,7	26-13	31-29, "0", 28-27, 12, 10-8, 6-3	31-27, "0", 12, 10-8, 6-3
0x1	8-7	26-13	31-29, "0", 28-27, 12-9, 6-3	31-27, "0", 12-9, 6-3
0x2	12-11	26-13	31-29, "0", 28-27, 10-3	31-27, "0", 10-3
0x3	14-13	26-15, 12-11	31-29, "0", 28-27, 10-3	31-27, "0", 10-3
0x4	22-21	26-23, 20-11	31-29, "0", 28-27, 10-3	31-27, "0", 10-3
0x5	24-23	26-25, 22-11	31-29, "0", 28-27, 10-3	31-27, "0", 10-3
0x6	25-24	26, 23-11	31-29, "0", 28-27, 10-3	31-27, "0", 10-3
0x7	26-25	24-11	31-29, "0", 28-27, 10-3	31-27, "0", 10-3
0x8	27-26	25, 23-11	31-29, "0", 28, 24, 10-3	31-28, 24, "0", 10-3
0x9[1]	28-27	26, 23-11	31-29, "0", 25-24, 10-3	NA
0xa[2]	29-28	26, 23-11	31-30, 27, "0", 25-24, 10-3	NA

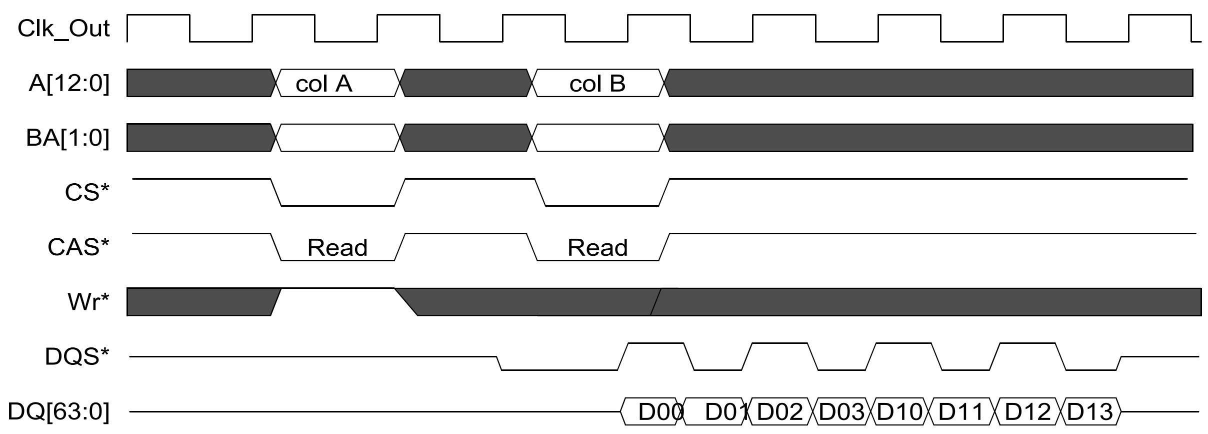
1. Only used for x4 or x8 or x16 devices.
2. Only used for x4 or x8 devices.

11.5 SDRAM Open Pages

It is possible to configure the MV64360/1/2 DRAM controller to keep DRAM pages open. It supports up to 16 pages - one page per each virtual bank.

When a page is kept open at the end of a burst (no pre-charge cycle) and if the next cycle to the same virtual bank hits the same page (same row address), there is no need for a new activate cycle. Figure 24 shows an example of access to open page.

Figure 24: Consecutive Reads to the Same Page



Via the SDRAM Open Pages Control register (see Table 293 on page 494), each of the 16 virtual banks can be configured separately to keep the page open at the end of a burst transaction, for fast consecutive accesses to the same page, or close the page, for faster accesses that follow to a different row of the same bank.

Once a page is open, it is kept open until one of the following events happen:

- There is an access to the same bank but to a different row address. The DRAM controller performs a pre-charge (closes the page) and opens a new one (the new row address).
- Refresh counter expired. DRAM controller closes all open pages and performs a refresh to all banks.

To keep pages open as long as possible, the DRAM controller does not terminate bursts to open pages with early pre-charge. Instead, it performs a burst terminate command in case of a read or asserts DM until end of the burst length (BL) in case of write.

11.6 Read Modify Write

The MV64360/1/2 supports Error Checking and Correction (ECC).

ECC is enabled via SDRAM Configuration register's `ECC` bit [18] (Table 287 on page 489). ECC checking and generation requires a 72-bit wide DRAM to store the ECC information, 64-bits for data and eight bits for ECC.

During reads, the DRAM controller can identify and correct single bit errors or detect (but not correct) two bits errors.

During writes, the DRAM controller calculates the ECC based on the write data, and drives it on CB[7:0] lines with the write data driven on DQ[63:0]. To generate the ECC on partial writes (less than 64-bits), an RMW access must perform the following:

1. Read the existing 64-bit data from DRAM.
2. Calculate ECC on the data, and compare to the ECC being read.
3. If no ECC error found, merge the new incoming data with the 64-bit read data. Calculate new ECC byte based on the data that is to be written.
   If detected single bit error, fix the corrupted bit, before merging with the new incoming data.
   If detected two bit error (non correctable), corrupt the new ECC byte after the merge (maintain two bit error in the DRAM).
4. Write the new data and new ECC byte back to the DRAM bank. On this write, all DM lines are de-asserted (write of a full 72-bit).

In case of burst write to DRAM, the MV64360/1/2 executes a RMW access of the whole burst, even if only part of the data requires RMW. When interfacing DDR SDRAM, performing RMW only for the required data is not efficient, in most cases, because of the overhead of bus turnaround cycles.

For more details on DRAM ECC support, see 19.2 “DDR SDRAM ECC” on page 313.

Note

When performing RMW access, the MV64360/1/2 checks ECC and reports an error, if necessary, on the whole data being read. The error is not just reported on the part of the data that was originally meant to be modified.

The DRAM controller also performs a RMW access when interfacing DRAM DIMMs that are based on x4 DRAM devices, due to the lack of DM signals in such DIMMs. See 11.10 “DDR DIMM Support” on page 142 for full details.

11.7 SDRAM Refresh

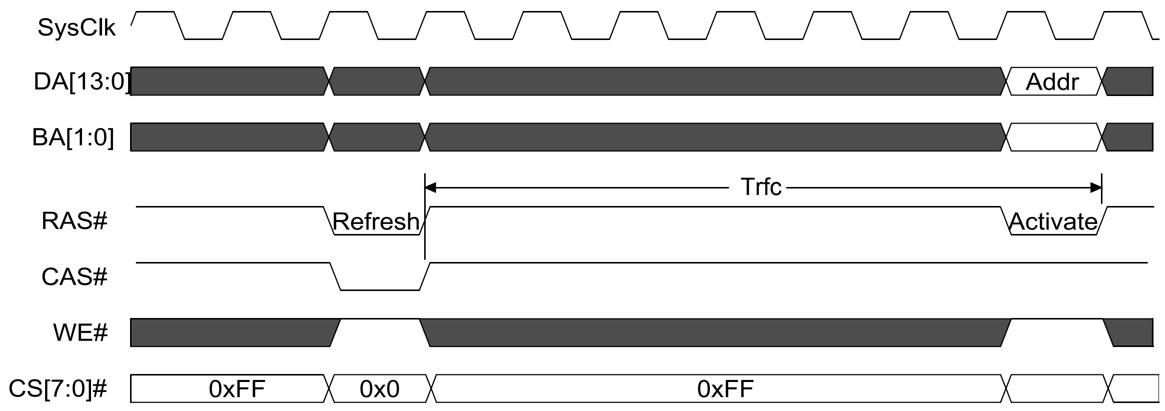
The MV64360/1/2 implements standard CAS before RAS refreshing.

The refresh rate for all banks is determined according to the 14-bit Refresh value in SDRAM Configuration register. For example, the default value of Refresh is 0x200. If the ClkOut frequency is 100 MHz (10ns cycle), a refresh sequence occurs every 5.12 us.

Every time the refresh counter reaches its terminal count, a refresh request is sent to the DRAM Controller. It has the higher priority over any other DRAM access request. As soon as the current outstanding DRAM transactions complete, the DRAM controller pre-charge all banks (both the ones that are opened, and the ones that are not), and performs an auto refresh command to all DRAM banks.

Figure 25 shows a refresh cycle example.

Figure 25: DRAM Refresh



Note

The DRAM controller will not issue a new access to DRAM (new activate cycle) for the number of Trfc cycles as specified by DDR DRAM AC spec.

11.8 SDRAM Initialization

The DRAM controller executes the DDR DRAM initialization sequence following reset de-assertion.

The initialization sequence consists of the following steps:

1. Pre-charge to all DRAM banks (all four physical banks).
2. To enable the DRAM DLL, load the Xtended DRAM Mode register, see Table 294 on page 495.
3. Load the DRAM Mode register with the DRAM parameters.

Notes

- The software can change these parameters as described in 11.9 SDRAM Operation Mode Register.
- Also, this step resets the DRAM DLL.

4. Wait 200 cycles.
5. Pre-charge all banks.
6. Wait two auto refresh cycles.
7. Load DRAM Mode register with the DRAM parameters and with the reset DLL bit [8] de-activated.

Notes

- The DRAM controller postpones DRAM initialization sequence until the internal DLL is stable. In case of a serial ROM initialization, it also waits for initialization to complete.
- The DDR DRAM spec requires at least 200us of stable clock preceding the above sequence. The SDRAM controller starts the initialization sequence immediately after reset. Assuming that the reset period is longer than 200us.
- The DRAM controller postpones any attempt to access DRAM before the initialization sequence completes.
- DDR SDRAM specification requires a gap of minimum 200 cycles between EMRS transaction, and the next read command. To meet this requirement, the MV64360/1/2 must always use t_{MRD} of 200.

11.9 SDRAM Operation Mode Register

In addition to the normal DRAM operation mode, DRAM controller also supports special DRAM commands through the DRAM Operation Mode register. These operations include:

- Normal SDRAM Mode (default mode).
- NOP Commands.
- Pre-charge All Banks.
- Load DRAM Mode Register.
- Load DRAM Extended Mode Register.
- Force a Refresh Cycle

The register contains 3-bits of command type. Once the CPU changes the register default to one of the command types, the SDRAM controller executes the required command, resets the register back to the default value, and returns to normal operation. The CPU must poll on this register to identify when the DRAM controller is back in normal operation mode.

Note

This mechanism is different than the one in the GT-64240/60 devices.

When using DDR DRAM DIMMs, the DRAM parameters are recorded in the DIMM Serial Presence Detect (SPD) serial ROM. The CPU can read the SPD via the MV64360/1/2 TWSI interface and program the DRAM parameters accordingly, using the Load Mode register command.

CPU must not attempt to change the DRAM Mode register setting, prior to DRAM controller completion of the DRAM initialization sequence. In order to guarantee this restriction, it is recommended that the CPU will set the SDRAM Operation Mode register to NOP command, then will perform read polling until the register is back in Normal operation value, and only then, set DRAM Mode register to it’s new value.

11.10DDR DIMM Support

The JEDEC (Joint Electron Device Engineering Council) standard defines many types of DDR DRAM DIMMs, such as:

- Registered or unbuffered DIMMs
- 64-bit or 72-bit wide (for ECC)
- One or two physical banks
- Different bank organization - 16 x4 devices, 8 x8 devices, 4 x16 devices
- Different densities

The MV64360/1/2 SDRAM controller supports all of these DIMMs.

The MV64360/1/2 supports up to 18 DQS (data strobe) signals. The number of DQS pin required depends on the DIMM configuration - whether it comprises x4, x8, or x16 devices. DQS is used by the DRAM controller as a read data strobe. Each DQS pin is coupled to part of the DQ data pins, as shown in Table 40.

Table 40: DQS Multiplexing

Data	Data Strobe	
	x4	x8/x16
CB[7:4]	DQS[17]	DQS[8]
DQ[63:60]	DQS[16]	DQS[7]
DQ[59:56]	DQS[7]	DQS[7]
DQ[55:52]	DQS[15]	DQS[6]
DQ[51:48]	DQS[6]	DQS[6]
DQ[47:44]	DQS[14]	DQS[5]
DQ[43:40]	DQS[5]	DQS[5]
DQ[39:36]	DQS[13]	DQS[4]
DQ[35:32]	DQS[4]	DQS[4]
CB[3:0]	DQS[8]	DQS[8]
DQ[31:28]	DQS[12]	DQS[3]
DQ[27:24]	DQS[3]	DQS[3]
DQ[23:20]	DQS[11]	DQS[2]
DQ[19:16]	DQS[2]	DQS[2]
DQ[15:12]	DQS[10]	DQS[1]
DQ[11:8]	DQS[1]	DQS[1]
DQ[7:4]	DQS[9]	DQS[0]
DQ[3:0]	DQS[0]	DQS[0]

Configure the MV64360/1/2 to the appropriate DQS multiplexing via SDRAM Configuration register's `DQS` field (bit [21:20], see Table 287 on page 489.

Note

The JEDEC standard does not specify a DIMM based on x32 devices. A x32 device has a single DQS pin for the entire 32-bit. When interfacing such devices, use DQS[0] and DQS[4] as the data strobe of DQ[31:0] and DQ[63:32], respectively (and DQS[8] for CB[7:0]). The rest MV64360/1/2 DQS pins must be pulled up.

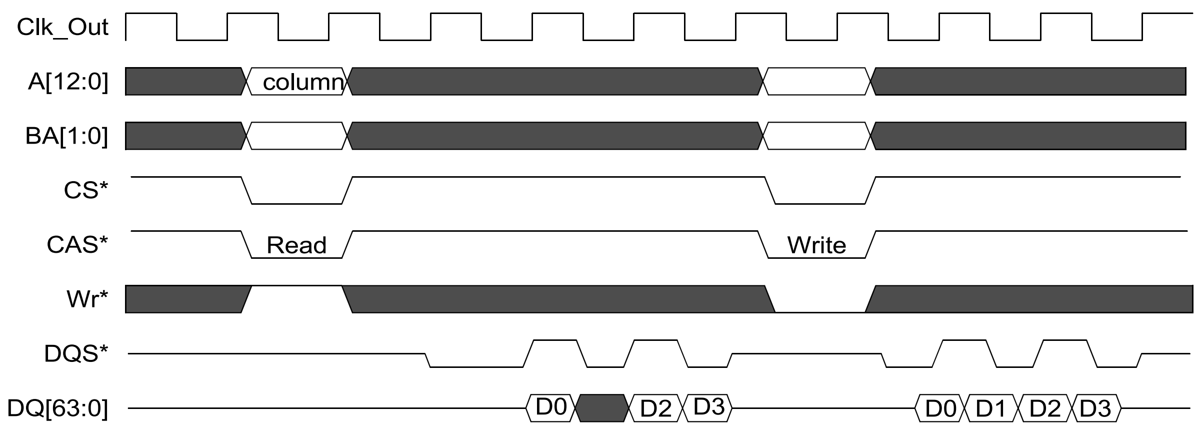
DDR DIMMs of x4 devices do not support data mask (DM). Resulting in the following restrictions:

- There is no byte/word/dword write support. All writes must be 64-bit wide.
- There is no way to terminate a burst write earlier than the burst length (BL) boundary. In the presence of DM signals, a burst can be early terminated with pre-charge + DM assertion.

Due to these x4 DDR DRAM restrictions, the MV64360/1/2 DRAM controller must perform a RMW access for partial writes (writes of less than 64-bits), even if ECC is not enabled. Additionally, it performs RMW even to non-partial writes, up to the next burst length (BL) boundary.

An example is shown in Figure 26. There is a write of a full 64-bit data to offset 0x1. Due to the lack of DM signals, the DRAM controller reads four 64-bit data from offsets 0x0, 0x1, 0x2, and 0x3. It then replaces the data being read from offset 0x1 with the new write data and writes back the whole burst.

Figure 26: Write to x4 DDR DIMM Example



When interfacing with registered DDR DRAM DIMMs, all address and control signals (DA[13:0], BA[1:0], RAS#, CAS#, WE# and CS#) are registered on the DIMM. This means that the signals arrive to the SDRAM device one cycle after they are driven by the DRAM controller. It also means that read data arrives back to the SDRAM controller one cycle later (in comparison to unbuffered DDR DRAM DIMM).

When the SDRAM controller is configured to registered DRAM via the SDRAM Configuration register's `RegDRAM` bit [17], it drives write data one cycle later and samples the read data one cycle later (in comparison to unbuffered DDR SDRAM DIMM).

Note

Registered DRAM support is not necessarily for registered DIMMs. The control signals can be registered on board.

11.11DRAM Clocking

The DRAM controller supports the DRAM running at a higher frequency than the MV64360/1/2 core clock (up to 183 Mhz) or running with the same clock frequency. Select the DRAM clock domain via DevAD [18] in the reset configuration (Table 117 on page 348).

The MV64360/1/2 contains two PLLs – one per the core clock tree, and one per the DRAM controller.

When configured to run with the core clock domain, the DRAM controller clock tree is connected to the core clock PLL. In this mode, no synchronization is required on transactions coming from the different MV64360/1/2 units, to the DRAM controller.

When configured to run with its own clock domain (different frequency than the core clock domain), the DRAM controller clock tree is connected to its own PLL. This PLL is used as a clock synthesizer. It receives the external SysClk as a reference clock and generates, from it, a multiplied clock. The DRAM frequency in this mode is calculated according the following formula:

$$F_{ClkOut} = F_{SysClk} * (N+2) / (M+2)$$

Notes

- M and N are set via DevWE[3:0]/DevDP[3:0] and TxD0[6:1] balls, respectively. See Table 117 on page 348.
- M value must meet the following constraint: 4 MHz > F_{SysClk}/(M+2) >= 2.5 MHz.

In this mode, requests from the different MV64360/1/2 units, are synchronized to the DRAM clock domain, before being driven to DRAM. Also, read data returned from DRAM is synchronized back to core clock domain.

Note

This synchronization has latency impact. Only for high DRAM frequency (183 MHz) when that kind of DRAM throughput is required, it is recommended to use this asynchronous mode.

11.12DRAM Address/Data Drive

The DRAM clock is driven by the MV64360/1/2 ClkOut/ClkOut* differential pair. All DRAM address and control signals driven by MV64360/1/2 (single data rate signals) are coupled to the rising or to the falling edge of this clock, according to the DevAD[19] reset configuration.

Note

Typically, address and control signals should be driven with the rising edge of ClkOut. However, under certain board topology and DRAM load, there may be a hold time problem on these signals. In this case, use the falling edge setting ('0').

To meet high frequency operation, the DRAM controller contains multiple pipeline stages on the control path. For operation at up to 133MHz, use two pipeline stages. Set the DUnit Control (Low) register's `CtrlPipe` bits [5:4] to '1', see Table 288 on page 490. For operation at higher frequency, set to three pipeline stages, `CtrlPipe` = '3'.

The front-end logic of the DRAM controller is responsible for correct drive of the double data rate data with the DQS signals, as well as unpack of the data from 128-bit SDR to 64-bit DDR.

During a write transaction, 128-bit wide data is pulled out of the write buffer and driven as 64-bit DDR on the bus. The first 64-bit is driven with rising edge of ClkOut and the second 64-bit with falling edge of ClkOut. The DRAM controller drives DQS (data strobe) along with the data. The DDR DRAM specification requires very accurate DQS timing in respect to the DRAM clock (DQS toggle on quarter of a cycle). The DRAM controller uses a configurable delay line (DFCDL) to "put the DQS in place".

Note

After silicon testing, exact guide lines on DFCDL setting will be provided.

11.13DRAM Read Data Sample

The front-end logic of the DRAM controller is responsible for correct sampling of the double data rate data with DQS signals, as well as the pack of data from 64-bit DDR to 128-bit SDR. 64-bit DDR read data is latched via the received DQS. The first 64-bits are sampled with DQS high and the next 64-bit data with DQS low. Again, the DRAM controller uses a DFCDL to shift DQS to the required sampling point.

To meet the DDR DRAM AC specification, packed 128-bit read data cannot simply be sampled with the internal DRAM controller clock. The exact sample point depends on CL (CAS latency) and board traces. The DRAM controller is using a feedback clock FBClkIn as a reference to the received read data. Also, the controller drives a StartBurst# signal that is routed back to the controller as StartBurstIn# feedback. This signal is used as an indication to the first read data. The DRAM controller supports several methods for accurate sampling of the read data. Select the proper method via DevAd[24:22] reset configuration, see Section 27. "Reset Configuration" on page 347

Note

The read data sample method depends on DRAM clocking mode (sync or non-sync) and board topology. Detailed recommendations on this setting will be available after silicon testing.

The DFCDL's SRAM must be initialized by writing 64 times to the SRAM Data0 register.

The following is the recommended initialization sequence for the DFSCDL when using DRAM at 133 MHz.

```
12'b000000_000000
12'b000001_000001
12'b000010_000010
12'b000011_000011
....
12'b111101_111101
12'b111110_111110
12'b111111_111111
```

The following is the recommended initialization sequence for the DFSCDL when using DRAM at 183 MHz.

```
12'b000000_000000
12'b000000_000001
12'b000000_000010
12'b000000_000011
12'b000000_000100
12'b000001_000101
12'b000010_000110
12'b000011_000111
....
12'b111001_111101
12'b111010_111110
12'b111011_111111
```

Note

This sequence is a preliminary recommendation. Detailed recommendations on this sequence will be available after silicon testing.

11.14DRAM Interface I/O Signaling

The DRAM interface supports the standard SSTL_2 signaling. Use parallel termination on all DRAM interface signals (including CLK and CLK#). Termination resistors values are determined according to board simulations.

Note

For the SSTL signaling standard, see EIA/JEDEC standard EIA/JESD8-9 (Stub series terminated logic for 2.5 volts, SSTL_2).

For better signal integrity and board timing design, the DRAM interface pads have a calibration mechanism to control pad drive and impedance. There are separate calibration circuits for the DRAM address/control signals and for the DRAM data signals. Connect the DRAM_ACAL and DRAM_DCAL pins to WDD_DRAM via resistors. The resistor size is 25 ohms for SSTL class 2. After reset de-assertion, the calibration logic tunes the DRAM interface output drivers impedance.

Section 12. Device Controller

The device controller supports up to five banks of devices. Each bank supports up to 512 MB of address space, resulting in total device space of 2.5 GB.

Each bank has its own parameters register. Bank width can be programmed to 8-, 16-, or 32-bits. Bank timing parameters can be programmed to support different device types (e.g. Sync Burst SRAM, Flash, ROM, I/O Controllers).

The five individual chip selects are typically separated into four individual device banks and one chip select for a boot device. The boot device bank is the same as any of the other banks except that it’s default address map matches the PowerPC CPU boot address (0xFFF0.0100) and that it’s default width is sampled at reset.

The device AD bus is a 32-bit multiplexed address/data bus. During the address phase, the device controller puts an address on the AD bus with a corresponding chip select asserted and DevRW indicated. It de-asserts Address Latch Enable (ALE) to latch the address, the chip select, and read/write signals by an external latch (or register).

CS# must then be qualified with CSTiming# to generate the specific device chip select and DevRW# must be qualified with CSTiming# to generate a read or write cycle indication. The CSTiming# signal is active for the entire device access time specified in the device timing parameters register

During the data phase, the device controller drives data on the AD bus, in case of write cycle, or samples data driven by the device, in case of read cycle. Use Wr[3:0]# as the byte enable signal during a write transaction.

Notes

- The MV64360/1/2 does not support READ byte enables.
- The MV64360/1/2 does not support multiple masters on the AD bus or external master access to the different MV64360/1/2 interfaces via the device bus.
- All device controller signals, including CSTiming#, are floated for the entire reset assertion period and an additional five SysClk cycles after reset de-assertion. Since the device chip select is qualified with CSTiming#, this signal must be pulled up or driven for the five additional cycles by some external logic, to prevent undesired accesses to the device.

12.1 Device Controller Implementation

The device interface consists of 128 bytes of write buffer and 128 bytes of read buffer. It can absorb up to four read plus four write transactions.

On a write transaction to a device, the data is written to the write buffer and then driven to the device bus. As soon as a device access is requested, the device controller drives an address on the AD bus for two cycles and de-asserts ALE, so it will be used by external logic to latch the address, chip select and DevRW#.

Note

The CS# must be qualified by the CSTiming# signal to generate the device’s actual chip select.

On the next cycle after ALE de-assertion, the device controller pops data from the write buffer and drives it on the bus. It drives the valid data based on the device timing parameters, see "Device Timing Parameters".

In case the device controller is still serving a previous transaction on the bus, the whole burst write is posted into the write buffer and driven to the device bus when all the previous transactions are completed.

On a read transaction, the device controller samples the read data from the AD bus. The sample window is determined according to the device timing parameters. Data is driven back to the requesting unit, when the whole read data is placed in the read buffer, or as soon as the first 8 bytes are available, depending on Device Interface Control register's `RdTrig` bit [16]setting, see Table 320 on page 508.

12.2 Device Timing Parameters

To allow interfacing with very slow devices and fast synchronous SRAMs, each device can be programed to different timing parameters.

12.2.1 TurnOff

The TurnOff parameter defines the number of SysClk cycles that the MV64360/1/2 does not drive the AD bus after the completion of a device read. This prevents contentions on the device bus after a read cycle from a slow device. The minimum setting of this parameter is 0x2.

12.2.2 Acc2First

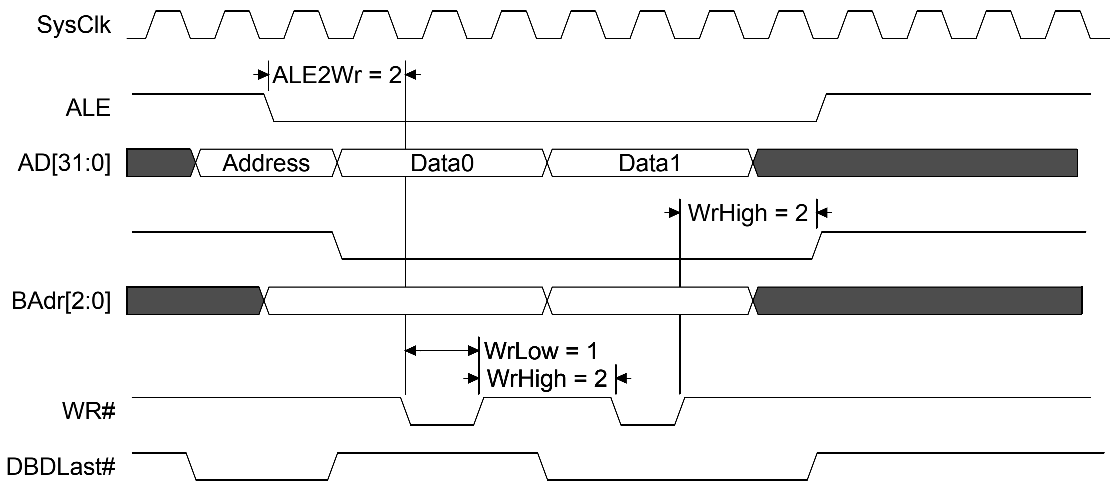
The Acc2First parameter defines the number of SysClk cycles from the assertion of ALE to the cycle that the first read data is sampled by MV64360/1/2. Extend this parameter by extending the Ready# pin, see 12.4 "Ready# Support" on page 151. The minimum setting of this parameter is 0x3.

12.2.3 Acc2Next

The Acc2Next parameter defines the number of SysClk cycles between the cycle that samples data N to the cycle that samples data N+1 (in burst accesses). Extend this parameter can be extended by the Ready# pin. The minimum setting of this parameter is 0x1.

Figure 27 shows a device read timing parameters example.

Figure 27: Device Read Parameters Example



12.2.4 ALE2Wr

The ALE2Wr parameter defines the number of SysClk cycles from ALE de-assertion cycle to Wr[3:0]# assertion. The minimum setting of this parameter is 0x1.

12.2.5 WrLow

The WrLow parameter defines the number of SysClks that Wr[3:0]# is active (low). Extend this parameter by the Ready# pin. BAdr and Data are kept valid for the whole WrLow period. This parameter defines the setup time of address and data to Wr rise. The minimum setting of this parameter is 0x1.

12.2.6 WrHigh

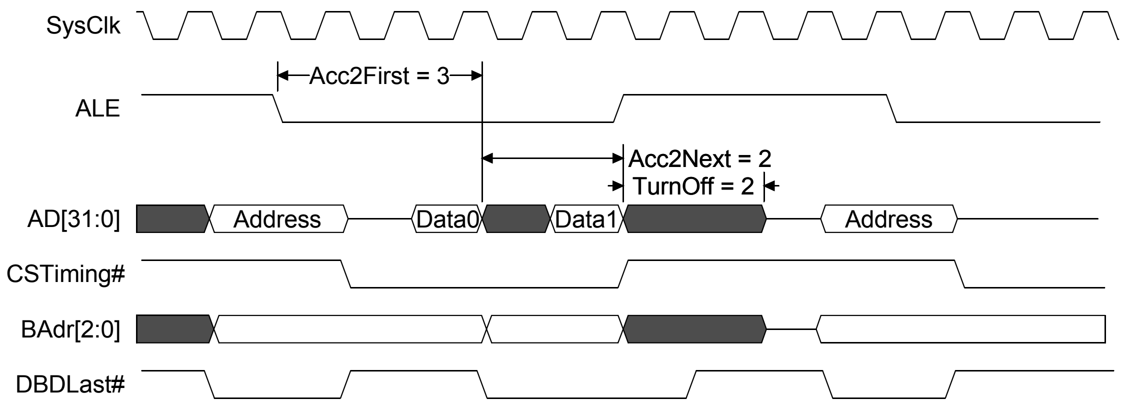
The WrHigh parameter defines the number of SysClk cycles that Wr[3:0]# is kept inactive (high) between data beats of a burst write. BAdr and Data are kept valid (don't toggle) for WrHigh-1 period This parameter defines the hold time of address and data after Wr rise. The minimum setting of this parameter is 0x0.

Notes

- Programing WrHigh to '0' is only used for zero wait states burst access (e.g. sync burst SRAM access). It is only allowed when WrLow is set to 1.
- If setting WrHigh to '0', BAdr and write data toggle every cycle.
- If setting WrHigh to '1', BAdr and write data toggle the same cycle Wr toggles from High to Low.

Figure 28 shows a device write timing parameters example.

Figure 28: Device Write Parameters Example

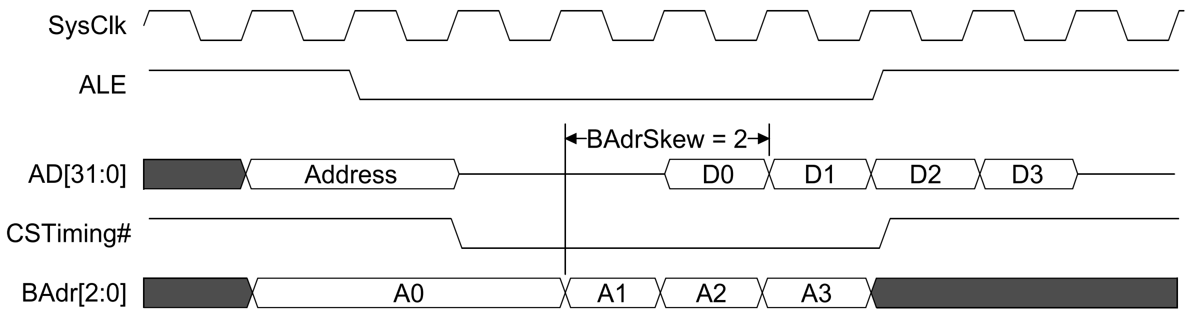


12.2.7 BAdrSkew

The MV64360/1/2 also supports early toggle of burst address during read access. BAdrSkew parameter defines the number of SysClk cycles from BAdr toggle, to read data sample. This parameter is useful for SyncBurst SRAM type of devices, where the address precedes the read data by one (Flow Through SRAM) or two (Pipelined SRAM) cycles.

Figure 29 shows a BAdrSkew usage example.

Figure 29: Pipeline Sync Burst SRAM Read Example



12.3 Data Pack/Unpack and Burst Support

The device controller supports 8-, 16-, or 32-bit wide devices. Specify the device width in the `DevWidth` field of each device parameters register.

The device controller supports up to 32 byte burst to a 32-bit wide device, and up to 8 bytes burst to 8- or 16-bit wide device. The burst address is supported by a dedicated three bit BAdr[2:0] bus. This bus must be connected directly to the device address bus (not like the latched address on the multiplexed AD bus). The device controller supports pack/unpack of data between the device (8-, 16-, or 32-bit wide) and the initiator (PCI, CPU, DMA).

An attempt to access a device with a non-supported burst results in an interrupt assertion.

Notes

- Since bursts to 8- and 16-bit devices are limited to eight bytes, never place these devices in a CPU cacheable region (that requires bursts of 32 bytes). Also, it is only possible to read these devices from a PCI's non-prefetchable region.
- Motorola MPC7450 performs burst reads during boot, even though its caches are disabled. When interfacing this CPU, the boot device must be 32-bit wide.
- Since bursts to 32-bit devices are limited to 32 bytes, DMA, GbE controller, MPSCs, or PCI accesses to such devices must not exceed 32 bytes.

During an access to an 8- or 16-bit wide device, the device controller calculates the start and end address of the transaction, based on the byte enables it receives from the requesting unit. For example, a CPU requests a read of 4 bytes, starting at offset 0x2 from a 16-bit device, the device controller issues a burst access of two 16-bit words to the device, to offset 0x2. Or, for instance, a PCI write of 5 bytes to offset 0x1 of an 8-bit wide device. On the 64-bit PCI bus, this write appears as data of 0x??vv.vvv? ('V" stands for valid byte,? stands for non valid byte) with byte enables of 1'b11000001. On the device bus, this write is translated to a burst of 5 bytes, starting at address offset 0x1.

Note

The device controller does not support non-sequential byte enables to 8- or 16-bit wide devices (e.g. write of 32-bit word to 8-bit wide device with byte enable 1'b1010).

As previously described, the device controller need to pack read data from 8/16/32-bit wide device to the MV64360/1/2 internal 64-bit data path. Since the device controller access to 8/16-bit devices is limited to 8 bytes, it is obvious that the device controller can drive the read data to the requesting unit only when the transaction on the device bus completes. However, on burst read access a 32-bit wide device, the device controller has valid read data to return to the requesting unit after packing of each two 32-bit words to 64-bit.

On burst read access to a 32-bit device, the device controller can return read data to the requester, as soon as first 64-bit data is available, or only when the whole burst data is available. If Device Interface Control register's `RdTrig` bit is set to 1, data is returned to the requester, only when the whole burst data is valid (store & forward policy). This is useful when interfacing a device with long wait states between data beats, to not waste the MV64360/1/2 crossbar bandwidth. If `RdTrig` is set to 0, data is returned as soon as packed 64-bit data is valid.

12.4 Ready# Support

Ready# input is used to extend the programable device timing parameters. This is useful for two cases:

- Interfacing a very slow device, which has access time greater than the maximum programable values
- Interfacing a device with a non deterministic access time (access time depends on other system events and activity)

Ready can extend the following timing parameters - Acc2First, Acc2Next and WrLow. During a read access, the device controller is first counting SysClk cycles based on Acc2First programable parameters. If at the time Acc2First is expired, Ready# input is not asserted, it keeps waiting until Ready# is sampled asserted, and only then samples first read data. Similarly, if at the time Acc2Next is expired, Ready# is not asserted, it keeps waiting until Ready# is sampled asserted, and only then samples next read data. On a write access, if at the time WrLow is expired, Ready# input is not asserted, it keeps driving write data until Ready# is sampled asserted Figure 30, Figure 31, and Figure 32 show examples of the Ready# operation.

Notes

- If Ready# is not used, Ready# pin must be tied low.
- If the WrLow or WrHigh timing parameter is set to ’0’, Ready# is not supported during a write access.
- When interfacing a device with a non deterministic access time, timing parameters should be set to their minimum values, and the actual access time is controlled via Ready# pin.

Figure 30: Ready# Extending Acc2First

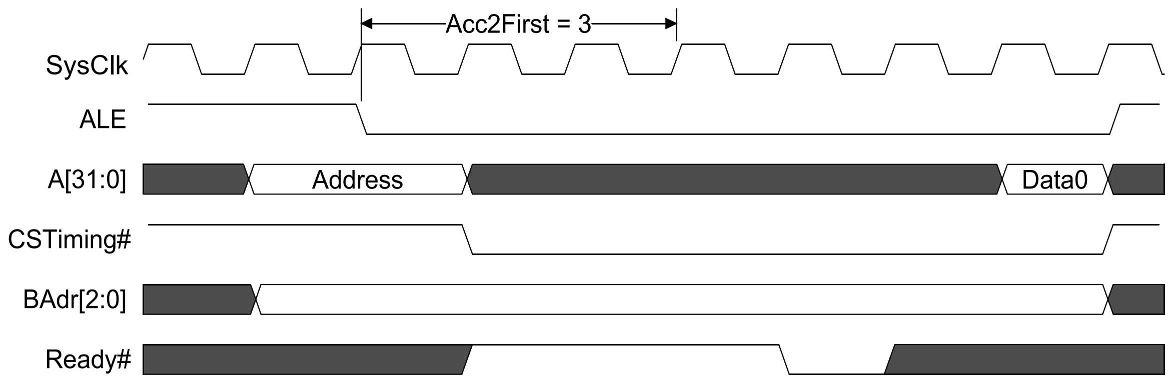


Figure 31: Ready# Extending Acc2Next

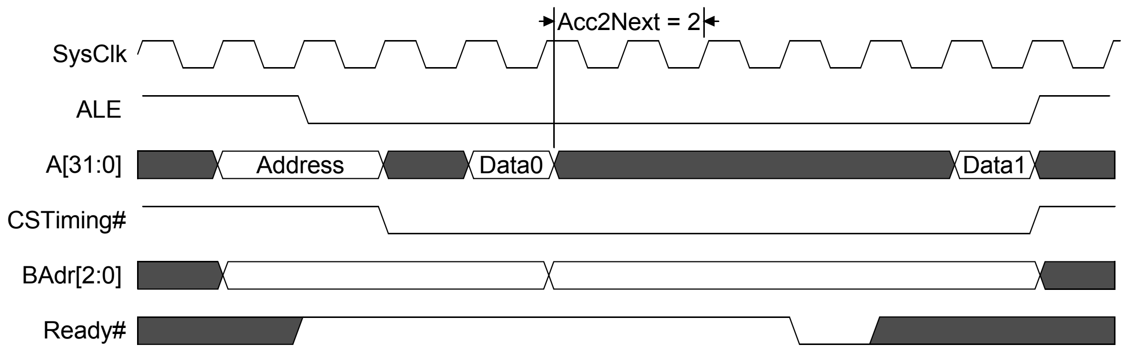
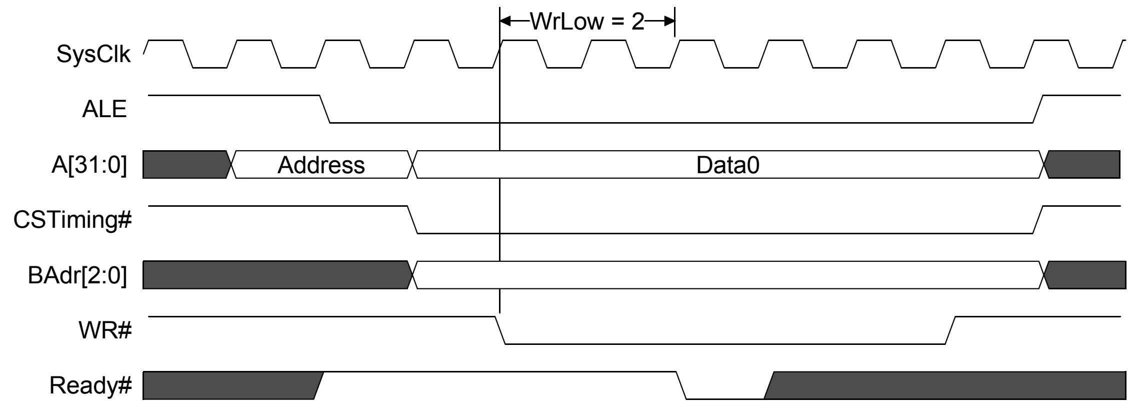


Figure 32: Ready# Extending WrLow Parameter



To prevent system hang due to a lack of Ready# assertion, the MV64360/1/2 implements a programable timer that allows termination of a device access even without Ready# assertion. If during a device access the time out timer expires, the device controller completes the transaction as if Ready# was asserted and generates an interrupt. Setting the timer to 0x0 disables it, and the device controller waits for Ready# forever.

Note

The timer is used only for preventing system hang due to a lack of Ready# pin assertion. If expired (which means a system hardware problem), the device controller completes the transaction ignoring Ready#. This might result in bad data read/write from/to the device. The timer must be programed to a number that must never be exceeded in normal operation.

12.5 Parity Support

The MV64360/1/2 device controller supports generating and checking of data parity via the DevDP[3:0] pins.

- DevDP[0] is the parity bit for AD[7:0].
- DevDP[1] is the parity bit of AD[15:8].
- DevDP[2] is the parity bit for AD[23:16].
- DevDP[3] is the parity bit of AD[31:24].

Enable/disable parity on a device chip select basis via the Device Bank Parameters register's `DPEn` bit [30], see Table 315 on page 505. Even or Odd parity is selectable via Device Interface Control register's `ParSel` bit [20]. It also supports address parity.

During the address phase, the MV64360/1/2 drives parity per each of the four address byte.

Note

Since the MV64360/1/2 is never a slave on the device bus, it does not check address parity. It only generates address parity (to be checked by the target device).

During write access, the MV64360/1/2 drives parity per each byte, with the same timing as the write data. It drives the parity for the whole 4 bytes (regardless of the device width). If errors propagation is enabled via Device Interface Control register's `PerrProp` bit [19], and the data received by the Device controller is marked as erroneous (e.g. CPU interface detects bad data parity on the CPU bus during CPU write to device), the MV64360/1/2 will force bad parity on all four parity bits.

During read access, it samples the parity bit(s), at the same timing of the read data. It calculates parity on the incoming read data and compares to the sampled parity bit(s), for the relevant bytes (based on device width). In case of mismatch, it sets a parity error indication, and asserts an interrupt, if not masked. The address, data and parity are latched in Device Error Address, Data, and Parity registers respectively.

Note

When ever the MV64360/1/2 is the driver of the AD bus, it drives parity on the parity bits for the whole 32-bits of the AD bus, regardless, if the bus is in idle state, address phase or write data phase. More over, it always drives parity for the whole 32-bit, even if not all of them are being used.

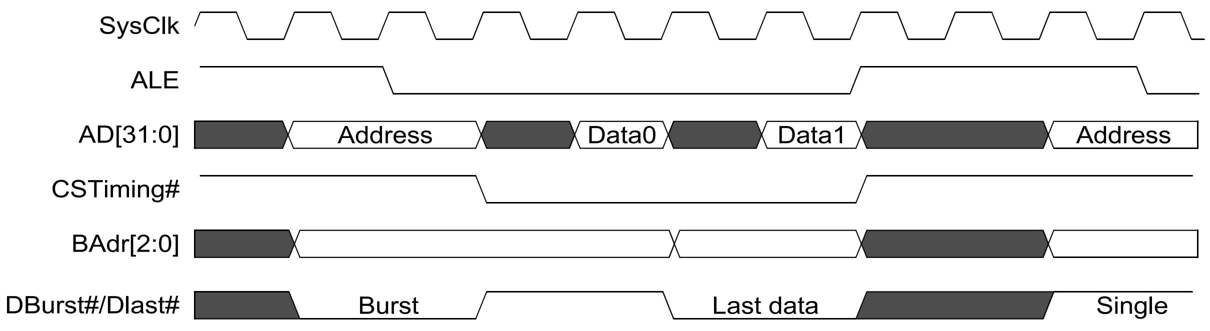
The device controller also supports forcing of user defined parity to the device bus for debug purposes.

12.6 Additional Device Interface Signaling

To make it easy to glue external logic on the device bus, the MV64360/1/2 supports burst and last indication via MPP lines. DBurst#/DLast# is driven low on the address phase (need to be latched via ALE#) to indicate a burst access and is driven low on the last data phase to indicate the last data transfer.

Figure 33 shows an example of these additional signals.

Figure 33: DBurst#/Dlast# Example



12.7 Interfacing With 8/16/32-Bit Devices

To connect the devices correctly, follow the pin connection information listed in the following tables.

Table 41: 8-bit Devices

Connection	Connect...	To...
Device Address	BADR[2:0] DevAD[27:2] ALE Latch Outputs	Device Address Bits [2:0] Address Latch Inputs Address LE Device Address Bits [28:3]
Device Data	DevAD[7:0]	Device Data Bits [7:0]
Device Control Pins	ALE DevAD[1] DevAD[0] DevAD[31:28]	Control latch LE Becomes DevRW# Becomes BootCS# Becomes CS[3:0]#
Write Strobes	DevWE[0]#	Device Data Bits[7:0] Write Strobe

Table 42: 16-bit Devices

Connection	Connect...	To...
Device Address	BADR[2:0] DevAD[27:3] ALE Latch Outputs	Device Address Bits[2:0] Address Latch Inputs Address LE Device Address Bits [27:3]
Device Data	DevAD[15:0]	Device Data Bits [15:0]
Device Control Pins	ALE DevAD[1] DevAD[0] DevAD[31:28]	Control latch LE Becomes DevRW# Becomes BootCS# Becomes CS[3:0]#
Write Strobes	DevWE[0]# DevWE[1]#	Device Data Bits[7:0] Write Strobe Device Data Bits[15:8] Write Strobe

Table 43: 32-bit Devices

Connection	Connect...	To...
Device Address	BADR[2:0] DevAD[27:4] ALE Latch Outputs	Device Address Bits [2:0] Address Latch Inputs Address LE Device Address Bits [26:3]
Device Data	DevAD[31:0]	Device Data Bits [31:0]

Table 43: 32-bit Devices (Continued)

Connection	Connect...	To...
Device Control Pins	ALE DevAD[1] DevAD[0] DevAD[31:28]	Control latch LE Becomes DevRW# Becomes BootCS# Becomes CS[3:0]#
Write Strobes	DevWE[0]# DevWE[1]#Dev WE[2]# DevWE[3]#	Device Data Bits[7:0] Write Strobe Device Data Bits[15:8] Write Strobe Device Data Bits[23:16] Write Strobe Device Data Bits[31:24] Write Strobe

12.8 PCI and CPU to Device Bus Addressing

Use the following tables to determine the 32- and 36-bit PCI and CPU to device bus addressing.

Table 44: 32-bit PCI and CPU Bus Addressing

Device	PCI Address	CPU Address	Device address	
			Device Address[x:3]	BAdr[2:0]
8 bit device	PAD[31..0]	A[0..31]	PAD[28..3]/ A[3..28]	PAD[2..0] / A[29..31]
16 bit device	PAD[31..0]	A[0..31]	PAD[28..4]/ A[3..27]	PAD[3..1] / A[28..30]
32 bit device	PAD[31..0]	A[0..31]	PAD[28..5]/ A[3..26]	PAD[4..2] / A[27..29]

Table 45: 36-bit PCI and CPU Bus Addressing

Device	PCI Address	CPU Address	Device address	
			Device Address[x:3]	BAdr[2:0]
8 bit device	PAD[31..0]	A[0..35]	PAD[28..3]/ A[6..32]	PAD[2..0] / A[33..35]
16 bit device	PAD[31..0]	A[0..35]	PAD[28..4]/ A[6..31]	PAD[3..1] / A[32..34]
32 bit device	PAD[31..0]	A[0..35]	PAD[28..5]/ A[6..30]	PAD[4..2] / A[31..33]

Table x provides 32- and 36-bit addressing examples.

Table 46: 32-Bit Addressing Example

Device	PCI address	CPU address	Device address[...0]
8 bit device	0x12345678	0x12345678	0x12345678 (Badr[2..0] = 0x0)
16 bit device	0x12345678	0x12345678	0x091A2B3C (Badr[2..0] = 0x4)
32 bit device	0x12345678	0x12345678	0x048D159E (Badr[2..0] = 0x6)

Table 47: 36-Bit Addressing Example

Device	PCI address	CPU address	Device address[...0]
8 bit device	0x12345678	0x912345678	0x12345678 (Badr[2..0] = 0x0)
16 bit device	0x12345678	0x912345678	0x091A2B3C (Badr[2..0] = 0x4)
32 bit device	0x12345678	0x912345678	0x048D159E (Badr[2..0] = 0x6)

Section 13. PCI Interface

The MV64360 supports two 64-bit PCI/PCI-X interfaces. The MV64361 supports two 32-bit PCI/PCI-X interfaces. And, the MV64362 supports one 64-bit PCI/PCI-X interface.

Each PCI interface is configured independently at reset to run as PCI or PCI-X interface.

The PCI interface runs up to 66MHz in conventional PCI mode or up to 133MHz in PCI-X mode. Each PCI interface runs with separate clock asynchronously.

The MV64360 and MV64361devices also supports PCI to PCI, PCI-X to PCI-X, and PCI to PCI-X bridging between the two PCI interfaces.

13.1 PCI Master Operation in Conventional PCI Mode

A PCI transaction may be initiated by the CPU, IDMA, Gb MAC, MPSC, or by the peer PCI interface (P2P bridging). Each of these interfaces uses its own address decoding logic, with dedicated address windows per PCI memory space and PCI I/O space. An address match in any of the PCI address windows results in a PCI master transaction. The transaction address is the same as the initiator's cycle address, unless address remapping is used (see Section 7. "Address Space Decoding" on page 82).

Note

A PCI transaction initiated by the peer PCI interface (P2P bridging) is only supported by the MV64360 and MV64361 devices.

The MV64360/1/2 PCI master supports the following transactions:

- Memory Read
- Memory Write
- Memory Read Line
- Memory Read Multiple
- Memory Write & Invalidate
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write
- Interrupt Acknowledge
- Special Cycle
- Dual Address Cycles

The master generates Memory Write & Invalidate transaction if:

- The transaction accessing the PCI memory space requests a data transfer size equal to multiples of the PCI cache line size, with all byte enables active.
- The transaction address is cache aligned.
- Memory Write and Invalidate Enable bit in the Configuration Command register is set.

The master generates Memory Read Line transaction if:

- The transaction accessing the PCI memory space requests a data transfer size equal to multiples of the PCI cache line size.
- The transaction address is cache aligned.

A Memory Read Multiple transaction is carried out when the transaction accessing the PCI memory space requests a data transfer that crosses the PCI cache line size boundary.

Note

The MV64360/1/2 supports only cache line size of eight (8 32-bit words). Setting the PCI cache line register to any other value is treated as if cache line size is set to ‘0’.

A Dual Address Cycles (DAC) transaction is carried out if the requested address is beyond 4 GB (address bits[63:32] are not set to ‘0’).

The MV64360/1/2 PCI master performs configuration read/write cycles, Interrupt Acknowledge cycles, or Special cycles using the Config Address and Config Data registers, as recommended in the PCI specification. For full details on generating these transactions, see 13.3 “PCI Master Configuration Cycles in Conventional PCI Mode” on page 162.

The master consists of 512 bytes of posted write data buffer and 512 bytes of read buffer. It can absorb up to four 128-byte write transactions plus four 128-byte read transactions. The PCI master posted write buffer in the MV64360/1/2 permits the initiator to complete the write even if the PCI bus is busy. The posted data is written to the target PCI device when the PCI bus becomes available. The read buffer absorbs the incoming data from PCI. Read and Write buffers implementation guarantees that there are no wait states inserted by the master.

Note

IRDY# is never de-asserted in the middle of a transaction.

13.1.1 PCI Master Write Operation

On a write transaction, data from the initiator unit is first written to the master write buffer and then driven on the PCI bus. The master does not need to wait for the write buffer to be full. It starts driving data on the bus when the first data is written into the write buffer. It can also be configured to drive the data only when the whole burst is placed in the write buffer via the PCI Command register’s `MWrTrig` bit [6], see Table 378 on page 535.

On consecutive write transactions, the transactions are placed into the queue. When the first transaction is done, the master initiates the transaction for the next transaction in the queue.

The master supports combining memory writes. This is especially useful for long DMA transfers, where a long burst write is required. If combining is enabled through the PCI Command register’s `MWrCom` bit [4], the master combines consecutive burst write transactions, if possible. For combining memory writes to occur, the following conditions must exist:

- Combining is enabled through the PCI Command register’s `MWrCom` bit.
- The start address of the second transaction matches the address of data n+1 of the first transaction.
- While the first transaction is still in progress, the request for the new transaction occurs.

The master supports fast back-to-back transactions. If there is a pending new transaction in the middle of a transaction in progress, the master starts the new transaction after the first transaction ends, without inserting dead cycle. For the master to issue a fast back-to-back transaction, the following conditions must exist:

- Fast back-to-back is enabled when the Status and Command register's `FastBTBEn` bit[9] is set to '1', see Table 419 on page 555.
- The first transaction is a write.
- While the first transaction is still in progress, the new transaction request occurs.

13.1.2 PCI Master Read Operation

On a read transaction, when the initiator requests a PCI read access, the PCI master drives the transaction on the bus (after gaining bus mastership). The returned data is written into the read buffer. The PCI master drives the read data to the initiating unit as soon as the first data arrives from the PCI bus. It can also be configured to only drive the data when the whole burst read is placed in the read buffer via PCI Command register's `MRdTrig` bit.

Note

In case of a CPU burst read, regardless of `MRdTrig` bit setting, the master absorbs the full burst into the read buffer and only then drives it to the CPU interface unit in linear wrap-around order.

The master also supports combining read transactions. This is especially useful for long DMA transfers, where a long burst read is required, and the PCI target drives long burst data without inserting wait states. If combining is enabled through PCI Command register's `MRdCom` bit [5], the master combines consecutive burst read transactions. For combining read transactions to occur, the following conditions must exist:

- Combining is enabled.
- The start address of the second transaction matches the address of data n+1 of the first transaction.
- While the first transaction is still in progress, the request for the new transaction occurs.

Note

If the target cannot handle a long burst without wait states, combining read transactions is not recommended since the MV64360/1/2 holds the PCI bus for a long time without using it.

13.1.3 PCI Master Termination

If there is no target response to the initiated transaction within four clock cycles (five clocks in case of DAC transaction), the master issues a Master Abort event. The master de-asserts FRAME# and on the next cycle de-asserts IRDY#. Also, the Interrupt Cause register's `MMAbort` bit is set and an interrupt is generated, if not masked.

The master supports several types of target termination:

- Retry
- Disconnect
- Target Abort

If a target terminated a transaction with Retry, the MV64360/1/2 master re-issues the transaction. In default, the master retries a transaction until it is being served. To limit the number of retry attempts, set the Retry Counter register to a desired count value. When the master reaches this count value, it stops the retries and a bit is set in the Interrupt Cause register.

Note

The MV64360/1/2 master keeps retry a transaction until it is being served (or until Retry Counter expires). If it has multiple pending transactions in it's queue, it will not start serving a new transaction before the current retried transaction get to a completion.

If a target terminates a transaction with Disconnect, the master re-issues the transaction from the point it was disconnected. For example, if the master attempts to burst eight 32-bit dwords starting at address 0x18, and the target disconnects the transaction after the fifth data transfer, the master re-issues the transaction with address 0x2C to burst the left three dwords.

If a target abnormally terminates a transaction with a Target Abort, the master does not attempt to re-issue the transaction. A bit in the Interrupt Cause register is set and an is interrupt generated, if not masked.

13.2 PCI Bus Arbitration

The MV64360/1/2 supports both external arbiter or internal arbiter configuration through the PCI Arbiter Control register's `EN` bit [31] (see Table 383 on page 540). If the bit is set to '1', the MV64360/1/2 internal PCI bus arbiter is enabled.

Note

The internal PCI arbiter REQ#/GNT# signals are multiplexed on the MPP pins. For the internal arbiter to work, the MPP pins must first be configured to their appropriate functionality, see 26.3 "MPP I/O Pads" on page 346. Additionally, since the MPP default configuration is general purpose input, pull ups must be set on all GNT# signals.

Since the internal PCI arbiter is disabled by default (the MPP pins function as general purpose inputs), changing the configuration can only be done by the CPU or through serial ROM initialization. The configuration cannot be done by an external PCI master (since an external master will not gain PCI bus arbitration).

13.2.1 PCI Master Bus Arbitration

Whenever there is a pending request for a PCI access, the PCI master requests bus ownership through the REQ# pin. As soon as the PCI master gains bus ownership (GNT# asserted), it issues the transaction. If no additional pending transactions exist, it de-asserts REQ# the same cycle it asserts FRAME#.

The MV64360/1/2 implements the Latency Timer Configuration register, as defined in PCI specification. The timer defines number of clock cycles starting from FRAME# assertion that the master is allowed to keep bus ownership, if not granted any more. If the Latency Timer is expired, and the master is not granted (GNT# not asserted), the master terminates the transaction properly on the next data transfer (TRDY# assertion). The master re-issues the transaction from the point it was stopped, similar to the case of disconnect.

One exception is Memory Write and Invalidate command. In this case, the master quits the bus only after next cache line boundary, as defined in PCI specification.

13.2.2 Internal PCI Arbiter

The MV64360/1/2 integrates a PCI arbiters per PCI interface. Each arbiter can handle up to six external agents plus one internal agent (PCI_0/1 master).

The same arbiter is used for both conventional PCI and PCI-X. This means that all REQ# inputs are sampled and all GNT# outputs are registered, thus the earliest GNT# to a non parked master, is two cycles after REQ# is asserted.

The PCI arbiters implement a fixed Round Robin (RR) arbitration mechanism. The PCI arbiter performs a default parking on the last agent granted. To overcome problems that happen with some PCI devices that do not handle parking properly, use the PCI Arbiter Control register's `PD` bits [20:14] (see Table 383 on page 540) as an option to disable parking on a per PCI master basis.

Note

In addition to disabling parking to avoid issues with some problematic devices, disable parking on any unused request/grant pair. This avoids possible parking on non-existent PCI masters. For example, if only three external agents are connected to PCI_0 arbiter, then PD[6:4] must be set to '1'.

The PCI arbiters also implement Broken Master detection. A master that requests the bus must initiate a transaction (assert FRAME#) as soon as its GNT# is asserted. If the master is broken and does not initiate a PCI transaction, the PCI bus hangs permanently. To avoid this condition, the internal PCI arbiter implements a programmable Broken Value counter. If the master granted on the bus does not issue a transaction within the number of cycles specified in PCI Arbiter Control register `BV` bits [6:3], the arbiter de-assert GNT# from the broken master and grants the bus to some other master.

13.3 PCI Master Configuration Cycles in Conventional PCI Mode

The MV64360/1/2 translates CPU read and write cycles into configuration cycles using the PCI configuration mechanism #1 (per the PCI spec).

The MV64360/1/2 contains two registers to support configuration accesses: Configuration Address and Configuration Data. The mechanism for accessing configuration space is to write a value into the Configuration Address register that specifies the:

- PCI bus number
- Device number on the bus
- Function number within the device
- Configuration register within the device/function being accessed

A subsequent read or write to the Configuration Data register causes the MV64360/1/2 to translate that Configuration Address value to the requested cycle on the PCI bus.

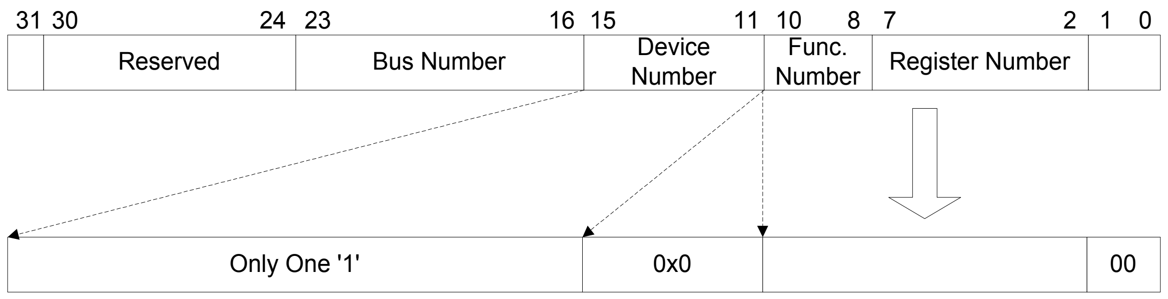
In conventional PCI mode, the PCI P2P Configuration register `BusNumber` bits [23:16] and `DevNumber` bits [28:24] (see Table 389 on page 544) defines the bus number to which the MV64360/1/2 PCI interface is connected and the MV64360/1/2 device number on this bus. These fields affect the type of configuration access the PCI master generates.

Note

A PCI transaction initiated by the peer PCI interface (P2P bridging) is only supported by the MV64360 and MV64361 devices.

If the `BusNumber` field in the Configuration Address register equals to the PCI P2P Configuration register Bus-Number field, but the DevNum fields do not match, a Type0 access is performed. This type of access addresses a device attached to the local PCI bus. The Configuration Address translation to the value driven on the AD bus during address phase is shown in Figure 34.

Figure 34: Conventional PCI Type 0 Configuration Transaction Address Translation



Driving only one ‘1’ on AD[31:16] allows easy generation of the IDSEL signal on board, by connecting each of these AD lines to the appropriate PCI slot IDSEL signal through a resistor.

Note

Driving only ‘1’ in MV64360/1/2 starts on AD[16], unlike the GT-64240/60 devices where it starts on AD[11].

Table 48 shows Device Number to IDSEL mapping.

Table 48: Device Number to IDSEL Mapping

Dev #	AD[31:16]
0x0	0000.0000.0000.0001
0x1	0000.0000.0000.0010
0x2	0000.0000.0000.0100
0x3	0000.0000.0000.1000
0x4	0000.0000.0001.0000
0x5	0000.0000.0010.0000
0x6	0000.0000.0100.0000
0x7	0000.0000.1000.0000

Table 48: Device Number to IDSEL Mapping (Continued)

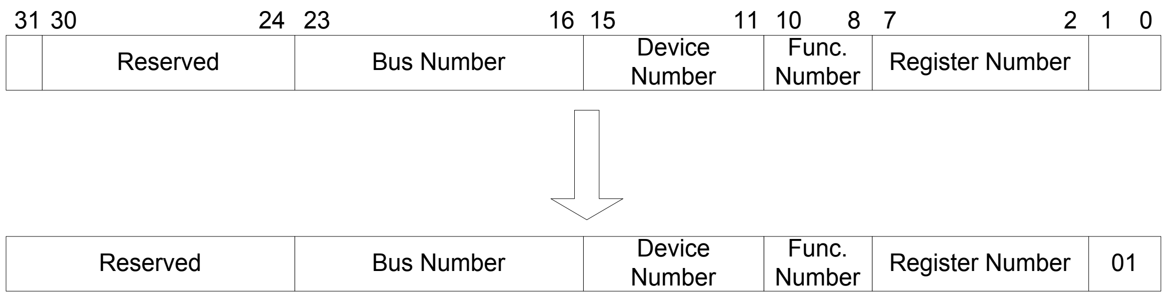
Dev #	AD[31:16]
0x8	0000.0001.0000.0000
0x9	0000.0010.0000.0000
0xA	0000.0100.0000.0000
0xB	0000.1000.0000.0000
0xC	0001.0000.0000.0000
0xD	0010.0000.0000.0000
0xE	0100.0000.0000.0000
0xF	1000.0000.0000.0000
0x10-0x1F	0000.0000.0000.0000

Note

The MV64360/1/2 performs address stepping for the PCI configuration cycles. Once granted on the bus, it drives a valid address and command on AD and C/BE# respectively one cycle before asserting FRAME#.

If the Configuration Address register's BusNum field does not match the PCI-X Status register's (for PCIX mode) or PCI P2P Configuration (for Conventional PCI) BusNum field, a Type1 access is performed. This access type addresses a device attached to a remote PCI bus. In this case, the Register Number, Function Number, Device Number, and Bus Number are copied directly from the Configuration Address register to the AD bus, as shown in Figure 35.

Figure 35: Conventional PCI Type 1 Configuration Transaction



A special cycle is generated if all of the following apply:

- The BusNum field in the Configuration Address register equals the PCI-X Status register's BusNum field.
- The DevNum field is 0x1f.
- The function number is 0x7.
- The register offset is 0x0.

The CPU accesses the MV64360/1/2's internal configuration registers when the DevNum and BusNum fields in the Configuration Address register match the corresponding fields in the PCI-X Status register.

Note

The configuration enable bit (`ConfigEn`) in the Configuration Address register must be set before the Configuration Data register is read or written. If this bit is not set, no transaction is driven on the PCI bus. In case of a write transaction, the data is lost. In case of a read transaction, a non-deterministic value is returned to the CPU.

13.4 PCI Target Address Decoding

For the different MV64360 and MV64361 devices, the PCI target interface supports 16 address windows. In the MV64362, the PCI target interface supports only 12 address windows, since there is only one PCI interface and no integrated SRAM.

- Four DRAM banks
- Five Device banks
- Integrated SRAM
- Two memory mapped P2P windows
- I/O mapped P2P window
- CPU (60x bus) window
- Memory mapped internal registers window
- I/O mapped internal register window

Each window is defined by the base and size registers. Each window (except of the internal registers windows) can decode up to 4 GB space.

All memory mapped BARs (Base Address Register) are 64-bit registers, supporting 64-bit addressing. If the upper 32-bit of the BAR is set to '0', the BAR acts as a 32-bit BAR and the MV64360/1/2 PCI slave only responds to a SAC transaction. If the BAR's upper 32-bit value is other than '0' (which means it allocates a window above the 4 GB address space), the slave responds only to DAC transactions.

The PCI slave responds to an address hit in the I/O BARs only if the configuration Command register's bit [0] (Target I/O Enable) is set to '1'. It responds to an address hit in any of the other BARs only if bit [1] of configuration Command register (Target Memory Enable) is set to '1'.

To disable a specific BAR space, the MV64360/1/2 includes a 16 -bit BAR Enable register - bit per BAR. Setting a bit to '1' disables the corresponding BAR. A disabled BAR is treated as a reserved register (read only 0). PCI access match to a disabled BAR is ignored and no DEVSEL# is asserted.

The PCI target interface also supports address remapping to any of the resources. This is especially useful when one needs to reallocate some PCI address window to a different location on memory.

Note

There are no size registers for the internal space BARs. The internal registers space has a fixed size of 64 KB.

The PCI specification defines that an I/O mapped BAR may not consume more than 256bytes of I/O space. This implies that the MV64360/1/2 I/O mapped BARs are not PCI compliant. By default, these BARs are disabled through Bar Enable register.

For full details on address decoding, see Section 7. “Address Space Decoding” on page 82.

13.5 PCI Access Protection

The PCI slave interface supports configurable access control. It is possible to define up to six address ranges to different configurations. Each region can be configured to:

- Write and access protection
- Byte swapping
- Cache coherency
- Maximum burst size
- Read prefetch

Three registers define each address window - Base (low and high) and size. An address received from the PCI, in addition to the address decoding and remapping process, is compared against the Access Control registers. This address comparison mechanism is the same one used by the address decoding logic (see Section 7. “Address Space Decoding” on page 82). If an address matches one of the access windows, the MV64360/1/2 handles the transaction according to transaction type and the attributes programed in the Access Control register, see Table 390 on page 544.

Each region contains two protection bits:

- Access protection
  Any PCI access to this region (read or write) is forbidden.
- Write protection
  Any PCI write access to this region is forbidden.

If an access violation occurs:

- The PCI slave interface terminates the transaction with Target Abort.
- The transaction address is latched in PCI Error Address register.
- The PCI `STAbort` bit in the interrupt cause register is set.

Note

The MV64360/1/2 internal registers space is not protected, even if the access protection windows contain this space.

The other attributes of the Access Control registers are discussed in 13.7 “PCI Target Operation in Conventional PCI Mode” on page 167.

13.6 P2P Configuration Transactions in Conventional PCI Mode

Note

The following P2P information only applies to the MV64360 and MV64361devices. The MV64362 has no P2P support.

The MV64360 and MV64361devices supports not only memory and I/O P2P transactions between the two PCI interfaces, but also propagation of configuration cycles.

Each PCI interface may respond to a type 1 configuration transaction, according to the settings of the PCI P2P Configuration register's `2ndBusL` bits [7:0] and `2ndBusH` bits [8:15] fields. These fields specify the buses resides on the other PCI interface. Upon detecting of PCI configuration type 1 transaction, the PCI target interface decodes the bus number driven on the `AD` bus (bits[23:16]). If the bus number is within the range of the other PCI interface (including the 2ndBusL and excluding the 2ndBusH boundaries), the transaction is propagated to the other PCI interface.

Note

By default, the `2ndBusL` field is greater than `2ndBusH`. This means that propagating a type 1 configuration transaction is disabled.

In case the type 1 configuration is claimed (DEVSEL# asserted), the transaction type driven by the PCI master is determined according to the device number, function number, and register offset driven on the AD bus (bits 15:11, 10:8 and 7:2 respectively), and according to the other PCI interface PCI P2P Configuration register `BusNumber` bits [23:16] and `DevNumber` bits [28:24] as follows:

1. If the received bus number is identical to the other PCI interface bus number, it converts the transaction to type 0.
2. If the received bus number differs from the other PCI interface bus number, it keeps the transaction as type 1.
3. If the received bus number is identical to the other PCI interface bus number, the device number is ‘11111, the function number is ‘111, and the register offset is 0x0. It drives a Special Cycle.

Note

Although the MV64360 and MV64361devices supports all types of P2P cycles, it is not P2P Bridge Specification compliant. It does not implement all required bridge configuration registers, nor keeps all P2P transactions ordering rules.

Unlike a P2P bridge that has a primary and secondary interfaces, in the MV64360/1/2 the P2P functionality is identical in both directions, and it supports propagation of type1 configuration access from any of the two PCI interfaces.

13.7 PCI Target Operation in Conventional PCI Mode

The MV64360/1/2 responds to the following PCI cycles as a target device:

- Memory Read
- Memory Write

- Memory Read Line
- Memory Read Multiple
- Memory Write and Invalidate
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write
- DAC Cycles

The MV64360/1/2 does not act as a target for Interrupt Acknowledge and Special cycles (these cycles are ignored). The MV64360/1/2 does not support Exclusive Accesses. It treats Locked transactions as regular transactions (it does not support LOCK# pin).

The slave consists of 512 bytes of posted write data buffer that can absorb up to four 128 byte write transactions (or a long burst write of 512bytes), and four read prefetch buffers, 128 bytes each, to support up to 4 delayed reads.

13.7.1 PCI Posted Write Operation

All PCI writes, except for configuration writes, are posted. Data is first written into the posted write buffer and later written to the target device.

The slave supports unlimited burst writes. The write logic separates the long PCI bursts to fixed length bursts towards the target device. Program the internal burst length to four, eight, or 16 64-bit words through PCI Access Control registers's `MBurst` bits. Whenever this burst limit is reached, the slave generates a write transaction toward the target device, while continuing to absorb incoming data from the PCI. The PCI burst writes have no wait states (TRDY# is never de-asserted). If the slave transaction queue is full, a new write transaction is retried (or if it becomes full during a long burst write, the burst is disconnected).

The write buffer capability to absorb the burst write transactions, depends on target device bandwidth and arbitration. For example, the MV64360/1/2 supports infinite burst write from PCI to DRAM, in case the DRAM controller is dedicated to serve accesses from PCI (not interfered with accesses from other agents).

The slave posting writes logic also aligns bursts that do not start on a 32/64/128-byte boundary, depending on the `MBurst` setting, for more efficient processing by the target units. For example, if `MBurst` is set to maximum bursts of eight 64-bit words, and a PCI long burst write transaction starts at address 0x18, the slave issues a write transaction of five 64-bit words to the target unit and continues with a new transaction to address 0x40.

Note

If the PCI address does not match any of the PCI Access Control registers address windows, the PCI slave acts as if `MBurst` is programmed to 32-bytes.

The PCI specification defines that I/O writes must not be treated as posted transactions. The slave does not meet this requirement.

13.7.2 PCI Non-posted Writes

PCI configuration writes are non-posted. The slave asserts TRDY# only when data is actually written to the configuration register. This implementation guarantees that there is never a race condition between the PCI transaction changing address mapping (Base Address registers) and the following transactions.

Note

A PCI write to any of the MV64360/1/2 internal registers, is treated as a posted write. This means that there might be a race condition, if setting these registers from PCI (e.g. changing some Size register, and then access the window that is defined by this window). In order to prevent these race conditions, it is recommended to follow the write transaction, with a read transaction to the same register, to guarantee that it's updated.

13.7.3 PCI Read Operation

Conventional PCI read access suffers from the following limitations:

- There is no way for the target device to know in advance the amount of data required, thus it needs to speculate how much data to prefetch from the target interface.
- Read accesses from high latency interface typically result in many dead cycles on the PCI bus. The PCI master is waiting for the read data to return. This results in low bus utilization.

The MV64360/1/2 PCI slave design, is targeted to solve these basic limitations by:

- User defined typical burst read size per address window. This allows the slave to prefetch the amount of data required by the initiating master.
- All reads are handled as delayed transactions. This enables better bus utilization, in case of multiple masters.

There are two fields in the PCI Access Control register (see Table 390 on page 544) that affects the slave read behavior - `Mburst` bits [9:8] and `RdSize` bits [11:10]. `Mburst` defines the maximum burst size the slave requests from the target interface. This parameter is mainly used for bandwidth and latency considerations on the target interface (see CrossBar section for more information). `RdSize` defines the typical amount of read data required. The slave prefetch data from the target interface is based on the `RdSize` setting. If for example, `Mburst` defines maximum burst of 64 bytes and `RdSize` defines typical burst of 256 bytes, the slave generates four 64 byte read transactions toward the target interface, to prefetch the 256 bytes.

There is one exception to the above prefetch scheme. This exception is a read with FRAME# assertion for a single cycle. This kind of transaction implies that the master initiating the transaction requires only a single data. In this case, the slave requests only a single data from the target interface.

Note

If the PCI address does not match any of PCI Access Control register's address windows, the slave acts as if both `Mburst` and `RdSize` are set to 32 bytes.

The slave supports up to four pending delayed reads. Upon receiving a read transaction, the slave issues a STOP# immediately (retry termination) but, internally, continues the transaction towards the target interface, prefetching the required amount of data as previously explained. When the data is received from the target, it is written to one of the four read buffers. When the data is ready in the read buffers, a retry of the original transaction results in data driven immediately on the PCI bus. Any attempt to retry the original transaction before the entire data amount is placed in the slave read buffers results in a STOP# issued by the slave.

If a PCI read transaction is still alive (implying that a longer burst is required) by the time all the burst data is driven on the PCI bus, the slave terminates the transaction with a disconnect.

The slave handles a queue of available free read buffers. With each incoming read transaction, the slave allocates a new read buffer. The read buffer is used storing the read data coming from the target interface. If all four read buffers are full when the slave receives a new read transaction, the incoming read transaction is terminated with RETRY.

To prevent dead locks due to "stuck" buffers (delayed reads that are never completed), the MV64360/1/2 supports a programmable Discard Timer register, see Table 381 on page 540. Each read buffer has its timer initialized to the Discard Timer value. Once a buffer is valid, the buffer timer starts counting down. If the buffer timer reaches '0' before being accessed (no delayed read completion), the buffer is invalidated. Setting the Discard Timer register to '0' prevents the slave from invalidating read buffers.

13.7.4 Non-Prefetchable Reads

The PCI specification allows a BAR space to be defined as non-prefetchable.

The PCI target device must guarantee that a read access to a non-prefetchable address space is not destructive (or as the PCI specification defines a prefetch from this memory space might cause "side effects"). An example of such memory space could be a FIFO device that speculative reads are destructive.

If a PCI read access matches a non-prefetchable BAR (bit[3] of the BAR is '0'), the MV64360/1/2 PCI slave treats this read access as a non-prefetchable read, regardless of the attributes defined in the PCI access registers. It treats it as a delayed read of a single data.

During non-prefetchable read transactions, the PCI slave requests a single 64-bit word from the target interface with the required byte enables, thus guarantees there is no destructive read. This is in contrast to prefetchable reads in which the MV64360/1/2 prefetches at least 4/8/16 64-bit words from the target interface, ignoring byte enables.

Upon the delayed read completion, if the initiating master requests more than a single data, the MV64360/1/2 PCI slave disconnects this burst attempt.

The PCI reads from the MV64360/1/2 internal and configuration registers are also treated as non-prefetchable reads.

13.7.5 PCI Target Termination

The MV64360/1/2 PCI slave supports the three types of target termination events described in the PCI specification – Target Abort, Retry, and Disconnect.

Target Abort is activated in the following cases:

- I/O transaction with address bits [1:0] is not consistent with byte enables.
- Address parity error.
- Violation of PCI access protection setting.
- Slave accessed with REQ64 and AD[2] is not zero.
- Slave accessed and the address match with two or more BAr (result of BAD programming of the BAR registers).

In any of these cases:

- The PCI slave interface terminates the transaction with Target Abort.
- The transaction address is latched in PCI Error Address register.
- The PCI STAbort bit in the interrupt cause register is set.

The slave generates a RETRY termination in the following cases:

- Delayed reads (first attempt, or completion attempt while read data is not ready yet).
- A new write transaction while write buffer is full.

- A new read transaction while read buffer is full.
- A sync barrier transaction while there is a pending unresolved previous sync barrier.
- Retry enable feature is active ("Initialization Retry")
- Non posted write (configuration write) while the write buffer is not empty.

The slave generates a DISCONNECT termination in the following cases:

- Burst access with start address bits[1:0] different than ‘00.
- Burst access that reaches BAR boundary.
- Write buffer becomes full during a burst write.
- Burst access to internal registers.
- Delayed read completion is not satisfied with the amount of data prefetched by the PCI slave.

Note

The MV64360/1/2 no longer supports the Timeout registers which are used by the GT-64240/60 devices for generating RETRY and DISCONNECT termination.

13.8 Initialization Retry

Most applications require CPU programing of some PCI configuration registers in advance of other bus masters accessing them. In a PC add-in card application, for example, the size registers must be set before the BIOS attempts to configure the card (the BIOS first scan all add in cards, and identify the address space requirement of each one, and only then allocates address range per each card). The MV64360/1/2 can be configured to Retry all PCI transactions until this configuration is complete. This prevents race conditions between the local processor and the BIOS.

If initialization Retry is enabled at reset, the PCI slave retries any transaction targeted to the MV64360/1/2’s space. The MV64360/1/2 remains in this retry mode until the CPU configuration register’s `StopRetry` bit is set. This mode is useful in all of the applications in which the local CPU programs the PCI configuration registers.

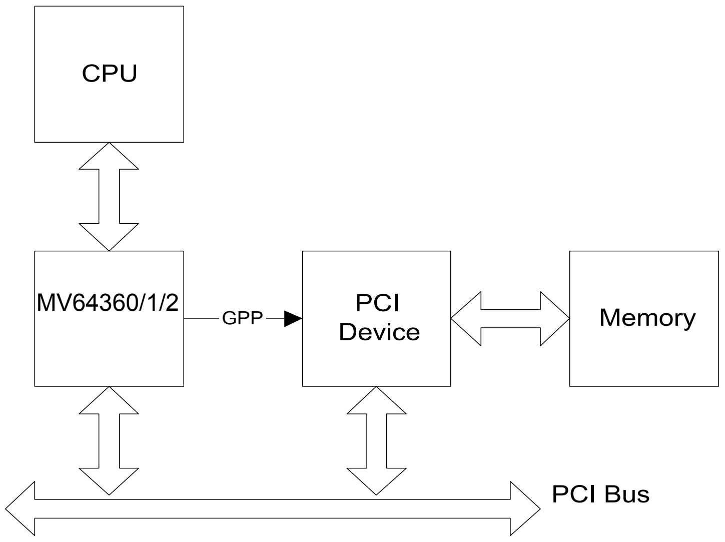
If serial ROM initialization is enabled, any PCI access to the MV64360/1/2 is terminated with Retry. This lasts until the end of the initialization.

13.9 Synchronization Barrier

The MV64360/1/2 supports a sync barrier mechanism. This mechanism is a hardware hook to help software synchronize between the CPU and PCI activities. The MV64360/1/2 supports sync barrier in both directions - CPU-to-PCI and PCI-to-CPU.

Figure 36 shows an example of the PCI sync barrier application.

Figure 36: PCI Sync Barrier Example



Assumes the CPU sends a packet to some PCI device and then notifies this device (via one of the GPP pins) that it has a packet waiting to handle. Since the packet may still reside in the MV64360/1/2 PCI master write buffer, the PCI device must first perform a sync barrier action, to make sure the packet is no longer in the MV64360/1/2 buffers.

The MV64360/1/2 PCI slave implements two registers - a PCI Sync Barrier Trigger register, and a PCI Sync Barrier Virtual register. A write to the PCI Sync Barrier Trigger register, triggers the sync barrier state machine. A subsequent read polling on the PCI Sync Barrier Virtual register will result in value of 0xFFFF.FFFF, until sync action completes. As soon as the PCI master write buffers are flushed, a read of the Virtual register will result in value of 0x0.

The MV64360/1/2 also supports nesting of sync barriers. The PCI agent that triggers the sync barrier, may decide to trigger a new sync barrier prior to the result of the previous one (new write to PCI Sync Barrier Trigger register). The MV64360/1/2 sync barrier implementation guarantees that a subsequent read polling on the PCI Sync Barrier Virtual register will result in a value of 0x0, only after the second sync barrier is resolved.

Note

This implementation is different than the one in the GT-64240/60 devices.

13.10Data Endianess

The MV64360/1/2 supports interfacing with Big Endian orientation CPU bus. Although the PCI specification defines the PCI bus only as Little Endian bus, the MV64360/1/2 also supports interfacing Big Endian PCI devices.

Endianess conversion is supported in both directions - access to PCI via the PCI master interface and access from PCI via the PCI slave interface. Both PCI master and slave supports byte and word swapping. The swapping is referred to a 64-bit words (as this is the MV64360/1/2 internal data path width). Table 49 shows an example of the data 0x0011223344556677.

Table 49: Data Swap Control

Swap Control	Swapping Granularity	Swapped Data							
00	Byte	77	66	55	44	33	22	11	00
01	Non	00	11	22	33	44	55	66	77
10	Byte and Word	33	22	11	00	77	66	55	44
11	Word	44	55	66	77	00	11	22	33

The right swapping setting depends on the PCI bus width (32/64) and endian orientation (big/little), as well as CPU bus endianess orientation, as shown in Table 50 and Table 51.

Table 50 summarizes the required swapping setting required for 32-bit PCI.

Table 50: 32-bit PCI Endianess Conversion

	CPU - big PCI - little	CPU - big PCI - big	CPU - little PCI - little	CPU - little PCI - big
Byte Swap	ON	OFF	OFF	ON
Word Swap	OFF	ON	OFF	ON

Table 51 summarizes the required swapping setting required for 64-bit PCI.

Table 51: 64-bit PCI Endianess Conversion

	CPU - big PCI - little	CPU - big PCI - big	CPU - little PCI - little	CPU - little PCI - big
Byte Swap	ON	OFF	OFF	ON
Word Swap	OFF	OFF	OFF	OFF

13.10.1PCI Slave Data Swapping

If all masters accessing the MV64360/1/2 PCI slave work in the same endianess orientation and with the same bus width, the user can use PCI Command register's `SByteSwap` bit [16] and `SWordSwap` bit [11] (see Table 378 on page 535) to set the appropriate endianess conversion. For cases of interfacing different PCI masters with different endianess orientation, the MV64360/1/2 also supports data swapping control on a per address window basis, via the Access Control register's `PCISwap` bits [7:6] (see Table 390 on page 544).

The MV64360/1/2 internal registers always maintain Little Endianess data. By default, it is assumed that data driven on the PCI bus is in Little Endian convention and there is no data swapping on PCI access to the internal registers. However, the MV64360/1/2 also supports data swapping on PCI accesses to internal registers via the PCI Command register's `SIntSwap` bits [24:25].

13.10.2PCI Master Data Swapping

If all PCI targets accessed by the MV64360/1/2 PCI master have the same endianess orientation and same bus width, use the PCI Command register's `MByteSwap` bit [0] and `MWordSwap` bit [10] to set the appropriate endianess conversion. However, for cases of interfacing different PCI targets with different endianess orientation, the MV64360/1/2 also supports data swapping control per address window basis, via the Base Address register's Swap bits, of each of the initiating interface (CPU Base Address Register bits[26:24], IDMA, Ethernet, and MPSCs).

13.1164-bit PCI Interface

Note

The following 64-bit interface information only applies to the MV64360 and MV64362 devices. The MV64361 only supports a 32-bit PCI interface.

The MV64360 and MV64362 devices support a 64-bit PCI interface. To operate as a 64-bit device, the REQ64# pin must be sampled LOW on RST# rise as required by PCI spec (Hold time of REQ64# in respect to RST# rise is 0).

When the devices are configured to 64-bit PCI, both master and target interfaces are configured to execute 64-bit transactions, whenever it is possible.

Note

The MV64360 and MV64362 device's PCI_0 interface supports CompactPCI Hot Swap Ready compliant, the 64EN pin is used to detect a 64-bit PCI bus rather than REQ64#. If not using CompactPCI, connect PCI_0 REQ64# to the 64EN pin.

If the third GbE port is used in the MV64360 (Table 117 on page 348), the PCI_1 interface is configured to 32-bit, regardless of REQ64 reset value.

13.11.1PCI Master 64-bit Interface

Note

The following 64-bit interface information only applies to the MV64360 and MV64362 devices. The MV64361 only supports a 32-bit PCI interface.

The PCI master interface always attempts to generate 64-bit transactions (asserts REQ64#), except for I/O or configuration transaction or when the required data is no greater than 64-bits. If the transaction target does not respond with ACK64#, the master completes the transaction as a 32-bit transaction.

The PCI master also avoids from generating a 64-bit transaction, if the requested address is not 64-bit aligned, and the M64Allign bit in PCI Command register is set cleared. For example the requested address is 0x4, the master issues a 64-bit transaction (assert REQ64#) with byte enables 0x0f. If the target does not respond with ACK64#, the transaction becomes a 32-bit transaction, with the first data phase driven with byte enable 0xF.

Although it is fully compliant with the PCI specification, some target devices do not tolerate this behavior. Use the M64Allign bit to prevent this problem.

When a PCI burst running in 64-bit mode is disconnected, and the amount of data the master needs to drive is not greater than 64-bit, it completes the disconnected transaction as a 32-bit master (does not assert REQ64#). This might be problematic when the target is a 64-bit Big Endian target. As described in section 13.10 “Data Endianess” on page 172, the byte swapping setting depends not only on the endianess nature of both initiator and target but also on the bus width. Changing bus width in the middle of a transaction targeted to a Big Endian device results in an incorrect data transfer.

If the targeted device on the PCI bus is a 64-bit device that ALWAYS responds with ACK64# to 64-bit transaction, the PCI master can be configured to always assert REQ64#, even if the amount of data needs to be transferred is less than or equal to 64-bit. Setting the initiating interface Base Address register’s PCIReq64 bit (), forces the PCI master to issue 64-bit transactions.

Note

Forcing REQ64# is allowed only when the target PCI device always responds with ACK64# to a 64-bit transactions. If the target device is not of that type and REQ64# is forced, a PCI violation occurs and the system might hang.

The PCI bus is defined as a Little Endian bus. Placing Big Endian devices on the bus is not compliant with the PCI specification. This feature of forcing REQ64# is implemented to support 64-bit Big Endian devices on the PCI bus. The hook of forcing REQ64# is not fully compliant with the PCI specification, and must be used carefully.

13.11.2PCI Slave 64-bit interface

Note

The following 64-bit interface information only applies to the MV64360 and MV64362 devices. The MV64361 only supports a 32-bit PCI interface.

The PCI target interface always responds with ACK64# to a 64-bit transaction, except for accesses to configuration space, internal registers, I_2O space, or I/O transaction.

13.1264-bit Addressing

Note

The following 64-bit interface information only applies to the MV64360 and MV64362 devices. The MV64361 only supports a 32-bit PCI interface.

Both PCI master and slave support 64-bit addressing cycles.

CPU, IDMA, Ethernet, and MPSCs includes upper 32-bit remap registers. This allows for 64-bit addressing when accessing the PCI through the MV64360 or MV64362 PCI master. If the master is accessed with an address higher than 4 GB (which means that the upper 32-bit address is not 0), the master initiates a DAC transaction. This means the transaction address phase takes two clock cycles.

On the first cycle, the master drives a ‘1101’ value on C/BE[3:0]# and the lower 32-bit address on AD[31:0]. On the next cycle it drives the required command on C/BE[3:0]# and the upper 32-bit address on AD[31:0].

If the PCI interface is configured to 64-bit bus, the master drives on the first cycle the required command on C/BE[7:4]# and the upper 32-bit address on AD[63:32]. This is useful when the target is also a 64-bit addressing capable device. In this case, the target starts address decoding on the first cycle, without waiting for the second address cycle.

On a DAC transaction, target address decode time is one cycle longer than in SAC transaction. Thus, the master issues a master abort on a DAC transaction only after five clock cycles, rather than four clocks in the case of SAC.

As a target, MV64360 and MV64362 devices respond to DAC transactions, only if the address matches one of it’s 64-bit BARs. In this case, the slave starts address decoding only after 2nd cycle (when the whole 64-bit address is available). This implies that DEVSEL# is asserted three clock cycles after FRAME# rather than two clocks in the case of SAC transaction.

Note

Unlike the GT-6424xx devices, the MV64360 and MV64362 do not have separate SAC and DAC BARs. All of its BARs are 64-bit wide, and support 32 or 64-bit addressing, depending on the setting of the upper 32-bit of the BAR.

13.13PCI Parity and Error Support

The MV64360/1/2 implements all parity features required by the PCI specification. This includes PAR, PERR#, and SERR# generation and checking, also PAR64 in case of 64-bit PCI configuration.

It also supports propagation of errors between the different interfaces. For example, a PCI read from SDRAM with ECC error detection may be configured to be driven on the PCI bus with bad PAR indication.

The PCI interface also supports other error conditions indications, such as access violation and illegal PCI bus behavior, see 13.5 “PCI Access Protection” on page 166 and 13.7.5 “PCI Target Termination” on page 170 for more details.

The PCI parity support is detailed in Section 19. “Address and Data Integrity” on page 313.

13.14Cache Coherency

The MV64360/1/2 supports PowerPC cache coherency. Any PCI access to the SDRAM or to the integrated SRAM may generate a snoop transaction on the CPU bus to maintain coherency with CPU caches.

Each of the six PCI Access Control windows can be defined to maintain, or not maintain, cache coherency.

For full details, see Section 20. “PowerPC Cache Coherency” on page 322.

13.15Configuration Space

The MV64360/1/2 PCI interface supports Type 00 configuration space header as defined in PCI specification. The MV64360/1/2 is a multi-function device. It supports functions 0 to 4 and the header is implemented in all of these five functions as shown in Figure 37. The configuration space is accessible from the CPU or PCI buses.

The MV64360/1/2 PCI slave responds to a type 0 PCI configuration transactions, if IDSEL is active and if the function number is between 0–4. The slave does not respond to configuration access to functions 5–7.

Many of functions 1–4 registers are aliased to function 0 registers. For example, access to Vendor ID register in function 1 actually accesses Vendor ID register of function 0.

Each of the two PCI interfaces implements the configuration header. Each PCI can also access the other PCI's configuration space, but with offset increment of 0x80. For example, the PCI_0 Vendor ID is accessed at offset 0x0 from PCI_0, but at offset 0x80 from PCI_1 bus. Or, the PCI_1 Vendor ID is accessed at offset 0x0 from PCI_1, but at offset 0x80 from PCI_0 bus. This is especially required for PC environment where BIOS expects to see the configuration header registers at specific offsets.

Note

Although the MV64360/1/2 supports P2P transactions, it does not contain the required P2P device configuration header and is not P2P spec compliant.

13.15.1Plug and Play Base Address Registers Sizing

Systems adhering to the plug and play configuration standard determine the size of a base address register's decode range by first writing 0xFFFF.FFFF to the BAR, then reading back the value contained in the BAR. Any bits that were unchanged (i.e. read back a zero) indicate that they cannot be set and are not part of the address comparison. With this information the size of the decode region can be determined.

The MV64360/1/2 responds to BAR sizing requests based on the values programmed into the Bank Size Registers. Whenever a BAR is being read, the returned data is the BAR's value masked by it's corresponding size register. For example, if CS[0] BAR is programed to 0x3FF0.0000 and CS[0] Size register is programed to 0x03FF.FFFF, PCI read of CS[0] BAR will result in data of 0x3C00.0000.

The Size registers can be loaded automatically after reset as part of the MV64360/1/2 serial ROM initialization, see Section 27. "Reset Configuration" on page 347 for more details.

Figure 37: PCI Configuration Space Header

Function 0 Header

Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Rev ID	08h
BIST	Header	Latency	Line Size	0Ch
SCS[0] BAR				10h 14h
SCS[1] BAR				18h 1Ch
Mem Mapped Internal BAR				20h 24h
Reserved				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM BAR				30h
Reserved			Cap. Ptr	34h
Reserved				38h
Max_Lat	Min_Gnt	Int. Pin	Int. Line	3Ch

Function 1 Header

▨		▨		00h
▨		▨		04h
▨			▨	08h
▨	▨	▨	▨	0Ch
SCS[2] BAR				10h 14h
SCS[3] BAR				18h 1Ch
Int SRAM BAR				20h 24h
Reserved				28h
▨		▨		2Ch
Reserved				30h
Reserved			Reserved	34h
Reserved				38h
▨	▨	▨	▨	3Ch

Function 2 Header

▨		▨		00h
▨		▨		04h
▨			▨	08h
▨	▨	▨	▨	0Ch
CS[0] BAR				10h 14h
CS[1] BAR				18h 1Ch
CS[2] BAR				20h 24h
Reserved				28h
▨		▨		2Ch
Reserved				30h
Reserved			Reserved	34h
Reserved				38h
▨	▨	▨	▨	3Ch

Function 3 Header

▨		▨		00h
▨		▨		04h
▨			▨	08h
▨	▨	▨	▨	0Ch
CS[3] BAR				10h 14h
BootCS BAR				18h 1Ch
CPU BAR				20h 24h
Reserved				28h
▨		▨		2Ch
Reserved				30h
Reserved			Reserved	34h
Reserved				38h
▨	▨	▨	▨	3Ch

Function 4 Header

▨		▨		00h
▨		▨		04h
▨			▨	08h
▨	▨	▨	▨	0Ch
P2P Mem0 BAR				10h 14h
P2P Mem1 BAR				18h 1Ch
P2P I/O BAR				20h
I/O Mapped Internal BAR				24h
Reserved				28h
▨		▨		2Ch
Reserved				30h
Reserved			Reserved	34h
Reserved				38h
▨	▨	▨	▨	3Ch

Reserved	Read Only 0
▨	Aliased to function 0 register

Notes

- In the MV64362, the P2P Memx Bars in Function 4 are reserved.
-

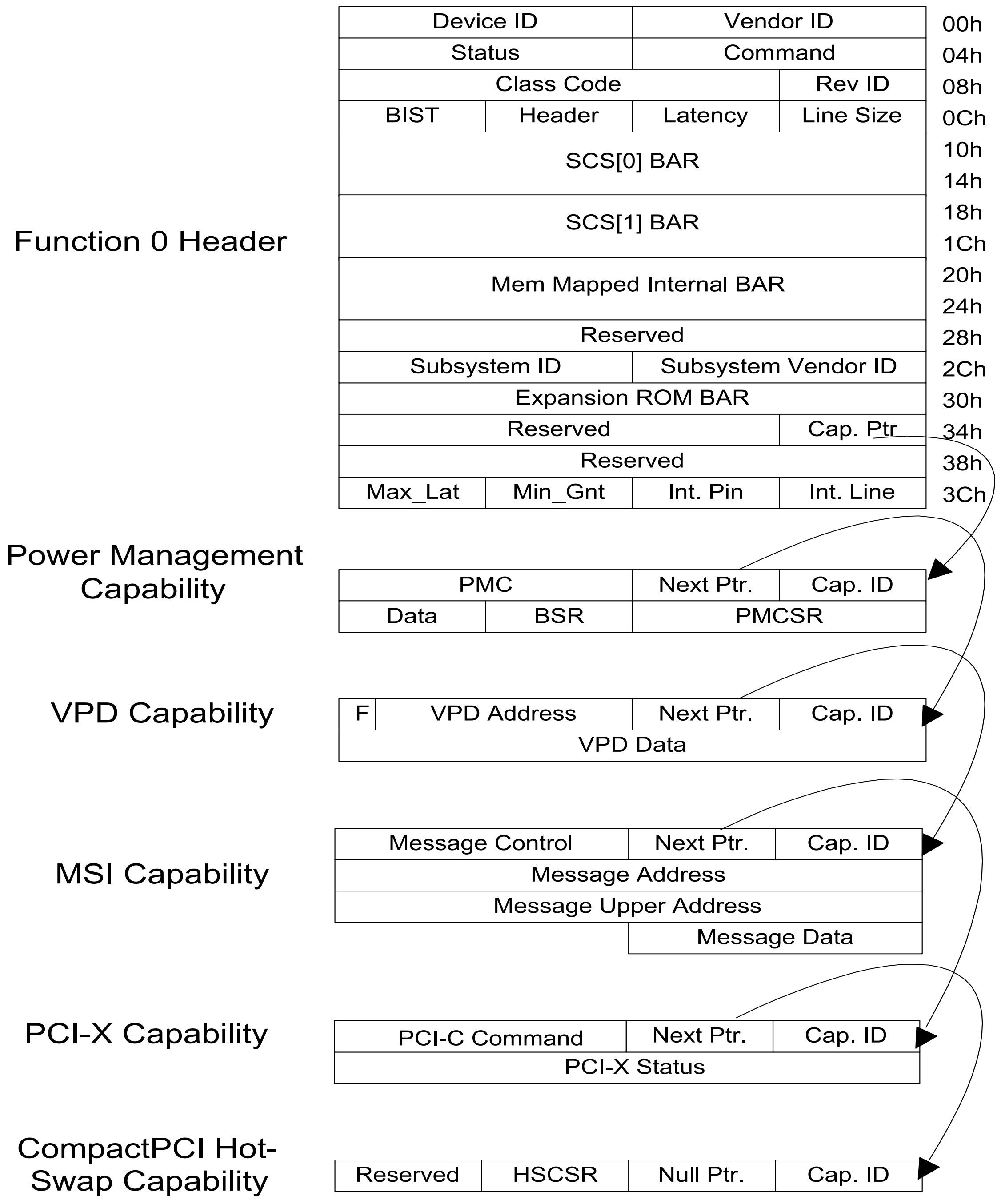
13.16PCI Special Features

The MV64360/1/2 supports the following special PCI features:

- Built In Self Test (BIST)
- Vital Product Data (VPD)
- Message Signaled Interrupt (MSI)
- Power Management
- CompactPCI Hot Swap

The VPD, MSI, PMG, and HotSwap features are configured through Capability List, as shown in Figure 38.

Figure 38: MV64360/1/2 Capability List



13.16.1Power Management

The MV64360/1/2 implements the required configuration registers defined by the PCI specification for supporting system Power Management as well as PME# pin. The registers are implemented on all PCI interfaces. This implementation is fully compliant with the specification.

Notes

- The required configuration registers for the PCI_1 interface are only valid for the MV64360 and MV64361 devices.
- For full details on system Power Management implementation, see the PCI specification.

The Power Management capability structure consists of the following fields:

- Capability structure ID. The ID of PMG capability is 0x1.
- Pointer to next capability structure.
- Power Management Capability.
- Power Management Status and Control.

Power Management registers are accessible from the CPU or PCI. Whenever PCI_0 or PCI_1 updates Power State bits (`PStat` bits[1:0] of Power Management Control and Status register Table 432 on page 561), the PCI Interrupt Cause register's PM interrupt bit is set and an interrupt to the CPU or PCI is generated, if not masked by interrupt mask registers.

PME# is an open drain output. When the CPU sets `PME_Status` bit to '1' in the PMCSR register, the MV64360/1/2 asserts PME#. It keeps asserting PME# as long as the bit is set, and the `PME_En` bit is set to '1' in the PMCSR register. The PCI clears the `PME_Status` by writing '1', causing the de-assertion of PME#.

PME0# and PME1# pins are multiplexed on the MV64360/1/2 MPP pins. If PME# support is required, first program the MPP pins to the appropriate configuration.

Note

The MV64360/1/2 does not support it's own power down. It only supports a software capability to power down the CPU or other on board devices.

13.16.2Vital Product Data (VPD)

VPD is information that uniquely identifies hardware elements of a system. VPD provides the system with information such as part number, serial number or any other information.

The PCI specification defines a method of accessing VPD. The MV64360/1/2 VPD implementation is fully compliant with the spec. For full details on the VPD's structure, see the PCI specification.

The VPD's capability structure consists of the following fields:

- Capability structure ID. The ID of VPD capability is 0x3.
- Pointer to next capability structure.
- VPD Address. The 15-bit address of the accessed VPD structure.
- Flag. Used to indicate data transfer between VPD Data register and memory.
- VPD Data. The 32-bit VPD data written to memory or read from memory.

The MV64360/1/2 supports a VPD located in DevCS[3]#. PCI access to this VPD results in access to DevCS[3]#. Although the PCI specification defines the address to be accessed, as the VPD Address field in the VPD capability list item (15-bit address), the MV64360/1/2 supports remapping of the 17 high bits by setting the PCI Address Decode Control register's `VPDHighAddr` bits [24:8] (Table 372 on page 531) to the required address.

For PCI VPD write, the PCI write VPD data first, then writes the VPD address with Flag bit set to '1'. As a response, the slave writes the VPD data to the VPD device to the required address and clears the Flag bit as soon as the write is done.

For a PCI VPD read, the PCI writes VPD address with the Flag bit set to '0'. As response, the slave reads the VPD device from the required address, places the data in the VPD data field, and sets the Flag bit to '1'. The VPD read is treated as a non-prefetched nor delayed read transaction.

13.16.3Message Signaled Interrupt (MSI)

The MSI feature enables a device to request an interrupt service without using interrupts. The device requests a service by writing a system specified message to a system specified address. The system software initializes the message destination and message during device configuration. The MV64360/1/2 MSI implementation is fully compliant with the PCI specification. It supports a single interrupt message.

The MSI capability structure consists of the following fields:

- Capability structure ID. The ID of MSI capability is 0x5.
- Pointer to next capability structure.
- Message Control.
- Message Address. 32-bit message low address.
- Message Upper Address. 32-bit message high address (in case 64-bit addressing is supported). (Only valid in the MV64360 and MV64362 devices.)
- Message data. 15-bit of message data.

Message Control word consists of the following fields:

- bit[0] - MSI Enable. If set to 1, MSI is enabled, and the MV64360/1/2 drives interrupt messages rather than asserting the PCI INT# pin.
- bits[3:1] - Multiple Message Capable. Defines the number of DIFFERENT MSI messages the MV64360/1/2 can drive.
- bits[6:4] - Multiple Message Enable. Defines the number of DIFFERENT MSI messages the system allocates for the MV64360/1/2.
- bit[7] - 64-bit address capable. Enables 64-bit addressing messages. (Only valid in the MV64360 and MV64362 devices.)

As soon as PCI enables MSI (set `MSIEn` bit [16]), MV64360/1/2 no longer asserts interrupts on the PCI bus. Instead, the PCI master drives a memory write transaction on the PCI bus, with address as specified in Message Address field and data as specified in the Message Data field.

If the Message Upper Address field is set to '0', the master drives a DWORD write, else it drives a DAC DWORD write.

Unlike the PCI INT#, a level sensitive interrupt that is active as long as there are active non-masked interrupts bits set, MSI is an edge like interrupt. However, to prevent the PCI interrupt handler from missing any new interrupt events, the MV64360/1/2 continues to drive new MSI messages as long as pending, non-masked interrupts exist.

The PCI Discard Timer register's Timer bit [15:0] (see Table 381 on page 540) defines the time gap (SysClk cycles) between sequential MSI requests. A timer starts counting with each new MSI request. If it reaches 0 and there is still a pending non-masked interrupt, a new MSI request is triggered. If the PCI interrupt handler clears one of the Interrupt Cause register bits, and there is still a pending interrupt, the MV64360/1/2 immediately issues a new MSI without waiting for the timeout to expire.

Setting the MSI Timeout register to ‘0’ disables the timer functionality (as if it was programed to infinity). In this case, the PCI interrupt handler must confirm that there are no interrupt event is missed.

Note

When programing the MSI Timeout register to a small value, the PCI master transaction queue is repeatedly filled with MSI requests. This prevents CPU or DMA access to the PCI until the PCI interrupt handler clears the interrupt cause bit(s).

13.16.4CompactPCI Hot Swap

The MV64360/1/2 is CompactPCI Hot-Swap ready compliant. It implements the required configuration registers defined by CompactPCI Hot-Swap specification as well as three required pins.

Note

CompactPCI Hot-Swap is only supported on PCI_0.

The CompactPCI Hot Swap capability structure consists of the following fields:

- Capability structure ID. The ID of HS capability is 0x6.
- Pointer to next capability structure.
- Hot Swap Status and Control.

Hot Swap Status and Control register (HS_CSR) is accessible from both CPU and PCI. This register bits give status of board insertion/extraction as defined in the spec. HS_CSR bits are:

- EIM - ENUM# Interrupt Mask. If set to ‘1’, the MV64360/1/2 won’t assert ENUM# interrupt.
- LOO - LED On/Off. If set to ‘1’ LED is on.
- REM - Removal. Indicates board is about to be extracted.
- INS - Insertion. Indicates board has been inserted.

The MV64360 and MV64362 support four Hot-Swap ready required pins. The MV64361 supports three Hot-Swap ready required pins.

- HS - Handle Switch input pin. Indicates insertion or extraction of board. A ‘0’ value indicates the handle is open.
- LED - LED control output pin. A ‘1’ value turns the on board LED on.
- ENUM# - open drain output. Asserted upon board insertion or extraction (if not masked by EIM bit).
- 64EN# - PCI 64-bit enable input. Replaces the REQ64# sample on reset de-assertion. (Only valid in the MV64360 and MV64362 devices.)

Note

If the MV64360 or MV64362 are not being used in a hot-swap board, REQ640# pin must be connected to 64EN#.

Board extraction consists of the following steps:

1. The operator opens board ejector handle. As a result, HS goes LOW, indicating board is about to be extracted.
2. As a result, the REM bit is set and the ENUM# pin is asserted, if not masked by EIM bit.
3. The System Hot Swap software detects ENUM# assertion. Checks the REM bits in all Hot-Swappable boards. Identifies the board about to be extracted and clears the REM bit (by writing a ‘1’ value).
4. The MV64360/1/2 acknowledges the system software by stop asserting the ENUM# pin.
5. The Hot Swap software might re-configure the rest of the boards, and when ready, it sets the LOO bit, indicating board is allowed to be removed.
6. As a result, MV64360/1/2 drive LED pin to 1, the on board LED is turned on indicating that the operator may remove the board.

Board insertion consists of the following steps:

1. Board is inserted. It is powered from Early Power and it’s reset is asserted from Local PCI Rst#. The on board LED is turned on by hardware (not as a result of LOO bit state).
2. Local PCI Rst# is de-asserted, causing LED to turn off, indicating that the operator may lock the ejector handle.
3. The operator locks the handle. As a result, HS goes HIGH, indicating board is inserted and locked.
4. As a result, INS bit is set and ENUM# is asserted, notifying Hot-Swap software that a board has been inserted.
5. System Hot Swap software detects ENUM# assertion, checks INS bits in all Hot-Swappable boards, identifies the inserted board and clears INS bit (by writing a value of 1).
6. MV64360/1/2 acknowledges system software by stop asserting ENUM# pin. Now software may re-configure all the boards.

Note

For full details on Hot-Swap process and board requirements, see the CompactPCI Hot-Swap specification.

To support HotSwap Ready requirements in the MV64360 and MV64362, the devices implement a 64EN# input pin. When hot inserting a board, REQ64# cannot be sampled with local reset de-assertion in order to identify 64-bit PCI bus, since REQ64# is an active signal on the bus. For this reason, the 64EN# signal is provided. The MV64360/1/2 samples this pin rather than the REQ64# on reset de-assertion (local reset) to determine whether it works in a 64-bit PCI environment.

In addition, the MV64360/1/2 supports the following hot swap device requirements:

- All PCI outputs floats when RST# is asserted.
- All MV64360/1/2 PCI state machines are kept in their idle state while RST# is asserted.
- The MV64360/1/2 PCI interface maintains it’s idle state until PCI bus is in an IDLE state. If reset is de-asserted in the middle of a PCI transaction, the PCI interface stays in it’s idle state until the PCI bus is back in idle.
- The MV64360/1/2 has no assumptions on clock behavior prior to it’s setup to the rising edge of RST#.
- The MV64360/1/2 is tolerant of the 1V pre-charge voltage during insertion.
- The MV64360/1/2 can be powered from Early VDD.

13.16.5BIST (Built In Self Test)

The MV64360/1/2 supports BIST functionality as defined by the PCI specification. It does not run its own self test. Instead, it enables the PCI to trigger CPU software self test.

The BIST Configuration register is located at offset 0xF of function 0 configuration header. It consists of the following fields:

- BIST Capable bit (bit[7]). If BIST is enabled through reset initialization, it is set to '1'. This bit is read only from the PCI.
- Start BIST bit (bit[6]). Set to '1' by the PCI to trigger CPU software self test. Cleared by the CPU upon test finish.
- Bits[5:4] - Reserved.
- Completion Code (bits[3:0]). Written by the self test software upon test finish. Any value other than '0' stands for test fail.

Upon PCI triggering of BIST (writing '1' to bit[6]), the CPU interrupt is asserted (if not masked) and the CPU interrupt handler must run the system self test. When the test is completed, the CPU software must clear bit[6] and write the completion code.

The PCI specification requires that BIST is completed in two seconds. It is the BIST software responsibility to meet this requirement. If bit[6] is not cleared by two seconds, the PCI BIOS may treat it as BIST failure.

Note

The MV64360/1/2 does not runs its own self test. The BIST register implementation is just a software hook for the CPU to run a system self test.

13.16.6Expansion ROM

With the Expansion ROM enabled through PCI Mode register (see Table 379 on page 538), the MV64360/1/2 configuration space includes an expansion ROM BAR at offset 0x30 of function0 configuration space as specified in the PCI specification. Like the other BARs, there are expansion ROM size and remap registers. Address decoding is done the same way as for the other devices. A hit in the expansion ROM BAR results in an access to DevCS[3]#.

Note

Expansion ROM size must not exceed DevCS[3]# size.

With the Expansion ROM disabled, the MV64360/1/2 does not support expansion ROM BAR, offset 0x30 in the configuration space is reserved.

If using expansion ROM, the register's `ExpRomEn` bit [0] (see Table 429 on page 561) must be set to '1'(via local processor or serial ROM initialization), prior to BIOS access to the MV64360/1/2. For the PCI slave to respond to a PCI address hit in the expansion ROM space, the system software must set the Status and Command register's `MEMEn` bit [1] to '1' (see Table 419 on page 555) and bit [0] of expansion ROM BAR to '1', as defined in PCI specification.

13.17PCI Timing Considerations

The AC spec of 66MHz conventional PCI requires a 6ns output delay for all PCI signals. To meet this requirement, the MV64360/1/2 PCI interface is using an internal DLL.

The PCI specification permits using a DLL only when interfacing a 66MHz PCI bus. The MV64360/1/2 samples M66EN pin on reset de-assertion to determine if it is connected to a 66MHz bus. If M66EN is sampled low, it means that it is interfacing with a 33MHz bus. The PCI interface DLL is bypassed and the MV64360/1/2 meets the AC requirements of a 33MHz PCI bus.

When configured to PCI-X, the MV64360/1/2 PCI interface uses the internal DLL regardless of M66EN indication.

Note

If PCI interface DLL is enabled, the MV64360/1/2 PCI slave ignores any access from the PCI (does not assert DEVSEL#) starting from reset de-assertion until the internal clock driven by the DLL is stable (1024 clock cycles). After the internal clock is stable, the slave continues waiting for the PCI bus to be in an idle state before responding to PCI transactions.

13.18PCI Pads

All PCI pads are 5V tolerant. More over, the pads integrate clamping diodes, to clap input signals to Vref value.

Notes

- Only when the chip is powered are the MV64360/1/2 PCI pads are 5V tolerant. When it is not, as in hot insertion), the pads can only tolerate up to 3.3V.
- The VREF pin (used by the clamping diodes) must always be the highest PCI bus voltage. In a hot insertion environment, where the PCI bus is powered before the MV64360/1/2, VREF must be isolated from the bus (or powered from early power). VREF must never be connected to 0V.

To improve signal integrity and board timing design, the CPU interface pads have a calibration mechanism to control pad drive and output impedance. Connect PCI_CAL pin to VDD via a resistor. The resistor size should be 36 ohms. After PCI reset de-assertion, the calibration logic tunes the PCI interface output drivers impedance.

13.19PCI-X Reset Configuration

The MV64360/1/2 samples FRAME#, IRDY#, DEVSEL#, STOP#, and TRDY# on the rising edge of PCI reset.

The PCI interface enters PCI-X mode when the following takes place:

- FRAME# and IRDY# are sampled High.
- Either DEVSEL#, STOP#, and TRDY# are sampled Low.

Note

According to the PCI spec, DEVSEL#, STOP#, and TRDY# must be pulled up. It expects the "central resource" to actively drive these signals to the desired value during PCI reset, and float them on PCI reset de-assertion.

The combination of DEVSEL#, STOP#, and TRDY# also defines the clock frequency range of the PCI-X bus, as shown in Table 52. The PCI mode is registered in the PCI Mode register's PciMode bits[5:4], Table 379 on page 538.

Table 52: PCI-X Initialization Pattern

DEVSEL#	STOP#	TRDY#	Mode	Frequency
High	High	High	Conventional PCI	0 - 66MHz
High	High	Low	PCI-X	50-66MHz
High	Low	High	PCI-X	66-100MHz
High	Low	Low	PCI-X	100-133MHz
Low	Don't care	Don't care	Reserved	Reserved

Notes

- In the MV64360 and MV64361 devices, the two PCI interfaces are independently configured.
- The MV64360/1/2 behaves the same for the three PCI-X frequencies ranges.
- In addition to the outlined requirements, it also necessary to strap DevAD[31:29] to the same configuration. For example, for PCI-X/66MHz, drive DEVSEL#, STOP#, and TRDY# to [1,0,1], connect DevAD[31:30] to a pull-up and DevAD[29] to a pull-down.

13.20PCI Master Operation in PCI-X Mode

When configured to PCI-X mode, the MV64360/1/2 PCI master generates PCI transactions in respond to requests coming from the CPU, IDMAs, Ethernet MACs, MPSCs or the peer PCI interface, very similar to conventional PCI mode. The main differences are:

- PCI-X bus protocol
- PCI-X commands
- PCI-X split response

PCI-X bus protocol is a registered one. Each output signal (e.g. FRAME#, IRDY#) is an output of a flip-flop and each input signal is sampled by a flip-flop (e.g. TRDY#,STOP#). Also, the PCI-X bus protocol introduces an attribute phase following the address phase. During the attribute phase, the PCI master drives some more relevant information on the transaction.

The MV64360/1/2 PCI master is fully compliant with the PCI-X bus protocol, and meets its AC timing requirements.

PCI-X bus protocol distinguishes between DWORD transactions (single 32-bit DWORD transfer) and burst transactions. The MV64360/1/2 PCI master supports all DWORDS transactions:

- Memory Read DWORD
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write

- Interrupt Acknowledge
- Special Cycle

The MV64360/1/2 PCI master supports the following burst transactions:

- Memory Write
- Memory Write Block
- Memory Read Block

A DAC transaction is carried out if the requested address is beyond 4 GB (address bits[63:32] are not ‘0’).

The PCI-X bus protocol supports split transactions. A target may terminate a transaction initiated by the master with Split Response termination. In this case, the target is expected to later initiate a Split Completion transaction. The MV64360/1/2 PCI master supports these transactions and responds to Split Completion transactions (see "PCI Master Write Operation" and "PCI Master Read Operation" for details).

PCI-X protocol defines the NS (No Snoop) attribute. The MV64360/1/2 PCI master drives this bit, according to the setting of NS attribute in the address decoding registers of the originating unit (e.g. CPU to PCI Mem0 Base Address register’s NS bit). This feature enables the user to specify the PCI address windows in which NS bit is set, thus preventing snoop penalty in the target device. Setting the PCI Command register’s MNS bit [14] to ’0’ (Table 378 on page 535) disables this feature and causes the master to always drive the NS attribute to ’0’.

13.20.1PCI Master Write Operation

The MV64360/1/2 PCI master generates a Memory Write or Memory Write Block command for write transactions targeted to the PCI memory space. If the size of write data is greater than 8 bytes, and all byte enables are active, the master issues a Memory Write Block. In other cases, the master uses Memory Write command (which is byte enable sensitive).

Write transactions coming from the initiating units (e.g. CPU interface unit) are placed in the master’s write buffer, the master arbitrates the PCI bus, and, when gain bus mastership, drives the transaction on the bus. This is very similar to the master behavior with a conventional PCI.

The maximum byte count per a single transaction from any unit to the PCI master is 128 bytes. In conventional PCI mode, the master is able to perform transaction combining. This allows for longer burst writes on the PCI bus and is especially useful for long DMA transfers to the PCI. The PCI-X bus protocol does not define combine write transaction, since the bus protocol requires that a master specifies the entire sequence’s byte count in the transaction’s attribute phase, and the MV64360/1/2 PCI master can not tell in advance the total amount of DMA data (this knowledge is only visible to the DMA engine), there for it cannot perform write combining, as in conventional PCI.

However, the MV64360/1/2 master may still generate burst writes of up to 512 bytes. If the PCI Command register’s MWrCom bit [4] (see Table 378 on page 535) is set, two consecutive write transaction might be combined, if the first one was not yet granted on the bus. If while waiting for bus mastership, the PCI master receives a new write request to an address that matches the first transaction’s address of data n+1, it combines the two transactions. If GNT# to the master is delayed long enough, it might combine up to 512 bytes.

Note

As soon as the master receives bus mastership, it drives the current pending write transactions on the bus.

In conventional PCI mode, the master is capable of generating fast back-to-back transactions. PCI-X bus protocol does not support this. The MV64360/1/2 PCI master can issue a new transaction in PCI-X mode as early as two cycles after the completion of a previous write transaction

13.20.2PCI Master Read Operation

The MV64360/1/2 PCI master uses Memory Read Block or Memory Read DWORD for read transactions targeted to the PCI memory space. If the required read data is up to 4 bytes and does not cross DWORD boundary, the master uses Memory Read DWORD command. It uses Memory Read Block in any other case.

Read requests that come from the initiating units are placed in the master's transaction queue. After it gains bus mastership, the master drives the transaction on the bus. This is very similar to the master behavior with conventional PCI.

The master supports read byte count of up to 128 bytes. The read data received from the PCI bus, is placed in the master's read buffer, and driven back to the initiating unit.

Note

The PCI master does not support read combining in PCI-X mode.

A PCI-X target may terminate a read transaction with a Split Response. In this case, it is expected to generate a Split Completion transaction, with the required read data. The MV64360/1/2 PCI master supports up to four split read transactions. The master allocates a 128-byte read buffer per each read transaction it issues. Once the target generates the Split Completion transaction, the master compares the Tag and the requester ID of the incoming transaction against the Tags and IDs of the outstanding pending reads. If there is a match, the master accepts the transaction, places the incoming data in the read buffer, and drives it back to the initiating unit.

For multiple outstanding read transactions, the MV64360/1/2 PCI master may return read data back to the initiating units out of order. For example, if a CPU initiates two reads to two different PCI target devices, and if both transactions are terminated with Split Response by the PCI targets, the Split Completion transactions may return out of order.

The maximum outstanding reads the master may issue is controlled through PCI-X Command register `MOST` field (bits [22:20], see Table 440 on page 565). Setting this field to a number less than '4' limits the MV64360/1/2 maximum number of outstanding reads.

For the MV64360/1/2 PCI master to respond to a Split Completion transaction (assert DEVSEL#) the transaction's bus number, device number, and function number attributes must match the MV64360/1/2 PCI interface bus number, device number, and function number as appear in, and the Tag attribute must match one of the master's outstanding pending transactions. If any of these conditions is not met, the MV64360/1/2 does not respond to the transaction.

To prevent dead-locks, the MV64360/1/2 implements a discard timer, see Table 381 on page 540. If the timer is enabled and the master read transaction is terminated with Split Response, it starts counting. If the timer reaches '0' before the Split Completion arrives, the master frees the pre-allocated read buffer. Setting the Discard Timer register to '0', disables the timer.

Note

If using the discard timer, use a large enough value so that the timer will only expire in real cases of a bus hang. If the split completion arrives after the timer expires, a fatal error can occur.

In case the byte count of a Split Completion transaction that is targeted to the PCI master, does not match the expected byte count, the master sets bit in the PCI Interrupt Cause register, and terminates the transaction with a Target Abort.

13.20.3PCI Master I/O and Configuration Transactions

The MV64360/1/2 PCI master generates an I/O read/write transaction in case the initiating unit accesses to the PCI I/O space. The PCI-X mode defines I/O transactions as a DWORD transaction (single 32-bit DWORD).

The PCI master will not generate an I/O transaction on the PCI bus if the initiating unit requests a read access to the PCI I/O space which is more than single DWORD or crosses DWORD boundary. The master returns non-deterministic data to the initiator. In case of a write transaction, the master accesses the bus with one data phase. In both cases, the PCI Interrupt Cause register's `MIllegal` bit [10] is set, and an interrupt is asserted if not masked, see Table 412 on page 552.

The MV64360/1/2 PCI master generates a Configuration Read/Write, Interrupt Acknowledge, and Special Cycle in the PCI-X mode, similar to conventional PCI. The only difference is the address stepping. In conventional PCI, the master drives the address of a configuration transaction one cycle prior to FRAME# assertion. In PCI-X mode, it drives the address for four cycles prior FRAME# assertion, as defined in the PCI-X specification.

A PCI target may terminate I/O Write, Configuration Write, and Interrupt Acknowledge transaction with a Split Response. In these cases, the MV64360/1/2 PCI master will not issue a new transaction to the PCI bus before receiving the Split Completion for the write transaction. This implementation guarantees that the write is actually completed on the target side before any new access to this target is generated (similar to the non posted writes concept of conventional PCI). Similar to pending outstanding reads, if the Discard Timer is enabled and it expires, the pending Split Completion is discarded and the master releases it's queue for new transactions.

13.20.4PCI Master Termination

The MV64360/1/2 PCI master terminates its own initiated transaction with Master Abort, if there is no target response (no DEVSEL#) within six clock cycles.

The Master when acts as a target, terminates a Split Completion transaction with a Target Abort in case it detects a parity error during the address or attribute phase. In any of these cases:

- The transaction address is latched in PCI Error Address register.
- The PCI `MCTabort` bit in the interrupt cause register is set.

A target may terminate a transaction with RETRY. Although PCI-X does not force masters to retry the transaction, the MV64360/1/2 PCI master always retries these transactions until the transaction is properly completed, or until Retry Counter reaches '0'.

The PCI-X mode restricts masters to disconnect long bursts on ADB (128 byte boundary). Since the MV64360/1/2 PCI master issues a burst write on the PCI bus, it only disconnects when the entire amount of data is placed in its write buffer. It never disconnects a burst transaction in the middle. However, if a master generates a burst write that crosses ADB (in case of combining), the target might disconnect on ADB. The master continues from the point the transaction was disconnected, updating the remaining byte count.

The MV64360/1/2 PCI master, when responding to a Split Completion transaction, is always capable of receiving the whole transaction burst. Thus, it never terminates with RETRY or DISCONNECT.

13.21PCI Target Operation in PCI-X Mode

When configured to PCI-X mode, the MV64360/1/2 PCI slave responds to PCI transactions initiated by external masters, when the transaction address matches the PCI slave address space defined by the PCI BARs. This is similar to conventional the PCI mode. There are two main differences in the slave behavior:

- PCI-X bus protocol
- PCI-X split transactions

The PCI-X bus protocol is registered. Each output signal (e.g. DEVSEL#, TRDY#) is an output of a flip-flop. Each input signal is sampled by a flip-flop (e.g. FRAME#,IRDY#).

Additionally, the PCI-X bus protocol introduces an attribute phase following the address phase. During the attribute phase, the PCI master drives some more relevant information on the transaction, particularly the transaction byte count. This is a basic change in compare to the conventional PCI, in which a target can only identify the end of a transaction by detecting the de-assertion of FRAME# for the last data phase.

The MV64360/1/2 PCI target is fully compliant with the PCI-X bus protocol, and meets its AC timing requirements.

The MV64360/1/2 responds to the following PCI-X cycles as a target device:

- Memory Read DWORD
- Memory Write
- Memory Read Block
- Memory Write Block
- Alias to Memory Read Block
- Alias to Memory Write Block
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write
- DAC Cycles

Note

As in conventional PCI mode, the MV64360/1/2 does not respond to Interrupt Acknowledge nor Special cycle.

PCI-X protocol defines the No Snoop (NS) attribute. The MV64360/1/2 PCI slave decodes this bit to decide if the transaction requires a snoop action. If the NS attribute is set, the transaction is treated as not requiring a snoop action, regardless of the PCI Access Control register’s Snoop attributes settings. This feature enables an external PCI master to access the memory, without suffering from the snoop penalty (see Section 20. “PowerPC Cache Coherency” on page 322 for more information about cache coherency support).

13.21.1PCI Target Write Operation

The MV64360/1/2 PCI slave treats Memory Write, Memory Write Block, and Alias to Memory Write Block commands the same. They are all treated as posted writes with the exception of access to the MV64360/1/2 configuration registers. Very similar to conventional PCI mode, write data from the PCI is placed in the slave write buffer and forwarded to the target interface.

PCI-X allows write bursts of up to 4 KB. The MV64360/1/2 PCI slave interface contains 512 bytes write buffer. However, as explained before, the slave drains the buffer as soon as it is filled with 32/64/128-bytes of data depending on the MBurst setting, see Table 390 on page 544. So, if the target interface to which the data is transferred is free to accept this long burst write, the slave can transfer the whole 4 KB of data. If the slave write buffer gets full, the slave disconnects on ADB (128-byte boundary).

13.21.2PCI Target Read Operation

The PCI slave treats all read transactions as split transactions. It terminates the incoming transaction with a Split Response, fetches the data from the target interface, and, when the data is available, generates a Split Completion transaction.

Unlike the conventional PCI mode in which the slave does not have advanced knowledge of the required byte count and must perform some speculative prefetch, in PCI-X mode, the initiator signals for the required byte count in the transaction's attribute phase. This means the slave knows exactly how much data to request from the target interface.

The PCI slave contains four 128bytes of read buffers and can handle up to four pending split read transactions.

When the slave receives a read transaction, it allocates the maximum read buffers it needs for the transaction. It also performs address alignment on ADB (128-byte) basis. If, for example, the slave is requested for 200 bytes from offset 0x20. The slave allocates two buffers. The first one for the first 96 bytes and the second one for the remaining 104 bytes.

The slave terminates the PCI transaction with a Split Response and issues multiple read requests to the target interface. The size of each of these requests depends on the MBurst setting (32/64/128 bytes). Read data returning from the target is placed in the slave read buffers. Then, the slave initiates a Split Completion transaction towards the original initiating master using the appropriate master's bus number, device number function number, and tag.

The PCI-X allows terminating a Split Completion only on ADB (128 byte boundary). To meet this requirement, the slave issues the Split Completion transaction only when there is enough data in it's read buffer. For example, a read of 200 bytes from offset 0x20 with MBurst set to 32 byte. Only after three reads of 32 bytes from the target interface, the first read buffer is ready for being transferred to the PCI.

In the case of long burst read request, the slave attempts to drive the whole burst as a single Split Completion transaction. To do so, the slave first fetches the maximum number of bytes from the target unit into its read buffers. Then, it starts the Split Completion. When it finishes driving the content of one read buffer to the PCI bus, it continues with the next buffer, and concurrently, fetches more data from the target unit into the read buffer that have just been free. Since the PCI-X specification does not require specifying the exact byte count, this kind of "combining" is enabled in the attribute phase of a Split Completion transaction.

The PCI slave supports four outstanding split read transactions. However, internally, it serves each one at a time. If, for example, the slave receives a read request of 1 KB followed by a read request of 128 bytes. First, it deals with the first read and uses all of it's read buffers space for this 1KB request. Once all 1 KB of data is prefetched from the target unit, the slave begins prefetching data for the second transaction, as soon as a read buffer gets freed.

The concept of split transactions (similar to the conventional PCI delayed transactions) is useful for having better bus utilization. While the target is busy with prefetching the read data, the bus can be used by other masters, or even by the same master to initiate additional transactions. However, if interfacing a single master, that is not capable of utilizing the bus, while waiting for the read data to return, it is better to make the read latency to this master as short as possible. If PCI Command register's SRdMode bit is set to '1', the MV64360/1/2 PCI slave drives the Split Completion transaction as soon as it has one read buffer available.

13.21.3Access to Internal and Configuration Registers

The slave treats write access to configuration registers as non posted writes, as in conventional PCI mode. Reads are treated as split reads like any other read. The slave never terminates a write transaction with Split Response.

As in conventional PCI mode, the slave only supports single 32-bit access to the MV64360/1/2 internal or configuration registers. The internal registers should only be accessed with a Memory Read DWORD command, or with burst commands (Memory Write, Memory Write Block or Memory Read Block) of a single DWORD (up to 4 bytes, which do not cross DWORD boundary). An attempt to perform burst write to the internal registers results in Single Data Phase Disconnect. An attempt to perform burst read to the internal registers results in Target Abort.

13.21.4PCI Target Termination

The PCI slave supports five types of terminations:

- Target Abort
- RETRY
- Split Response
- Single data phase Disconnect.
- Disconnect on ADB.

When running in the PCI-X mode, the slave generates Target Abort in the following fatal cases:

- DWORD transaction (I/O or memory) with address bits [1:0] not consistent with byte enables.
- Detection of parity error during address or attribute phase
- Burst read to internal registers.
- Violation of PCI access protection setting.
- Burst read request that crosses BAR boundary.

In any of these cases:

- The PCI slave interface terminates the transaction with Target Abort.
- The transaction address is latched in PCI Error Address register.
- The PCI STAbort bit in the interrupt cause register is set.

The PCI slave generates a RETRY termination in the following cases:

- Write transaction while write buffer is full.
- Read transaction while read buffer is full.
- Non-posted write (configuration write) when buffer is not empty.

The PCI slave generates a DISCONNECT termination on ADB (128-byte boundary) in the following cases:

- Burst write access that reaches BAR boundary
- Write buffer becomes full during a burst write

The slave generates a Single Data Phase Disconnect on a burst write attempt to internal registers, as explained in "Access to Internal and Configuration Registers".

As an initiator of a Split Completion transaction, the slave also disconnects on ADB, when it cannot satisfy the amount of read data required, as explained in "PCI Target Read Operation".

13.22P2P Bridging in PCI-X Mode

Note

The following P2P information only applies to the MV64360 and MV64361 devices. The MV64362 uses one PCI interface.

The MV64360 and MV64361 supports bridging between the two PCI interfaces. This support also applies to cases when one, or both, PCI interface is running in PCI-X mode. Table 53 through Table 55 summarizes the conversion of transactions from one PCI interface to the other.

Table 53: PCI-X to PCI-X Bridging

Originating Side	Destination Side	Comments
I/O Read	I/O Read	
I/O Write	I/O Write	
Configuration Read	Configuration Read	
Configuration Write	Configuration Write	
Memory Read DWORD	Memory Read DWORD	
Memory Read Block	Memory Read Block	
Memory Write	Memory Write Block	More than 8 bytes, and all byte enables are active.
	Memory Write	Otherwise
Memory Write Block	Memory Write Block	More than 8 bytes, and all byte enables are active.
	Memory Write	Otherwise

Table 54: PCI to PCI-X Bridging

Originating Side	Destination Side	Comments
I/O Read	I/O Read	
I/O Write	I/O Write	
Configuration Read	Configuration Read	
Configuration Write	Configuration Write	
Memory Read	Memory Read DWORD	Up to 4 bytes that do not cross DWORD boundary.
	Memory Read Block	Otherwise

Table 54: PCI to PCI-X Bridging (Continued)

Originating Side	Destination Side	Comments
Memory Read Line	Memory Read DWORD	Up to 4 bytes that do not cross DWORD boundary. **NOTE:** The PCI masters are not expected to use the Memory Read Line command for reading 4 bytes or less.
	Memory Read Block	Otherwise
Memory Read Multiple	Memory Read DWORD	Up to 4 bytes that do not cross DWORD boundary. **NOTE:** The PCI masters are not expected to use the Memory Read Line command for reading 4 bytes or less.
	Memory Read Block	Otherwise
Memory Write	Memory Write	Up to 8 bytes, or in case not all bye enables are active.
	Memory Write Block	More than 8 bytes, and all byte enables are active.
Memory Write and Invalidate	Memory Write	Up to 8 bytes, or in case not all bye enables are active. **NOTE:** In this case, a PCI master that is using Memory Write and Invalidate commands violates the PCI spec
	Memory Write Block	More than 8 bytes, and all byte enables are active.

Table 55: PCI-X to PCI Bridging

Originating Side	Destination Side	Comments
I/O Read	I/O Read	
I/O Write	I/O Write	
Configuration Read	Configuration Read	
Configuration Write	Configuration Write	
Memory Read DWORD	Memory Read	
Memory Read Block	Memory Read	Up to 8 bytes
	Memory Read Line	One or more cache line(s) aligned.
	Memory Read Multiple	Cross cache line (32 byte) boundary.

Table 55: PCI-X to PCI Bridging (Continued)

Originating Side	Destination Side	Comments
Memory Write	Memory Write and Invalidate	One or more cache line(s) aligned (all byte enables are active).
	Memory Write	Otherwise
Memory Write Block	Memory Write and Invalidate	One or more cache line(s) aligned.
	Memory Write	Otherwise

Note

Although supporting P2P transactions, MV64360 and MV64361 is not compliant with the P2P bridge specification.

- It does not support the PCI-X Bridge Capabilities List structure nor PCI bridge configuration header.
- It's P2P functionality is fully symmetric between the two PCI interfaces (no primary and secondary bus).
- It does not attempt to meet PCI bridge ordering rules.

13.23PCI Master Configuration Cycles in PCI-X Mode

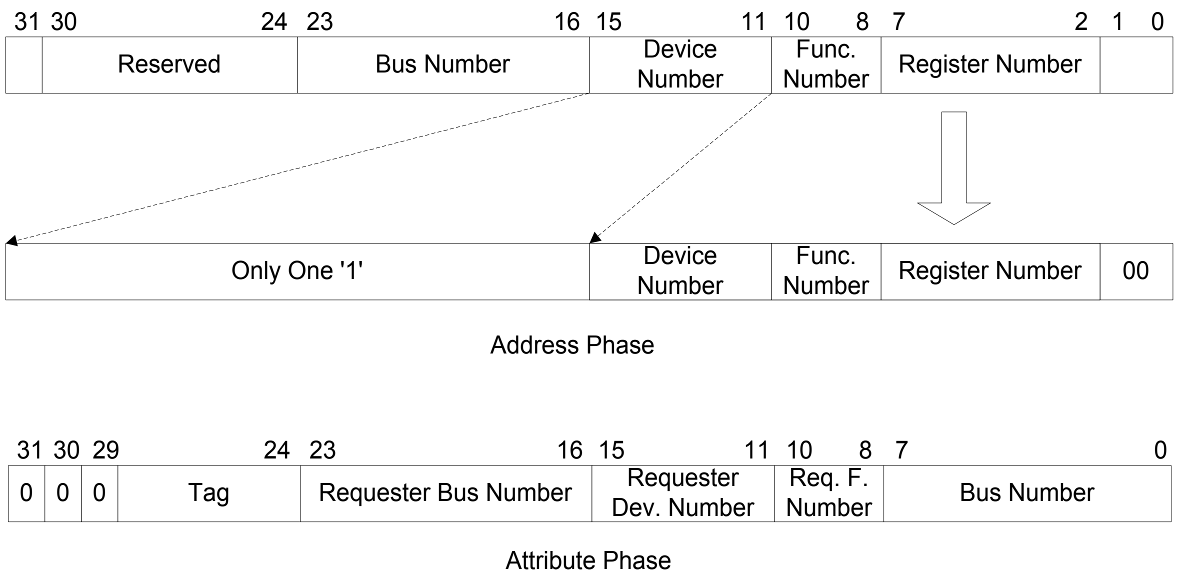
Generation of PCI configuration cycles in PCI-X mode is similar to the conventional PCI mode, using PCI Configuration Address and Configuration Data registers.

In PCI-X mode, PCI-X Status register's `BusNum` bits [7:3] and `DevNum` bits [15:8] (see Table 441 on page 566) defines bus number to which the MV64360/1/2 PCI interface is connected and the MV64360/1/2 device number on this bus. These fields affect the type of configuration access the PCI master generates.

If the BusNum field in the Configuration Address register equals the PCI-X Status register's BusNum field, but the DevNum fields do not match, a Type0 access is performed. This type of access addresses a device attached to the local PCI bus.

PCI-X specification requires some more information driven on the AD bus during the configuration cycle address and attribute phases, as shown in Figure 39.

Figure 39: PCI-X Type 0 Configuration Transaction Address Translation



Driving only one ‘1’ on AD[31:16] is similar to the conventional PCI implementation (13.3 “PCI Master Configuration Cycles in Conventional PCI Mode” on page 162).

If the Configuration Address register’s BusNum field does not match the PCI-X Status register’s, a Type1 access is performed. This access type addresses a device attached to a remote PCI bus. In this case, the Register Number, Function Number, Device Number, and Bus Number are copied directly from the Configuration Address register to the AD bus, as in the case of conventional PCI.

A special cycle is generated if all of the following apply:

- The BusNum field in the Configuration Address register equals the PCI-X Status register’s BusNum field.
- The DevNum field is 0x1f.
- The function number is 0x7.
- The register offset is 0x0.

The CPU accesses the MV64360/1/2’s internal configuration registers when the DevNum and BusNum fields in the Configuration Address register match the corresponding fields in the PCI-X Status register.

Note

The configuration enable bit (ConfigEn) in the Configuration Address register must be set before the Configuration Data register is read or written. An attempt by the CPU to access a configuration register without this bit set results in PCI master behavior as if it performed a master abort - no PCI transaction is driven on the bus, data of 0xFFFFFFFF is return in case of read transaction, and Mabort interrupt is set.

The PCI-X requires that every target device updates it’s own PCI-X Status register’s `Device Number` bits [7:3] and `Bus Number` bits [15:8] (see Table 441 on page 566), whenever accessed with a configuration type 0 cycle. For that reason, a master that initiates a type0 configuration transaction must drive these fields on the PCI AD bus during the address and attribute phases.

When the CPU generates a type0 configuration cycle, the PCI master drives the Requester Device Member, Requester Bus Number, and Requester Function Number on the AD bus, based on the corresponding PCI-X Status register's fields. In cases of type0 transactions, the Requester Bus Number and the Target Bus Number are the same because the MV64360/1/2 and the target PCI-X device sit on the same bus.

Note

Since the PCI master is driving its own device number during a type0 configuration transaction, an external master that is simultaneously driving a type0 configuration transaction to the MV64360/1/2 must not change the device number.

When the CPU generates a type1 configuration cycle, the Bus Number of the target device is driven on AD[23:16] during the address phase as in conventional PCI mode and AD[7:0] are reserved during the attribute phase.

When receiving a type1 configuration transaction, the transaction is transferred to the peer PCI interface, if the target bus number is within the range of the P2P Configuration register's 2ndBusL bits [7:0] and 2ndBusH bits [15:8] fields, as described in 13.6 “P2P Configuration Transactions in Conventional PCI Mode” on page 167.

Note

In PCI-X mode, the P2P Configuration register's BusNum bit [23:16] and DevNum bits [28:24] fields are a copy of the PCI-X Status and Command BN and DN fields. These fields can be used by the local CPU to identify the bus number and device number that were assigned to the MV64360/1/2.

13.24Parity in PCI-X Mode

When configured to PCI-X mode, the MV64360/1/2 PCI interface drives and checks parity, similar to the conventional PCI (see Section 19. “Address and Data Integrity” on page 313). As a master it drives even parity on PAR and PAR64 with address phase, attribute phase, and write data phase. As a slave it drives parity with Split Response termination, Split Completion address phase, attribute phase, and data phase. Upon detection of parity error, the MV64360/1/2 PCI interface asserts PERR#, and optionally SERR#. The following items differ from the conventional PCI mode:

- Attribute parity error cause SERR# (if enabled).
- PERR# is asserted one cycle later than in the conventional PCI.
- The slave does not check data parity while it is inserting initial wait states on a write transaction. The master does not check data parity while it is inserting initial wait states on a Split Completion transaction.
- During a read transaction, a target drives parity on clock N+1 for the read data it drove on clock N and the byte enables driven by the master on clock N-1. This rule applies only to the PCI master for checking data parity on DWORD reads it initiates. The PCI slave treats all reads as split transactions (it drives the read data as Split Completion transaction which is a “write type” of transaction).

The PCI-X specification defines that a device is required either to recover from data parity error (which means software involvement) or to assert SERR#. The PCI-X Command register's `DPERE` bit [16] (see Table 440 on page 565) defines this recovery ability. The MV64360/1/2 implements the PCI SERR# Mask register that defines all cases in which it asserts SERR#. If the local CPU software can handle recovery from parity errors, it needs to set PCI-X Command register's DPERE bit. Otherwise, it needs to enable SERR# assertion upon data parity errors via the PCI SERR# Mask register, see Table 411 on page 550.

By default, the PCI SERR# Mask register enables SERR# assertion in cases of parity error detection on address and attribute phase.

Note

For full details on parity generation and checking, refer to the PCI-X specification.

13.25PCI Interface Registers

Note

The PCI interface registers are located in Appendix E. “PCI Interface Registers” on page 513.

For the MV64360 and MV64361 devices, the same set of registers are duplicated for both PCI_0 and PCI_1. The only difference is that PCI_0 and PCI_1 registers are located at different offsets.

For the MV64362, only the register offsets for PCI_0 are valid.

The PCI_1 interface contains the same set of INTERNAL registers as PCI_0 interface. However, unless specified otherwise, the PCI_1 registers offsets are PCI_0 registers offsets + 0x080. For example, the PCI_0 CS[0] Size register is located at offset 0xC08. The PCI_1 CS[0] Size register is located at offset 0xC88.

All PCI CONFIGURATION registers are located at their standard offset in the configuration header, as defined in the PCI spec, when accessed from their corresponding PCI bus. For example, if a master on PCI_0 performs a PCI configuration cycle on PCI’s Status and Command Register, the register is located at 0x004. Likewise, if a master on PCI_1 performs a PCI configuration cycle on PCI_1’s Status and Command Register, the register is located at 0x004.

On the other hand, if a master on PCI_0 performs a PCI configuration cycle on PCI_1’s Status and Command Register, the register is located at 0x084. Likewise, if a master on PCI_1 performs a PCI configuration cycle on PCI’s Status and Command Register, the register is located at 0x084.

A CPU access to the MV64360/1/2’s PCI_0 configuration registers is performed via the PCI_0 Configuration Address and PCI_0 Configuration Data registers (internal registers offset 0xcf8 and 0xcfc respectively). A CPU access to the MV64360/1/2’s PCI_1 configuration registers is performed via the PCI_1 Configuration Address and PCI_1 Configuration Data registers (internal registers offset 0xc78 and 0xc7c respectively).

Note

A write to unmapped offsets of the MV64360/1/2’s internal registers space may result in a destructive write to existing PCI interface registers. The following is a list of these cases:

- Write to offset 0x0f3c results in a write to register at offset 0xd3c.
- Write to offset 0x0e04 results in a write to register at offset 0xc04.
- Write to offset 0x0e38 results in a write to register at offset 0xc38.
- Write to offset 0x1f04 results in a write to register at offset 0x1d04.
- Write to offset 0x1f14 results in a write to register at offset 0x1d14.
- Write to offset 0x0ef8 results in a write to register at offset 0xcf8.
- Write to offset 0x0e28 results in a write to register at offset 0xc28.
- Write to offset 0x1f5c results in a write to register at offset 0x1d5c.
- Write to offset 0x1f08 results in a write to register at offset 0x1d08.
- Write to offset 0x1f0c results in a write to register at offset 0x1d0c.

Section 14. Messaging Unit

The MV64360/1/2 messaging unit includes hardware hooks for message transfers between PCI devices and the CPU. This includes all of the registers required for implementing the I_2O messaging, as defined in the Intelligent I/O (I_2O) Standard specification.

The I_2O hardware support found in the MV64360/1/2 also provides designers of non-I_2O embedded systems with important benefits. For example, the circular queue support in the Messaging Unit provides a simple, yet powerful, mechanism for passing queued messages between intelligent agents on a PCI bus. Even the simple message and doorbell registers can improve the efficiency of communication between agents on the PCI.

The I_2O specification defines a standard mechanism for passing messages between a host processor (a Pentium, for example) and intelligent I/O processors (a networking card based on the MV64360/1/2 and a PowerPC processor, for example.) This same message passing mechanism may be used to pass messages between peers in a system.

The MV64360/1/2 Messaging Unit is implemented in both PCI interfaces. It allows for messaging between the CPU and PCI and inter-PCI interfaces messaging.

The MV64360/1/2 Messaging Unit registers are accessible from the PCI through the MV64360/1/2 internal space, as any other internal register. Setting the PCI Address Decode Control register's `MsgACC` bit [3] to '0' (Table 372 on page 531) enables access to these registers through the lower 4KB of CS[0] BAR space as well.

Note

To support PCI accesses to I_2O queues in memory, PCI Address Control register's `MsgACC` field must be set to '0'. This means that the first 4 KB of CS[0] BAR space are used as Messaging Unit internal registers space, instead of DRAM space.

14.1 Message Registers

The MV64360/1/2 uses the message registers to send and receive short messages over the PCI bus, without transferring data into local memory. When written to, the message registers may cause an interrupt to be generated either to the CPU or to the PCI bus. There are two types of message registers:

- Outbound messages sent by the MV64360/1/2's local CPU and received by an external PCI agent.
- Inbound messages sent by an external PCI bus agent and received by the MV64360/1/2's local CPU.

The interrupt status for outbound messages is recorded in the Outbound Interrupt Cause Register.

Interrupt status for inbound messages is recorded in the Inbound Interrupt Cause Register.

14.1.1 Outbound Messages

There are two Outbound Message Registers (OMRs).

When an OMR is written from the CPU side, a maskable interrupt request is generated in the Outbound Interrupt Status Register (OISR). If this request is unmasked, an interrupt request is issued on the PCI bus. The interrupt is cleared when an external PCI agent writes a value of '1' to the Outbound Message Interrupt bit in the OISR. The interrupt may be masked through the mask bits in the Outbound Interrupt Mask Register.

Note

An OMR can be written by the CPU or by the other PCI interface. It allows passing messages between CPU and PCI and between the two PCI interfaces.

14.1.2 Inbound Messages

There are two Inbound Message Registers (IMRs).

When an IMR is written from the PCI side, a maskable interrupt request is generated in the Inbound Interrupt Status Register (IISR). If this request is unmasked, an interrupt is issued to the CPU. The interrupt is cleared when the CPU writes a value of '1' to the Inbound Message Interrupt bit in the IISR. The interrupt may be masked through the mask bits in the Inbound Interrupt Mask Register.

Note

An inbound message sent from PCI bus can be targeted to the CPU or to the other PCI interface. The destination depends on the software setting of the interrupt mask registers, see 25.1.2 "Interrupts Mask Registers" on page 342.

14.2 Doorbell Registers

The MV64360/1/2 uses the doorbell registers to request interrupts on both the PCI and CPU buses. There are two types of doorbell registers:

- Outbound doorbells are set by the MV64360/1/2's local CPU to request an interrupt service on the PCI bus.
- Inbound doorbells are set by an external PCI agent to request interrupt service from the local CPU.

14.2.1 Outbound Doorbells

The local processor can generate an interrupt request to the PCI bus by setting bits in the Outbound Doorbell Register (ODR). The interrupt may be masked in the OIMR register. However, masking the interrupt does not prevent the corresponding bit from being set in the ODR.

External PCI agents clear the interrupt by setting bits in the ODR (writing a '1').

Note

The CPU or the other PCI interface can set the ODR bits. This allows for passing interrupt requests not only between CPU and PCI, but also between the two PCI interfaces.

14.2.2 Inbound Doorbells

The PCI bus can generate an interrupt request to the local processor by setting bits in the Inbound Doorbell Register (IDR). The interrupt may be masked in the IIMR register. However, masking the interrupt does not prevent the corresponding bit from being set in the IDR.

The CPU clears the interrupt by setting bits in the IDR (writing a '1').

Note

The interrupt request triggered from the PCI bus can be targeted to the CPU or to the other PCI interface, depending on software setting of the interrupt mask registers.

14.3 Circular Queues

The circular queues form the heart of the I_2O message passing mechanism and are the most powerful part of the messaging unit built into the MV64360/1/2. There are two inbound and two outbound circular queues in the Messaging Unit (MU).

Note

Whenever a reference is made to messages coming to or from the CPU, it also applies to messages coming to or from the other PCI interface.

14.3.1 Inbound Message Queues

The two inbound message queues are:

- Inbound Post
  Messages from other PCI agents that the CPU must process.
- Inbound Free
  Messages from the CPU to other PCI agent in response to an incoming message.

The two inbound message queues allow external PCI agents to post inbound messages to the local CPU in one queue and receive free messages (no longer in use) returning from the local CPU. The process is as follows:

1. An external PCI agent posts an inbound message.
2. The CPU receives and processes the message.
3. When the processing is complete, the CPU places the message back into the inbound free queue so that it may be reused.

Note

The two inbound queues are accessed from PCI, via fixed offset 0x40 within CS[0]# BAR space (named Inbound Port) - PCI read from Inbound port results in read from DRAM Inbound Free Queue, write to the Inbound port results in write to DRAM Inbound Post Queue

14.3.2 Outbound Message Queues

The two outbound message queues are:

- Outbound Post
  Messages from the CPU to other PCI agents to process.
- Outbound Free
  Messages from other PCI agents to the CPU in response to an outgoing message.

The two outbound queues allow the CPU to post outbound messages for external PCI agents in one queue and receive free messages (no longer in use) returning from other external PCI agents. The process is as follows:

1. The CPU posts an outbound message.
2. The external PCI agent receives and processes the message.
3. When the processing is complete, the external PCI agent places the message back into the outbound free queue so that it may be reused.

Note

The two outbound queues are accessed from PCI, via fixed offset 0x44 within CS[0]# BAR space (named Outbound Port) - PCI read from Outbound port results in read from DRAM Outbound Post Queue, write to the Outbound port results in write to DRAM Outbound Free Queue

14.4 Circular Queues Data Storage

Data storage for the circular queues must be allocated in local memory. It can be placed in any of CS[3:0] BARs address ranges, depending on the setting of `CirQDev` bits in Queue Control register. The base address for the queues is set in the Queue Base Address Register (QBAR). Each queue entry is a 32-bit data value. The circular queue sizes range from 4K entries (16 KBs) to 64K entries (256 KBs) yielding a total local memory allotment of 64 KBs to 1 MB. All four queues must be the same size and be contiguous in the memory space. Queue size is set in the Queue Control Register.

The starting address of each queue is based on the QBAR address and the size of the queues as shown in Table 56.

Table 56: Circular Queue Starting Addresses

Queue	Starting Address
Inbound Free	QBAR
Inbound Post	QBAR + Queue Size
Outbound Post	QBAR + 2*Queue Size
Outbound Free	QBAR + 3*Queue Size

Each queue has a head pointer and a tail pointer which are kept in the MV64360/1/2 internal registers. These pointers are offsets from the QBAR. Writes to a queue occur at the head of the queue.Reads occur from the tail. The head and tail pointers are incremented by either the CPU software or messaging unit hardware. The pointers wrap around to the first address of a queue when they reach the queue size.

Note

PCI read/write from a queue is always a single 32-bit word. An attempt to burst from an I_2O queue results in disconnect after the first data transfer. Additionally, the MV64360/1/2 never responds with ACK64# to an attempt to access the queue with a 64-bit transaction.

14.4.1 Inbound/Outbound Queue Port Function

Circular queues are accessed by external PCI agents through the Inbound and Outbound Queue Port virtual registers.

Note

With circular queues, you are not reading/writing a physical register within the MV64360/1/2. Instead, you are reading and writing pointers into the circular queues (located in SDRAM or Device) controlled by the MV64360/1/2. Refer to Figure 40 as you read the following sections.

When an Inbound Queue Port (IQP) is written from the PCI, the written data is placed on the Inbound Post Queue; it is posting the message to the local CPU.

When the Inbound Post Queue is written to alert the CPU that a message needs processing, an interrupt is generated to the CPU.

When this register is read from the PCI side, it is returning a free message from the tail of Inbound Free Queue.

The Outbound Queue Port (OQP) returns data from the tail of the Outbound Post Queue when read from the PCI side; it is returning the next message requiring service by the external PCI agent. When this register is written from the PCI, the data for the write is placed on the Outbound Free Queue; thus returning a free message for reuse by the local CPU.

Figure 40: I_2O Circular Queue Operation

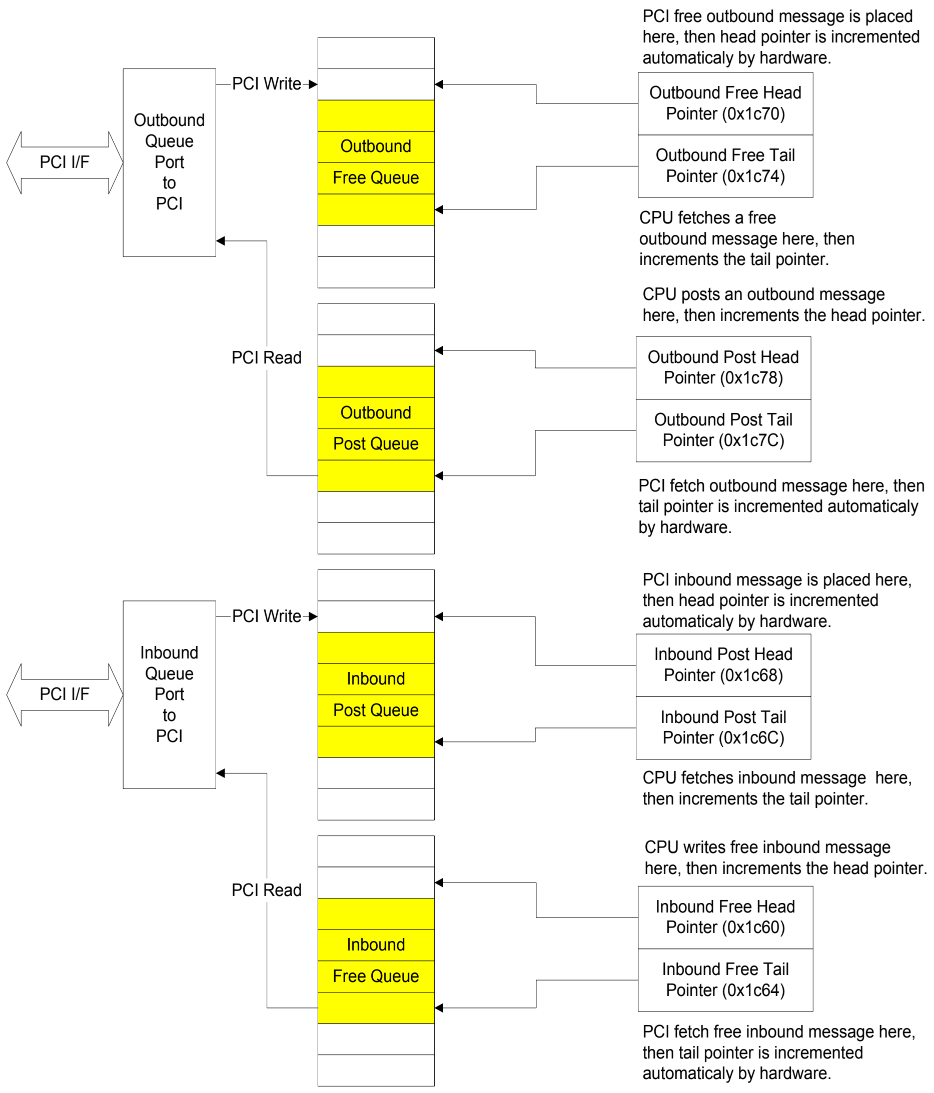

PCI free outbound message is placed here, then head pointer is incremented automaticaly by hardware.
PCI Write
Outbound Queue Port to PCI
PCI I/F
Outbound Free Queue
Outbound Free Head Pointer (0x1c70)
Outbound Free Tail Pointer (0x1c74)
CPU fetches a free outbound message here, then increments the tail pointer.
CPU posts an outbound message here, then increments the head pointer.
PCI Read
Outbound Post Queue
Outbound Post Head Pointer (0x1c78)
Outbound Post Tail Pointer (0x1c7C)
PCI fetch outbound message here, then tail pointer is incremented automaticaly by hardware.
PCI inbound message is placed here, then head pointer is incremented automaticaly by hardware.
PCI Write
Inbound Queue Port to PCI
PCI I/F
Inbound Post Queue
Inbound Post Head Pointer (0x1c68)
Inbound Post Tail Pointer (0x1c6C)
CPU fetches inbound message here, then increments the tail pointer.
CPU writes free inbound message here, then increments the head pointer.
PCI Read
Inbound Free Queue
Inbound Free Head Pointer (0x1c60)
Inbound Free Tail Pointer (0x1c64)
PCI fetch free inbound message here, then tail pointer is incremented automaticaly by hardware.

Table 57: I_2O Circular Queue Functional Summary

Queue Name	PCI Port	Generate PCI Interrupt?	Generate CPU Interrupt?	Head Pointer maintained by...	Tail Pointer maintained by...
Inbound Post	Inbound Queue Port	No	Yes, when queue is written	MV64360/1/2	CPU
Inbound Free		Yes, when queue is full.	No	CPU	MV64360/1/2
Outbound Post	Outbound Queue Port	Yes, when queue is not empty.	No	CPU	MV64360/1/2
Outbound Free		No	Yes, when queue is full	MV64360/1/2	CPU

14.4.2 Inbound Post Queue

The Inbound Post Queue holds posted messages from external PCI agents to the CPU.

The CPU fetches the next message process from the queue tail; external agents post new messages to the queue head. The tail pointer is maintained by the CPU. The head pointer is maintained automatically by the MV64360/1/2 upon posting of a new inbound message.

PCI writes to the Inbound Queue Port are passed to a local memory location at QBAR + Inbound Post Head Pointer. After this write completes, the MV64360/1/2 increments the Inbound Post Head Pointer by 4 bytes (1 word); it now points to the next available slot for a new inbound message. An interrupt is also sent to the CPU to indicate the presence of a new message pointer.

From the time the PCI write ends till the data is actually written to SDRAM or Device, any new write to the Inbound port results in RETRY. If the queue is full, a new PCI write to the queue results in RETRY.

Inbound messages are fetched by the CPU by reading the contents of the address pointed to by the Inbound Post Tail Pointer. It is the CPUs responsibility to increment the tail pointer to point to the next unread message.

14.4.3 Inbound Free Queue

The Inbound Free Queue holds available inbound free messages for external PCI agents to use.

The CPU places free message at the queue head; external agents fetch free messages from the queue tail. The head pointer is maintained in software by the CPU. The tail pointer is maintained automatically by the MV64360/1/2 upon a PCI agent fetching a new inbound free message.

PCI reads from the Inbound Queue Port return the data in the local memory location at QBAR + Inbound Free Tail Pointer. The following conditions apply:

- If the Inbound Free Queue is not empty (as indicated by Head Pointer not equal to Tail Pointer), the data pointed to by QBAR + Inbound Free Tail Pointer is returned.
- If the queue is empty (Head Pointer equals Tail Pointer), the value 0xFFFF.FFFF is returned. Indicating that there are no Inbound Message slots available. This is an error condition.

The processor places free message buffers in the Inbound Free Queue by writing the message to the location pointed to by the head pointer. It is the processor's responsibility to then increment the head pointer.

Note

It is the CPU's responsibility to make sure that the PCI agent keeps up the pace of the free messages and avoids pushing a new free message to the queue if it is full. There is no overflow indication when the Inbound Free Queue is full.

14.4.4 Outbound Post Queue

The Outbound Post Queue holds outbound posted messages from the CPU to external PCI agents.

The CPU places outbound messages at the queue head; external agents fetch the posted messages from the queue tail. The Outbound Post Tail Pointer is automatically incremented by the MV64360/1/2; the head pointer must be incremented by the local CPU.

PCI reads from the Outbound Queue Port return the data pointed to by QBAR + Outbound Post Tail Pointer (the next posted message in the Outbound Queue.) The following conditions apply:

- If the Outbound Post Queue is not empty (the head and tail pointers are not equal), the data is returned as usual and the MV64360/1/2 increments the Outbound Post Tail Pointer.
- If the Outbound Post Queue is empty (the head and tail pointers are equal), the value 0xFFFF.FFFF is returned.

As long as the Outbound Post Head and Tail pointers are not equal, a PCI interrupt is requested. This is done to indicate the need to have the external PCI agent read the Outbound Post Queue. When the head and tail pointers are equal, no PCI interrupt is generated since no service is required on the part of the external PCI agent (or PCI system host in the case of a PC server.) In either case, the interrupt can be masked in the OIMR register.

The CPU places outbound messages in the Outbound Post Queue by writing to the local memory location pointed to by the Outbound Post Head Pointer. After writing this pointer, it is the CPU's responsibility to increment the head pointer.

14.4.5 Outbound Free Queue

The Outbound Free Queue holds available outbound message buffers for the local processor to use.

External PCI agents place free messages at the queue head; the CPU fetches free message pointers from the queue tail. The tail pointer in maintained in software by the CPU. The head pointer is maintained automatically by the MV64360/1/2 upon a PCI agent posting a new ("returned") outbound free message.

PCI writes to the Outbound Queue Port result in the data being written to the local memory location at QBAR + Outbound Free Head Pointer. After the write completes, the MV64360/1/2 increments the head pointer.

From the time the PCI write ends till the data is actually written to SDRAM or Device, any new write to Outbound port will result in RETRY. If the head pointer and tail pointer become equal (an indication that the queue is full), an interrupt is sent to the CPU. If queue is full, a new PCI write to the queue will result in RETRY.

The processor obtains free outbound message buffers from the Outbound Free Queue by reading data from the location pointed to by the tail pointer. It is the processor's responsibility to increment the tail pointer.

Note

When initializing the Outbound Free Head /Tail Pointer registers, the Head Pointer register must be programmed first. If the Tail Pointer register is programmed first, the Outbound Free Queue Overflow Interrupt `OutFQOvr` is asserted.

14.4.6 Queue Data Endianess

Circular Queues access is not controlled by the PCI Access Control registers. The endianess convention of data placed in the circular queues is determined by the PCI Command register’s `SByteSwap` bit [16] and `SWordSwap` bit [11] (Table 378 on page 535). For more details, see 13.10 “Data Endianess” on page 172.

Section 15. Gigabit Ethernet Controller

The MV64360 implements three Gigabit Ethernet controllers operating at 10, 100, or 1000 Mbps. Two of the MACs have dedicated pins. The third port is multiplexed on the PCI interface's upper bits (Section 27. Reset Configuration).

The MV64361 implements two Gigabit Ethernet controllers operating at 10, 100, or 1000 Mbps. And, the MV64362 implements one Gigabit Ethernet controller operating at 10, 100, or 1000 Mbps.

15.1 Functional Overview

Each of these Gigabit Ethernet port includes an IEEE 802.3 compliant 10/100/1000 Mb MAC that supports GMII, MII, and 10-bit interfaces with an external PHY/SERDES device. The port speed, duplex and 802.3 flow control can be auto-negotiated, according to IEEE standards 802.3u and 802.3x. Backpressure is supported for half-duplex mode when operating at 10/100 Mb speeds. Each port supports MIB counters.

Each receive port includes a dedicated MAC-DA (Destination Address) with address filtering of up to 16-Unicast MAC addresses, 256 IP Multicast addresses, and 256 Multicast/Broadcast address. The receive ports may also detect frame-type encapsulation on layer2, as well as common layer3 and layer4 protocols.

IP checksum, Transmission Control Protocol (TCP) checksum, and User Datagram Protocol (UDP) checksum are always checked on received traffic, and may be generated for transmitted traffic. This capability increases performance significantly by off-loading these operations to the MAC from the CPU. Jumbo-frames are also supported.

Each port includes eight dedicated receive DMA queues and eight dedicated transmit DMA queues, plus two dedicated DMA engines (one for receive and one for transmit) that operate concurrently. Each queue is managed by buffer-descriptors that are chained together and managed by the software. The integrated memory is optional for placing the descriptor queues. This lowers the latency for the CPU, while off-loading the DRAM bandwidth. DRAM may be used optionally for larger storage for buffering data. Memory space may be mapped using configurable address windows to fetch/write buffer data and descriptors to any other interfaces in the device, such as DRAM, integrated SRAM, and even PCI, a device bus, and the CPU interface.

Queue classification on received traffic is assigned to the DMA queue based upon a highly configurable analysis, which evaluates the DA-MAC, IP, TOS (Type of Service), 802.1P priority tag, and protocol (ARP, TCP, or UDP). An example for use of this feature is implementing differentiated services in a router interface or real-time, jitter-sensitive voice/video traffic intermixed with data traffic. As each queue has its own buffering, blocking is avoided and latency is reduced for service by the CPU.

Detailed status is given for each receive frame in the packet descriptors, while statistics are accumulated for received and transmitted traffic in the MIB counters, on a per port basis.

There are eight receive queues and eight transmit queues. Receive/Transmit buffer management is by buffer-descriptor linked lists. Buffers and descriptors can reside throughout the entire MV64360/1/2 memory space. A Transmit buffer of any byte alignment and any size, above 8 bytes, is supported.The Receive Buffers must be 64-bit aligned. The core frequency assumption is a minimum of 83 MHz in gigabit operation.

Frame type/encapsulation detection is available on Layer2 for Bridge Protocol Data Unit (BPDU), VLAN (programmable VLAN-ethertype), Ethernet v2, LLC/SNAP, on Layer3 for IPv4 (according to Ethertype), other (no MPLS or IPv6 detection), and on Layer4 (only over IPv4) for Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and other. Frame enqueueing is in accordance with DA, VLAN-802.1p,IP-TOS using the *highest* priority counts, frame enqueueing is or captured according to protocol type for TCP, UDP, ARP, or BPDU. Frames smaller than the programmable minimum frame size are automatically discarded. Reception and transmission of long frames, up to 9700 bytes, are supported. The frame type, encapsulation method, errors, and checksums are reported in the buffer descriptor. Automatic IP header 32-bit alignment is done in memory by adding 2 bytes at the beginning of each frame. The TCP and UDP checksum calculations are put into the receive descriptor (and are compared with the frame checksum for non-IP fragmented frames), even for frames over 9 KB.

The Ethernet ports provide a great amount of flexibility with many programmable features. The TCP, UDP, and IP checksums are generated on any frame size. This is programmable on a per frame basis in the first descriptor. In addition CRC generation is programmable for each frame. There are separate, programmable transmit and receive interrupt coalescing mechanisms to aggregate several interrupts (on a time based masking window) before sending an indication to the CPU. The unit provides programmable zero padding of short frames, frames less that 64 bytes.

Byte based band-width distribution among transmit queues by a weighted-round-robin arbitration mechanism is programmable. This includes programming of hybrid fixed and round-robin priorities. The maximum byte based band-width allocation per transmit queue is also programmable. A transmit buffer of any byte alignment and any size (greater than 8 bytes) is supported; minimum packet size is 32 bytes.

In the event of collision, frames are retransmitted automatically without additional fetch. An Error and Collision report is provided in the last buffer descriptor.

15.2 Port Features

The 10/100/1000 Mbps Gigabit Ethernet ports provide the following features:

- IEEE 802.3 compliant MAC layer function.
- IEEE 802.3u compliant MII interface.
- 1000 Mbps operation - full duplex.
- 10/100 Mbps operation - half and full duplex.
- GMII symmetric flow control: IEEE 802.3x flow-control for full-duplex operation mode.
- MII symmetric flow control: Backpressure for half-duplex operation mode.
- Transmit functions:
  - Short frame (less than 64 bytes) zero padding.
  - Long frames transmission (limited only by external memory size).
  - Checksum on transmit frames for frames up to 9 KB.
  - Programmable values for Inter Packet Gap and Blinder timers.
  - CRC generation (programmable per frame).
  - Backoff algorithm execution.
  - Error report.

- Receive functions:
  - Address filtering modes:
    - 16 Unicast
    - 256 IP Multicast
    - 256 Multicast
    - Broadcast
  - Broadcast reject mode.
  - Automatic discard of errored frames, smaller than the programmable minimum frame size.
  - Reception of long frames (Programmable legal frame size is up to 9700 bytes).
    Note: Frames larger than the limit are actually received, however, they are mark in the descriptor as Oversize errors.
  - CRC checking.
  - Unicast promiscuous mode reception (receptions of unicast frames, even those not matched in the DA filter).
  - Error report.

15.3 Gigabit Ethernet Unit External Interface

Each of the Gigabit Ethernet ports has an external interface that can operate as a GMII, MII or TBI port.

MDIO — External serial interface to Ethernet MII PHY

For the Gigabit Ethernet port interface pin assignment, see Table 11: “Ethernet Port_0 Interface Pin Assignments” on page 56.

For the Gigabit Ethernet control interface pin assignment, see Table 13: “Ethernet Control Interface Pin Assignments” on page 61.

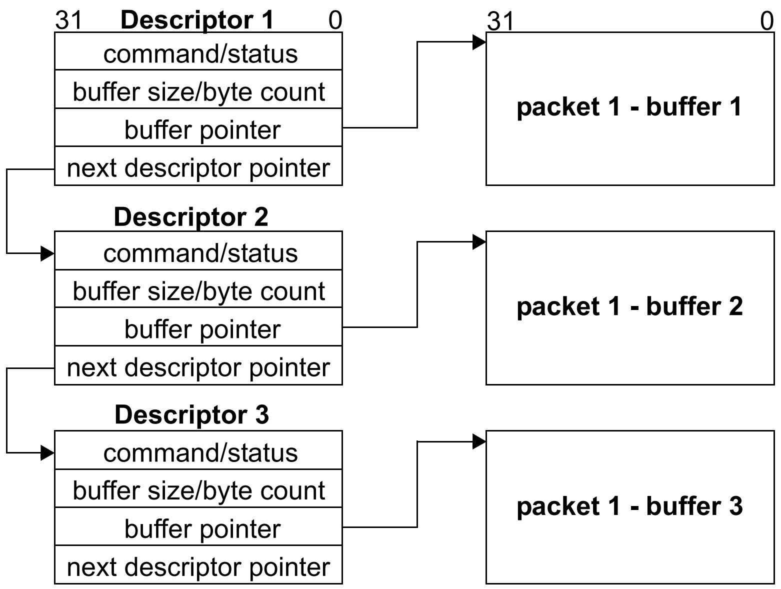
15.4 DMA Functionality

The Gigabit Ethernet unit provides Ethernet ports functionality, with each port capable of running at either 10 or 100Mbps (half- or full duplex), or 1000Mbps (full duplex only) independently of the other port. Each port interfaces a MII/GMII PHY or a 10-bit SERDES on its serial side and manages packet data transfer between memory and PHY or SERDES. The data is stored in memory buffers, with any single packet spanning multiple buffers if necessary. Upon completion of packet transmission or reception, a status report, which includes error indications, is (optionally) written by the Ethernet unit to the first descriptor (for receive ports) or to the last descriptor (for transmit ports) associated with this packet.

The buffers are allocated by the CPU and are managed through chained descriptor lists. Each descriptor points to a single memory buffer and contains all the relevant information relating to that buffer (that is buffer size, buffer pointer, etc.) and a pointer to the next descriptor. Data is read from the buffer or written to the buffer according to information contained in the descriptor. Whenever a new buffer is needed (end of buffer or end of packet), a new descriptor is automatically fetched, and the data movement operation is continued using the new buffer.

Figure 41 shows an example of memory arrangement for a single packet using three buffers.

Figure 41: Ethernet Descriptors and Buffers



The following sections provide detailed information about the operation and user interface of the Ethernet unit and its logic subsections.

Tx and Rx buffers are managed via link list of descriptors. Descriptors and buffers can be placed in any of the different MV64360/1/2 interfaces. However, the buffers are typically placed in DRAM and descriptors in DRAM or integrated SRAM. Buffers and descriptors are being read/write from/to memory by the port Rx and Tx DMAs.

15.4.1 Address Decoding

This section explains how the Gigabit Ethernet unit determines where to access memory for reading/writing descriptor and packets data, in the chip architecture.

The MV64360 and MV64361 Gigabit Ethernet ports share six address windows. Each address window can be individually configured.

Note

The MV64362 uses two address windows.

With each of the ports' DMA transactions (buffer read/write, descriptor read/write), the address is compared against the address decoding registers. Each window can be configured to different target interface. Address comparison is done to select the correct target interface (DRAM, integrated SRAM, etc.).

Note

The MV64360/1/2 Gigabit Ethernet unit has its own address decoding map that is de-coupled from the CPU interface address decoding windows.

The PCI interface supports 64-bit addressing. Four of the six address windows have an upper 32-bit address register. To access the PCI bus with 64-bit addressing cycles (DAC cycles), assign one (or more) of these four windows to target the PCI bus. The address generated on the PCI bus is composed of the window base address and the High Remap register.

For the port DMA to avoid accessing a forbidden address space (due to a programing bug), each port uses access protection logic that prevents it from read/write accesses to specific address windows.

If the address does not match any of the address windows, or if it violates the access protection settings, an interrupt is generated. The transaction is executed but not to the original address. Instead, the transaction is executed to a default address and target as specified in the Default Address and ID registers (see Section G.3 “Address Decoding Registers” on page 593).

Every access to the integrated SRAM or to the DRAM might require snoop action on the CPU bus to maintain cache coherency. Any of the address decoding windows targeted to the DRAM can be marked as a cache coherent region. Access to these regions results in a snoop on the CPU bus.

15.4.2 Endianess and Swap Modes

Each DMA channel has configurable behavior on Little or Big Endian support, per DMA channel data receive and data transmission - See the `BLMT` and `BLMR` fields in SDMA Configuration register (Table 514 on page 604).

For every DMA channel, descriptor accesses may be swapped or not. See Swap-mode field in SDMA Configuration register.

15.4.3 Transmit DMA Descriptors

15.4.3.1 Transmit Operation

In order to initialize a transmit operation, the CPU must do the following:

1. Prepare a chained list of descriptors and packet buffers.

Note

The TxDMA supports several priority transmit queues with programmable fixed or weighted priority with optional bandwidth limiting on the port or queue (see Section 15.8.1 “Priority Modes” on page 235). If the user wants to take advantage of this capability, a separate list of descriptors and buffers must be prepared for each of the priority queues.

2. Write the pointer to the first descriptor to the DMA’s current descriptor registers (TxCDP) associated with the priority queue to be started. If more than one of the priority queues are needed, initialize TxCDP for each queue.
3. Initialize and enable the Ethernet port by writing to the port’s configuration and command registers.
4. Initialize and enable the DMA by writing to the DMA’s configuration and command registers (Triggering the DMA is accomplished by setting the ENQ bit in the Tx Command register).

After completing these steps, the DMA starts and performs arbitration between the transmit queues according to the configuration, on a packet by packet basis, as explained in Section 15.8.1 “Priority Modes” on page 235. The DMA then fetches the first descriptor from the specific queue it decided to serve, and starts transferring data from the memory buffer to the Tx-FIFO. When the entire packet is in the FIFO (during which it may potentially calculate and update IP checksum, TCP, or UDP checksum), the port initiates transmission of the packet across the MII/GMII/10-bit interface. While data is read from the FIFO, new data is written into the FIFO by the DMA.

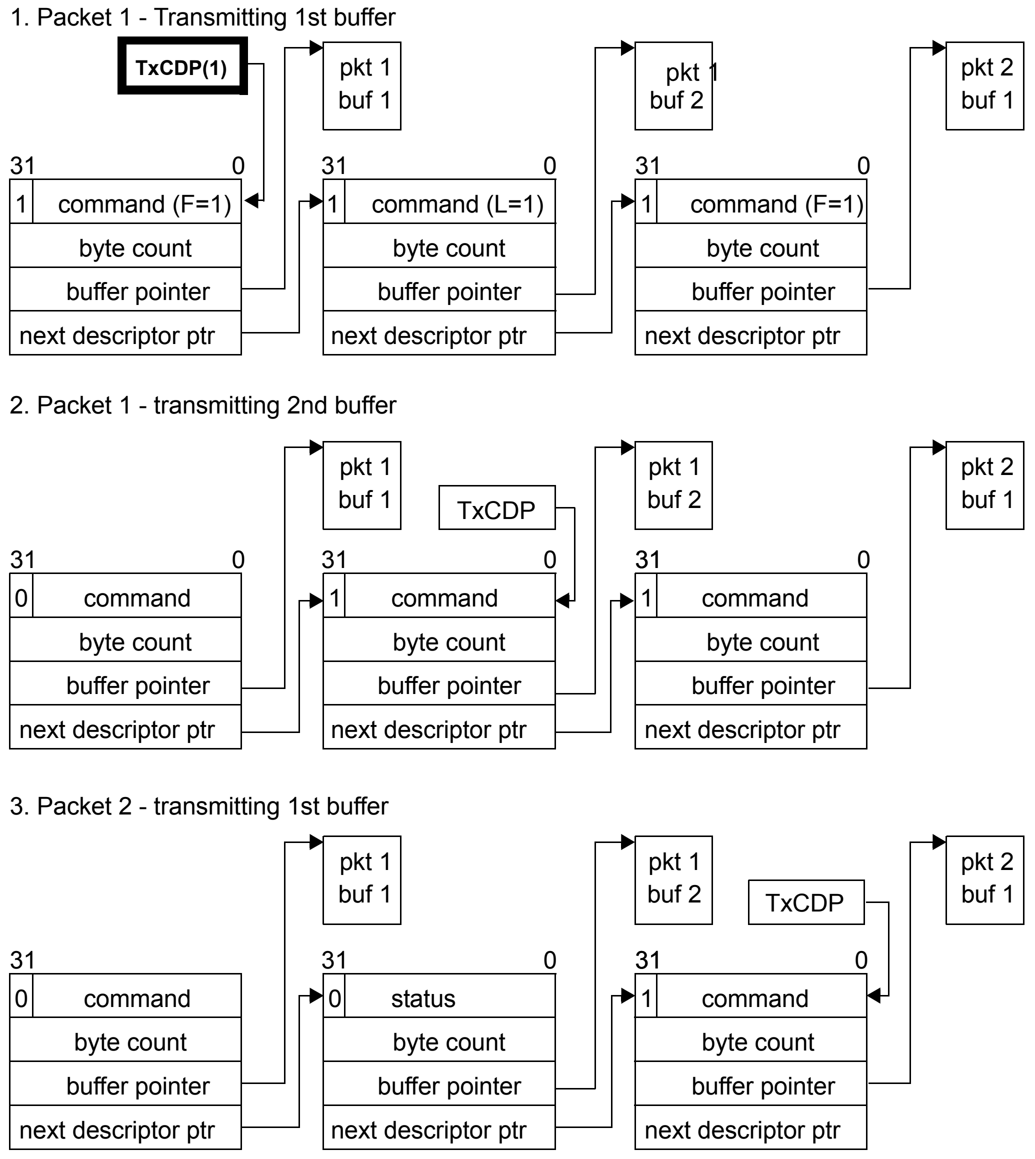
For packets that span more than one buffer in memory, the DMA will fetch new descriptors and buffers as necessary.

When transmission is completed, status is (optionally) written to the first long word of the last descriptor. The Next Descriptor's address, which belongs to the next packet in the queue, is written to the current descriptor pointer register.

This process (starting with DMA arbitration) is repeated as long as there are packets pending in the transmit queues. When that happens the DMA resets the ENQ bit in the Tx command register (one bit per queue) and reports the queue end via a TxEnd maskable interrupt in the ICRE register (for each queue).

Figure 42 shows how the transmit descriptors are managed when a two buffers packet is transmitted.

Figure 42: Ethernet Packet Transmission Example



1. TxCDP = Transmit Current Descriptor Pointer.
Key: pkt = packet, buf = buffer, ptr = pointer.

Ownership of any descriptor other than the last is returned to the CPU upon completion of data transfer from the buffer pointed by that descriptor. The Last descriptor, however, is returned to CPU ownership only after the actual transmission of the packet is completed. While changing the ownership bit of the Last descriptor, the DMA also writes status information, which indicates any errors that might have happened during transmission of this packet. There are two relevant modes:

- AM (Auto Mode): When this mode is set, the DMA will not close descriptors that are not last descriptors (since the only change in non-last descriptors is their ownership).
- AMnoTxES programmable bit in SDMA Configuration Register (Table 514 on page 604) mode: When this mode is set, the Last descriptors also are not closed.

Both modes, save time for crossbar access to DRAM or SRAM for descriptor closing.

The transmit buffer supports any byte alignment and any size (> 8 bytes) with a minimum packet size of 32 bytes.

15.4.3.2 Retransmission (Collision)

Full collision support is integrated into the Ethernet port for half-duplex operation mode. Half-duplex mode is supported in 10 and 100Mbps speeds only.

In half-duplex operation mode, a collision event is indicated each time receive and transmit are active simultaneously. When that happens, active transmission is stopped, the jam pattern is transmitted and the collision count for the packet increments. The packet is retransmitted after a waiting period, which conforms to the binary backoff algorithm specified in the IEEE 802.3 standard. The retransmit process continues for multiple collision events as long as a limit is not reached. This retransmit limit, which sets the maximum number of transmit retries for a single packet, is defined by the IEEE 802.3 standard as 16. However, the user can program the retries to work in a collision forever mode (see also Section 15.16.1 Enabling Partition Mode). This mode, ensures that once a frame is in the FIFO, it is transmitted (eventually) unless the line is very bad. The event of a single packet colliding 16 times is known as *excessive collision*.

As long as a packet is being retransmitted, its last descriptor is kept under port ownership. When a successful transmission takes place (i.e. no collision), a status word containing collision information is written to the last descriptor and ownership is returned to the CPU.

If a retransmit limit is reached with no successful transmission, a status word with error indication is written to the packet’s last descriptor, and the transmit process continues with the next packet.

It is important to note that collision is considered legal only if it happens before transmitting the 65th byte of a packet. Any collision event that happens outside the first 64 byte window is known as a *late collision*, and is considered a fatal network error. Late collision is reported to the CPU through the packet status, and no retransmission is done.

Note

Any collision occurring during the transmission of the transmit packet’s last four bytes is not detected.

15.4.3.3 Zero Padding of Short Frames

Zero Padding is a term used to denote the operation of adding zero bytes to a frame. This feature is used for CPU off-loading.

The Ethernet port offers a per frame padding request bit in the transmit descriptor. This causes the port logic to enlarge frames shorter than 64 bytes by appending zero bytes. When this feature is used, only frames equal or larger than 64 bytes are transmitted as is. Frames smaller than 64 bytes are zero padded and transmitted as 64-byte packets.

15.4.3.4 CRC Generation

Ethernet CRC denotes four bytes of Frame-Check-Sequence appended to each packet.

CRC logic is integrated into the port and can be used to automatically generate and append CRC to a transmitted packet. One bit in the transmit descriptor is used for specifying if CRC generation is required for a specific packet.

Error handling: If data was fetched with an unrecoverable error (for example, a data integrity error or a non-correctable ECC error from memory), CRC is not generated.

15.4.3.5 IP Checksum Generation

IPv4 checksum may be calculated during the packet DMA from memory, and it is replaced in the checksum field, for IPv4 packets, encapsulated in Ethernet-v2 format, with or without VLAN tag (This must be specified in the descriptor). IPv4 checksum is similarly supported for LLC/SNAP packets, including Jumbo frames per the Alteon definition (The CPU must set the LLC/SNAP-bit in the descriptor for such packets).

One bit in the transmit descriptor is used for specifying if the IPv4 checksum generation is required for a specific packet.

15.4.3.6 TCP Checksum Generation

The TCP checksum may be enabled per frame. When TCP checksum is enabled, it is calculated during the packet DMA from memory and replaced in the checksum field before transmission begins.

This is supported for TCP over IPv4 over Ethernet-v2, with or without VLAN tag (This must be specified in the descriptor). It is similarly supported for LLC/SNAP packets, including Jumbo frames per the Alteon definition. The CPU must set the LLC/SNAP-bit in the descriptor for such packets.

Since TCP segment may be transmitted over several Ethernet packets, and since the checksum in the next packets continue the checksum calculation of previous packets, there are two types of checksum generation commands (depending on bit 10 in the Tx descriptor):

- Calculate the checksum on the *first* packet in the segment: In that case the 16 bit checksum field in the descriptor must be zero. The checksum is done fully by the MV64360/1/2, and will include parsing the header according to the descriptor fields, calculate the checksum on pseudo-header. The checksum continues with full checksum calculation on the TCP data, and finally it is placed in the packet before transmission.
- Calculating checksum on *non-first* packets in the segment: The CPU is required to calculate the initial checksum, including the pseudo-header checksum in the L4iChk field in the Tx descriptor. The DMA uses this initial checksum value in calculating the TCP checksum over the TCP payload in the packet and place it in the TCP checksum field of the packet before transmission.

Note that the CPU may choose to always calculate the checksum over the pseudo header, and let the hardware take care of the payload checksum.

15.4.3.7 UDP Checksum Generation

The UDP checksum generation is the same as the TCP checksum generation support, with both first and non-first modes (see above).

15.4.3.8 VLAN Bit

The CPU is required to specify whether the packet is VLAN tagged or not. This is needed to facilitate the packet parsing during fetching from DRAM. It is used to correctly locate the IP header when the IP checksum, TCP checksum, or UDP checksum generation is required.

15.4.3.9 LLC/SNAP Bit

The layer3 and layer4 checksum generation is supported for Ethernet-v2 frames, or for LLC/SNAP frames (including jumbo frames). This bit must be set in case the checksum generation features is required for LLC/SNP frames or frames that comply with Alteon Jumbo Frame definition.

15.4.3.10Transmit Descriptor Structure

- Descriptor length is 4 long words (4LW), and it must be 4LW aligned (that is, Descriptor_Address[3:0]==0000).
- Descriptors may reside anywhere in the MV64360/1/2 address space except for a null address (0x00000000), which is used to indicate the end of the descriptor chain. Descriptor may not be placed on a Device-bus. Descriptors are fetched always in burst of 4LW.
- The last descriptor in the linked chain must have a null value in the Transmit Descriptor - Next Descriptor's `NextDescriptorPointer` bits [31:0] (Table 61 on page 221). Alternatively, the last descriptor may be not owned. Having a not owned descriptor is useful for performance optimization, by using a dummy pointer for adding descriptors to a chain without reprogramming the First Descriptor Pointer (FDP) register (see also Section 18.4.3 "Chain Mode" on page 304 and Section 18.4.8 "Descriptor Ownership" on page 309).
- For packets that span multiple descriptors, the CPU must provide ownership on all the packet's descriptors before giving ownership on the first descriptor of the packet, to avoid underrun situations.
- TX buffers associated with TX descriptors are limited to 64 KB and can reside anywhere in memory. However, buffers with a payload of one to eight bytes must be aligned to a 64-bit boundary. Zero size buffers are illegal.

Figure 43: Transmit Descriptor Description

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	Offset
Byte 3								Byte2								Byte1								Byte0								
Command / Status																																+0
Byte Count[15:0]																L4iChk/Reserved																+4
Buffer Pointer [31:0]																																+8
Next Descriptor Pointer [31:4]																																+C

15.4.3.11 Tx Descriptor Command/Status

Table 58: Transmit Descriptor — Command/Status

Bits	Field	Description
0	ES	Error Summary of MAC level errors on frame transmission. 0 = No Error 1 = Error occurred (Late Collision - LC, or Retransmit Limit - RL, or Underrun Error - UR) NOTE: This field is only valid only if `L` bit [20] is set. If `AM` bit [30] is set and the Port Configuration register's `AMNoTxES` bit [12] (Table 507 on page 600) is set, this field, as well as `EC` bits [2:1], are not updated.
2:1	EC	Error Coding 00 = LC 01 = UR 10 = RL reached (excessive collision) 11 = Reserved NOTE: Valid only if `L` bit [20] is set and `ES` bit [0] is set.
8:3	Reserved	Reserved
9	LLC/SNAP	When set, this bit signifies that the packet has an LLC/SNAP format. 0 = Not LLC/SNAP 1 = LLC/SNAP NOTE: Valid only if F is set, and if GL4chk or GIPchk is set. IP and TCP/UDP checksum is supported for LLC/SNAP frames or for Ethernetv2 frames This bit must be set for jumbo-frame formatted, according to Alteon specification as described in IEEE 802.3 LLC/SNAP.
10	L4Chk_Mode	Provides the TCP/UDP frame type for checksum calculation mode when GL4chk=1. 0 = Frame is IP fragmented. (The CPU must provide the initial checksum value calculated over the pseudo-header in the L4iChk field.) 1 = Frame is *not* IP fragmented. (The CPU must provide zero value in the L4iChk field.) NOTE: The payload length over which the checksum is calculated is determined by the Layer4 LENGTH field in the packet, and therefore it should NOT include any pad bytes. Valid only if F is set, and if GL4chk=1 and L4type=TCP or UDP
14:11	IPv4HdLen	Provides the length in long words (4 bytes) of the IPv4 header. NOTE: This is only valid if `GL4chk` bit [17] and `F` [21] are set.
15	VLAN	When `GL4chk` bit [17] is set, VLAN signifies if the Ethernet-v2 frame is VLAN tagged or not. Only if `GIPchk` bit [18] or GL4chk are set, this field must have a correct value. 0 = Frame is *not* VLAN tagged. 1 = Frame is VLAN tagged. NOTE: This is only valid if `F` bit [21] is set.

Table 58: Transmit Descriptor — Command/Status (Continued)

Bits	Field	Description
16	L4type	When GL4chk is set, signifies which Layer4 protocol is carried in the frame. 0 = TCP 1 = UDP NOTE: This is only valid if the `F` bit [21] is set.
17	GL4chk	Generate TCP/UDP Checksum 0 = No operation 1 = Generate TCP/UDP checksum. NOTE: This may only be set to TCP or to UDP over IPv4 over Ethernetv2 frames (tagged or untagged). The CPU must provide the initial checksum value calculated over the pseudo-header in the Transmit Descriptor register's `L4iChk` bits [15:0]. The payload length over which the checksum is calculated is determined by the Layer4 Length field in the packet, and therefore, it should NOT include any pad bytes. This is only valid if `F` bit [21] is set.
18	GIPchk	Generate IPv4 checksum. This is supported for Ethernetv2 and LLC/SNAP frames (tagged or untagged), with a valid IPv4 Header (`IPHL>=5, IPHL*4<=IPTL`). NOTE: This is only valid if the `F` bit [21] is set.
19	P	Padding When this bit is set and the packet is smaller than 60 bytes, zero-value bytes are appended to the packet. Use this feature to prevent transmission of fragments. NOTE: This is only valid if `L` bit [20] is set. If set, the `GC` bit [22] is regarded as also set.
20	L	Last Indicates the last buffer of frame.
21	F	First Indicates the first buffer of a frame.
22	GC	Generate Ethernet CRC 0 = Do not generate. 1 = Generate. NOTE: If `GIPchk` or `GL4chk` are set, this bit is regarded as set. Only valid if the `F` bit [21] is set.
23	EI	Enable Interrupt When set, a maskable interrupt will be generated upon the closing descriptor. NOTE: To limit the number of interrupts and prevent an interrupt per buffer situation, set this bit only in descriptors associated with Last buffers. This way the TxBuffer interrupt is only set when transmission of a frame is completed. Interrupts may be further delayed by the Interrupt coalescing mechanism (see Section 15.7.1 “Interrupt Coalescing” on page 234).

Table 58: Transmit Descriptor — Command/Status (Continued)

Bits	Field	Description
29:24	Reserved	Reserved.
30	AM	Auto Mode When set, the DMA will not clear the `Ownership` bit at the end of the buffer process. If the Port configuration register's `AMNoTxES` bit [12] (Table 507 on page 600) is set, no status is reported in the last descriptor (See `ES` bit [0] and `EC` [2:1], fields above).
31	O	Ownership Bit 0 = Buffer owned by the CPU. 1 = Buffer owned by the DMA.

Table 59: Transmit Descriptor — Byte Count

Bits	Name	Description
15:0	L4iChk	When the Transmit Descriptor's `GL4chk` bit [17] is set, the CPU should provide the initial checksum value calculated on the pseudo header. Otherwise these bits are reserved. NOTE: Only valid if the `F` bit [21] is set.
31:16	Byte Count	Number of bytes to be transmitted from the associated buffer. This is the payload size in bytes.

Table 60: Transmit Descriptor — Buffer Pointer

Bits	Name	Description
31:0	Buffer Pointer	A 32-bit pointer to the beginning of the buffer associated with this descriptor. NOTE: There is a 64-bit alignment requirement for buffers that have a setting in the Transmit Descriptor register's Byte Count bits [31:16] of 1–8 bytes.

Table 61: Transmit Descriptor — Next Descriptor Pointer

Bits	Name	Description
31:0	NextDescriptor Pointer	A 32-bit pointer that points to the beginning of the next descriptor. NOTE: bits [3:0] must be set to `0`. A DMA operation is stopped when a null (all zeros) value is encountered in this field.

15.4.3.12 Transmit DMA Pointer Registers

The Tx DMA employs a single 32-bit pointer register per queue: TxCDP.

- *TxCDP - TX DMA Current Descriptor Pointer:* TxCDP is a 32-bit register used to point to the current descriptor of a transmit packet. The CPU must initialize this register before enabling DMA operation. The value used for initialization should be the address of the first descriptor to use.

15.4.3.13 Transmit DMA Notes

The transmit DMA process is packet oriented. The transmit DMA does not close the last descriptor of a packet, until the packet has been fully transmitted. When closing the last descriptor, the DMA writes packet transmission status to the Command/Status word and resets the ownership bit. A TxBuffer maskable interrupt is generated in the ICRE register for each queue, if the `EI` bit in the last descriptor is set.

Updating the status in the descriptor is programmable per the `AM` bit in the Tx descriptor. When set, the DMA will not clear the Ownership bit at the end of buffer process. If, in addition AMNoTxES bit is set in the Port Configuration register, no status will be reported in last descriptor. The advantage of this is that it reduces memory write access per descriptor This versus the trade-off of not getting error indications per packet, like late collisions, and not relying on the ownership bit for each descriptor.

Transmit DMA stops processing a Tx queue whenever a descriptor with a null value in the Next Descriptor Pointer field is reached or when a CPU owned descriptor is fetched. When that happens, a TxEnd maskable interrupt is generated in the ICRE register (per queue) and the ENQ bit is reset. To restart the queue, the CPU should issue a Enable-queue command by writing '1' to the ENQ bit in the Tx command register.[1]

The transmit DMA does not expect a null Next Descriptor Pointer or a CPU owned descriptor in the middle of a packet. Also the transmit DMA does not expect a data integrity error on descriptors. When any of these events occurs, the DMA aborts transmission and stops queue processing (that is, it resets the ENQ bit). A TxError maskable interrupt is generated. To restart the queue, the CPU should issue an Enable_Queue command.

A transmit underrun occurs when the DMA cannot access the memory fast enough and packet data is not transferred to the FIFO before the FIFO becomes empty. In this case, the DMA aborts transmission and closes the last descriptor with a UR bit set in the status word. Also, a Tx_Underrun maskable interrupt is generated. The transmit process continues with the next packet. In the MV64360/1/2 Tx DMA, transmitting packets less than 10 KB long, such an error *cannot* happen, as the packet is fully buffered in the FIFO before transmission begins.

To stop DMA operation before the DMA reaches the end of descriptor chain, the CPU should issue a Disable-Queue command by writing '1' to the DISQ bit in the DMA command register. The DMA stops queue processing as soon as the current packet transmission is completed and its last descriptor returned to CPU ownership, and then resets the ENQ bit. In addition, a TxEnd maskable interrupt is generated. To restart this queue, the CPU must issue a Enable-Queue command.

When the Ethernet link was lost during normal operation, the DMA will disable all the queues by resetting the ENQ bits. Since loosing link may happen anytime during DMA programming by the CPU (for example, a disconnected cable or a far end disconnect) the following precaution must be taken: If the CPU gets a link-down interrupt, then the CPU must wait for the DMA to reset the ENQ bits of the DMA channels for Tx, after the link down event, before re-enabling the DMA channels.

Note

The RX DMA does not reset the enable bits under link down. To reprogram disable the queue by writing to the DISQ bits.

The CPU must never modify the DMA configuration register or the TxCDP register while the DMA ENQ bit is set. Modifying the TxCDP registers is allowed only when the respective DMA ENQ bit is reset. Modifying the DMA configuration registers may be done only when *all* the DMA channels ENQ bits are reset.

The DMA ENQ bit cannot be reset by the CPU. Only the hardware resets it as a response to the DISQ command, or an end-condition, error condition, or link down.

1. When the DMA stops due to a null descriptor pointer, the CPU has to write TxCDP before issuing an Enable_Queue command. Otherwise, TxCDP remains null and the DMA cannot restart the queue processing.

Note

Most of the terms used to denote either DMA commands (Enable_Queue and Disable_Queue) or interrupts (TxBuffer, TxEnd, and TxError) actually reflect multiple terms (one per queue). For example, the MV64360/1/2 provides eight Enable_Queue commands. The same applies to the other commands and interrupts listed above.

15.4.4 Receive DMA Descriptors

15.4.4.1 Receive Operation

To initialize a receive operation, the CPU must do the following:

1. Prepare a chained list of descriptors and packet buffers.

Note

The RxDMA supports eight priority queues. If the user wants to take advantage of this capability, a separate list of descriptors and buffers should be prepared for each of the priority queues.

2. Write the pointer to the first descriptor to the DMA's current receive descriptor registers (RxCDP) associated with the priority queue to be started. If multiple priority queues are needed, the user has to initialize RxCDP for each queue.
3. Initialize and enable the DMA channel by writing to the DMA's configuration and command registers.
4. Initialize the Ethernet port by writing to the port's configuration registers (among them PSCR, Address Filter Tables, MII/GMII Serial Parameter registers, if necessary) for the desired operational modes. Enable the port by writing to the `PortEn` bit in the PSCR register.

After completing these steps, the port starts waiting for a receive frame to arrive at the MII/GMII/10-bit interface. When this occurs, receive data is packed and transferred to the RxFIFO. At the same time, address filtering test is done to decide if the packet is destined to this port. If the packet passes the address filtering check, a decision is made regarding the destination queue to which this packet should be transferred. When this is done, actual data transfer to memory takes place. For detailed address filtering and priority queue assignment decisions, refer to Section 15.5 "Receive Frame Processing" on page 230.

Note

Packets which fail address filtering are dropped and not transferred to memory.

For packets that span more than one buffer in memory, the DMA will fetch new descriptors as necessary. However, the first descriptor pointer will not be changed until packet reception is completed.

When reception is completed, status is written to the first long word of the first descriptor, and the Next Descriptor's address is written to the current descriptor pointer register. This process is repeated for each received packet.

Notes

- Only after the packet had been fully received and status information was written to the first LW of the first descriptor, will the ownership bit be reset (that is, the descriptor is returned to CPU ownership).
- Ownership of any descriptor other than the first is returned to the CPU upon completion of the data transfer to the buffer pointed by that descriptor. This means that, for each packer, the first descriptor of a packet is the last descriptor to return to CPU ownership.

15.4.4.2 Receive DMA Pointer Register

The Rx DMA employs one 32-bit pointer register per queue: RxCDP.

- RxCDP is a 32-bit register used to point to the first descriptor of a receive packet. The CPU must initialize this register before enabling DMA operation. The value used for initialization should be the address of the first descriptor to use. CPU must not write to this register while the DMA is enabled. Reading from this register could be used to assess the DMA progress, as well as to monitor the queue status.

15.4.4.3 Receive DMA Notes

The Receive DMA process is packet oriented. The DMA does not close the first descriptor of a packet, until the last descriptor of the packet is closed. When closing the first descriptor, the DMA writes the status to the Command/Status word and resets the ownership bit. A RxBuffer maskable interrupt is generated if the EI bit in the first descriptor is set.

When the DMA encounters a null next descriptor pointer or a CPU owned descriptor during normal operation (both are the only legal queue end conditions), the current received frame may be closed with error status in the descriptor, if there is insufficient space to store it in memory. The RxDMA engine will assert a maskable RxErrorQueue interrupt.

If the end-condition was a null next descriptor pointer, the DMA disables the queue by resetting the ENQ bit once it tries to prefetch the next descriptor. If the RxDMA requests a new descriptor before the CPU re-enables the queue, the DMA increments the Discarded Frames Counter (DFC). Any new frame to this queue will be discarded. If the ending condition was a unowned descriptor, then the DMA does not disable itself, but rather continues to try to read the descriptor, every time a new frame arrives to this queue.

The latter case optimizes for high speed descriptor-buffer receive allocation, as it allows the CPU to avoid re-enabling the queue, every time it adds new descriptors to the queue.

Before the CPU may enable the queue again, it must write the correct descriptor pointer to the RxCDP register. Alternatively, in case the queue end was a result of an unowned descriptor, the CPU may simply provide ownership of it to the DMA and re-enable it.

When a frame is received while the Ethernet link was lost (link down), the last frame received is cut-off and closed as a bad CRC in the first descriptor.

The CPU must never modify the DMA configuration register or the RxCDP register while the DMA ENQ bit is set. Modifying the RxCDP registers is allowed only when the respective DMA ENQ bit is reset.

DMA ENQ bits are reset after the CPU writes to the DISQ bits, and the DMA completes the current transaction on the disabled Queue (if working with the specific disabled Queue).If the CPU gets a NULL of not owned descriptor in the middle of a chain and the CPU does not solve the problem in time, the frame will be discarded, The last closed descriptor will be re-closed as a last descriptor, and the first descriptor will be closed with a resource error.

15.4.4.4 Frame Type Indications

The receive processing of the frame (See Section 15.5 “Receive Frame Processing” on page 230) allows passing various useful indications about each individual packet in the Rx descriptor to convey MAC level errors (like Ethernet CRC check fail) and to facilitate CPU processing overhead in packet header processing and in layer3 and layer4 checksum calculations.

See the descriptor description for details, and for a definition of the indications see Section 15.5 “Receive Frame Processing” on page 230.

15.4.4.5 TCP Checksum Checking

TCP frames include a 16-bit checksum that protects the entire segment payload (that usually spans over a number of packets) as well as TCP header and some of the IPv4 fields.

Frames may be received in an interleaved fashion from different TCP connections, and also out of order, within any TCP connection.

The Rx frame parsing allows off loading most of the overhead from the software. The Rx descriptor below, provides frame type indications such as: IPv4, validity of IP header with correct IPHL, IPTL, and IP checksum checked OK, Layer2 encapsulation information (VLAN, Ethernetv2 or LLC/SNAP) and TCP or UDP type detection.

TCP checksum check results are generated in the Rx descriptor in the following way, where two cases are identified:

1. For frames that have in the IP Header Flags<MF> = 0 and Offset = 0x0: This means that the IP is not fragmented (The IPv4Frg bit is reset in the descriptor), and therefore, take into account that the L4 has the L4 header in this frame and that the L4 payload can be calculated (see note below for the calculation).

   Note

   The length field for the pseudo header is taken from the following operation: IPTL - IPHL * 4.

   For the checksum calculation, the value 16'h00 is used instead of the checksum field in the received frame as required by the standard. In addition, the checksum calculation for each frame always starts with the initial value of 16'h00.

   The descriptor will be closed with an indication that frame is not fragmented, and the L4 checksum compare result will be valid.

2. For frames that are not from the type of #1 (either MF!= 0 or Offset!= 0) - This means that the IP is fragmented (The IPV4Frg bit is set in the descriptor.) and therefore, the pseudo header is not calculated in the checksum.
   The checksum is calculated only on the L4 payload and places the result in the F descriptor of each frame. The fragment bit is set. Therefore, the checksum compare bit (L4ChkOK bit) is not valid.

   Note

   For this type of frame, the checksum calculation does *not* put zero in the checksum field, and therefore, in frames that have Offset =0x0 and MF!= 0 (first fragment of IP), the checksum including the checksum field of the TCP header may be calculated. This should be corrected by the software driver.

   For this type of frame, the software should sum all the checksum calculations for the complete IP frame and subtract the actual checksum field that was received in the frame and then do the compare by itself.

15.4.4.6 UDP Checksum Checking

UDP frames include a 16-bit checksum that protects the entire segment payload (which usually spans over a number of packets) as well as UDP header and some of the IPv4 fields.

Frames may be received in an interleaved fashion from different UDP streams, and also out of order, within any UDP stream.

The Rx frame parsing allows off loading most of the overhead from the software. The Rx descriptor below, provides frame type indications such as: IPv4, validity of IP header with correct IPHL, IPTL, and IP checksum checked OK, Layer2 encapsulation info (VLAN, Ethernetv2 or LLC/SNAP), and TCP or UDP type detection.

UDP checksum check results are generated in the Rx descriptor in the following way, where two cases are identified:

1. For frames that have in the IP Header Flags<MF> = 0 and Offset = 0x0: This means that the IP is not fragmented (IPv4Frg bit is reset in the descriptor) and therefore, take into account that the L4 has the L4 header in this frame as a result:

   The checksum is calculated with the corresponding pseudo header and compares the results to the frame L4 checksum. *(If the frame checksum is 0x0, then the checksum function does not compare and close the descriptor as checksum OK, since this is the indication that checksum check was disabled, according to the standard)*.

   Note

   The length field for the pseudo header is taken from the following operation: IPTL - IPHL * 4.

   For the checksum calculation, the value 16'h00 is used instead of the checksum field in the received frame as required by the standard. In addition, the checksum calculation for each frame always starts with the initial value of 16'h00.

   The descriptor will be closed with an indication that frame is not fragmented and the L4 checksum compare result will be valid.

2. For frames that are not from the type of #1 (either MF!= 0 or Offset!= 0) - This means that the IP is fragmented (The IPv4Frg bit is set in the descriptor.) and the pseudo header is not calculated in the checksum.

   The checksum is calculated only on the L4 payload and places the result in the F descriptor of each frame. The fragment bit is set. Therefore, the checksum compare bit (L4ChkOK bit) is not valid.

   Note

   For this type of frame do NOT put zero in the checksum field, and therefore, in frames that have Offset =0x0 and MF!= 0 (first fragment of IP), calculate the checksum including the checksum field of the UDP header. This should be corrected by the software driver.

   For this type of frame, the software should sum up all the checksum calculations for the complete IP frame and subtract the actual checksum field that was received in the frame and then do the compare by itself.

15.4.4.7 BPDU Indication

If frame is detected as BPDU, and BPDU detection is enabled then the BPDU bit is set (see also Section 15.5 “Receive Frame Processing” on page 230). The rest of the L3/4 fields are still provided, but the user may want to ignore them, as they will likely not be relevant for most BPDU protocols.

15.4.4.8 Receive Descriptor Structure

- Descriptor length is 4LW, and it must be 4LW aligned (i.e. Descriptor_Address[3:0]==`0000`).
- Descriptors may reside anywhere in the address space except for the null address (0x00000000), which is used to indicate the end of a descriptor chain. Descriptors cannot be placed on a Device-bus as they are fetched always in a burst of 4LW.

- The last descriptor in the linked chain must have a null value in the Transmit Descriptor - Next Descriptor's `NextDescriptorPointer` bits [31:0] (Table 61 on page 221). Alternatively, the last descriptor may be not owned. The latter option is useful for performance optimization, by using a dummy pointer for adding descriptors to a chain without reprogramming the RxCDP register (see also Section 18.4.3 “Chain Mode” on page 304 and Section 18.4.8 “Descriptor Ownership” on page 309).
- Receive buffers associated with Receive descriptors are limited to 64 KB and must be 64-bit aligned (i.e. Buffer_Address[2:0]==000).
- The minimum buffer size for the Receive buffer is eight bytes.

Figure 44: Receive Descriptor Description

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	Offset
Byte 3								Byte2								Byte1								Byte0								
Command / Status																																+0
Byte Count[15:0]																Buffer Size[15:3]													Frg	0	0	+4
Buffer Pointer [31:3]																													0	0	0	+8
Next Descriptor Pointer [31:4]																												0	0	0	0	+C

15.4.4.9 Receive Descriptor Command/Status

Table 62: Receive Descriptor — Command/Status

Bits	Name	Description
0	ES	Error Summary 0 = No Error 1 = Error Occurred (SF or MF or OR or RE), NOTE: This is only valid if `F` bit [27] is set.
2:1	EC	MAC Error Coding 00 = CE - CRC Error 01 = OR - Overrun Error 10 = MF - Maximum Frame Length Error. Frame is longer than the MAX_FRAME_SIZE. 11 = RE - Resource Error (No descriptors in the middle of the frame) NOTE: This is only valid if the `F` bit [27] and the `ES` bit [0] are set. If multiple errors occurred, then the reporting priority is Resource Error, Maximum Frame Length Error, Overrun Error, and CRC Error.

Table 62: Receive Descriptor — Command/Status (Continued)

Bits	Name	Description
18:3	L4Chk	Calculated TCP/UDP Checksum NOTE: This is only valid if Layer4 bits [22:21] are set to '00' or '01'. This is only valid if the `F` bit [27] is set and no MAC errors occurred. This is only valid if the `IPHeadOK` bit [25] and the `L3IP` bit [24] are set. The calculation does not include the pseudo header if the Receive Descriptor - Byte Count register's `IPv4Frg` bit [2] is set.
19	VLAN	VLAN Frame is VLAN tagged (according to programmed VLAN-Ethertype). NOTE: This is only valid if the `F` bit [27] is set, and the `ES` bit [0] is set to '0'.
20	BPDU	Bridge Protocol Data Unit Set when the frame is BPDU. NOTE: Only valid if the `F` bit [27] is set and the `ES` bit [0] is set to '0'.
22:21	Layer4	Frame encapsulation and protocol. 00 = Frame is TCP over IPv4 over Ethernetv2 or LLC/Snap (with or without VLAN tag). The Checksum result is provided in the `L4Chk` bits [18:3]. 01 = Frame is UDP over IPv4 over Ethernetv2 or LLC/Snap (with or without VLAN tag). The Checksum result is provided in the `L4Chk` bits [18:3]. 10 = Other Frame type. 11 = Reserved NOTE: This is only valid if the `F` bit [27] is set, and the `ES` bit [0] is set to '0'. This is only valid if the `IPHeadOK` bit [25] and the `L3IP` bit [24] are set.
23	Layer2Ev2	Set if Layer2 is Ethernetv2.
24	L3IP	Frame type is IPv4. This is only set if Ethertype-0x800 over Ethernetv2, or over LLC/SNAP (with or without VLAN tag). Otherwise, reset. NOTE: This is only valid if the `F` bit [27] is set, and the `ES` bit [0] is set to '0'.
25	IPHeadOK	IP header is "ok" is: • Frame type is IPv4 • IPHL >=5 • IPHL *4<= IPTL • IPheader Checksum is OK. 0 = Check failed 1 = Check passed NOTE: This is only valid if `L3IP` bit [24] is set. This is only valid if the `F` bit [27] is set, and the `ES` bit [0] is set to '0'.
26	L	Last Indicates the last buffer of a frame.
27	F	First Indicates the first buffer of a frame.

Table 62: Receive Descriptor — Command/Status (Continued)

Bits	Name	Description
28	U	Unknown Destination Address The frame is Unicast and was not matched to the MAC Address Base (DA[47:6]). NOTE: This is only set if working in promiscuous mode. See the Port Configuration register's `UPM` bit [0] (Table 507 on page 600). This is only valid if the `F` bit [27] is set and the `ES` bit [0] is set to '0'.
29	EI	Enable Interrupt When set, a maskable interrupt is generated upon the closing descriptor. NOTE: To limit the number of interrupts and prevent an interrupt per buffer situation, set the RIFB (Receive Interrupt on Frame Boundary) bit [0] in the SDMA Configuration register (see Table 514 on page 604). Interrupts may be further delayed by the Interrupt coalescing mechanism (see Section 15.7.1 "Interrupt Coalescing" on page 234).
30	L4ChkOK	Layer4 Checksum OK 1 = OK (passed) 0 = Check failed NOTE: If `Layer4` bits [22:21] is '01' and the received frame checksum field was 16'h00, then the bit will indicate passed. This is only valid if the `IPHeadOK` bit [25] and the `L3IP` bit [24] are set. This is only valid if Layer4 is '00' or '01'. This is only valid if the `F` bit [27] is set, and the `ES` bit [0] is set to '0'. This is only valid if the Receive Descriptor - Byte Count register's `IPv4Frg` bit [2] is cleared.
31	O	Ownership 0 = Buffer owned by the CPU. 1 = Buffer owned by the DMA.

Table 63: Receive Descriptor — Byte Count

Bits	Name	Description
1:0	Reserved	Reserved.
2	IPv4Frg	IPv4 is fragmented. 1 = Fragmented 0 = Not fragmented NOTE: This is only valid if the `IPHeadOK` bit [25] and the `L3IP` bit [24] are set. This is only valid if the Receive Descriptor - Command/Status register's `F` bit [27] is set, and `ES` bit [0] is set to '0'.

Table 63: Receive Descriptor — Byte Count (Continued)

Bits	Name	Description
15:3	Buffer Size	Buffer Size in Bytes When the number of bytes written to this buffer is equal to the field's value, the DMA closes the descriptor and moves to the next descriptor. NOTE: The number of bytes must be a multiple of 8; therefore, bits Buffer Size[2:0] must be set to 0.
31:16	Byte Count	When a descriptor is closed, this field is written by the device with a value indicating the number of bytes actually written by the DMA into the buffer. NOTE: This is only valid if the F bit [27] is set.

Table 64: Receive Descriptor — Buffer Pointer

Bits	Name	Description
31:0	Buffer Pointer	A 32-bit pointer to the beginning of the buffer associated with this descriptor. NOTE: This field must be 64-bit aligned; therefore, bits [2:0] must be set to 0.

Table 65: Receive Descriptor — Next Descriptor Pointer

Bits	Name	Description
31:0	Next Descriptor Pointer	A 32-bit pointer that points to the beginning of the next descriptor. NOTE: This field must be 4LW aligned; therefore, bits [3:0] must be set to 0. The DMA operation is stopped when a null (all zeros) value in the Next Descriptor Pointer field is encountered.

15.5 Receive Frame Processing

Once a frame is received by the port, the frame is parsed to go through the following processing:

- MAC errors checking.
- Accept or reject decision.
- Select the Receive queue (0 through 7).
- MIB counter increments.
- Extract Layer2/3/4 protocols and perform IP and/or TCP/UDP checksum.

Some MAC level errors, like fragments, are normally filtered from reception of frames and are only counted in MIB counters. Other MAC level errors (like CRC errored frames and frames beyond the maximum allowed size) are reported in the first descriptor and in the MIB counters.

Notes

- The frames maximum size is defined in the Port Serial Control register's MRU bits [19:17] (Table 519 on page 607). The receiver can also accept jumbo frames, where the 802.3 Type/Length field is set to 0x8870 (with or without 802.1Q VLAN tag).
- In this datasheet, the MAC Destination address bit [47] is the Multicast/Unicast bit. The first DA byte received on the GMII RXD[7:0] pins is DA[40:47]. The last byte GMII received on the RXD[7:0] pins is DA[0:7]).

15.5.1 Parsing the Frames

The frame goes through the following stages to decide whether or not to accept the frame and to determine which queue receives it:

1. If the frame is in the Bridge Protocol Data Unit (BPDU) format (DA is equal to 01-80-C2-00-00-00 through 01-80-C2-00-00-FF, except for the Flow-Control Pause packets) and the Port Configuration Extend register's Span bit [1] is set (Table 508 on page 602), the frame is received to queue 7, the highest priority queue.
2. If the frame is Unicast then the MAC DA bits [47:4] are compared with MAC[47:4] (see Table 492 on page 592 and Table 492 on page 592). If they do not match, then according to the Port Configuration register's `UPM` (Unicast Promiscuous Mode) bit [0] (see Table 507 on page 600), the frame is rejected or accepted to the queue in the register's `RXQ` bits [3:1]. If matched, then the MAC DA[3:0] bits are used as a pointer to the Unicast Table entries in the DA-Filter table. The DA Filter Special Multicast Table register's `Pass` and `Queue` bits (Table 545 on page 625) determines whether to filter the frame and its queue number.
3. Or, if DA=0xFFFFFFFF and the protocol is 0x806 Address Resolution Protocol (ARP), in Ethernet-v2, tagged or not, the frame is accepted or rejected according to the Port Configuration register's `RBArp` bit [9] setting and the queue is handled by the `RxqARP` bits [6:4].
4. Or, if DA=0xFFFFFFFF and the protocol is 0x800 Internet Protocol (IP), in Ethernet-v2 or LLC/SNAP, tagged or not, the frame is accepted or rejected according to the Port Configuration register's `RBIP` bit [8] and the queue is handled by the RXQ bits [3:1].
5. Or, if DA=0xFFFFFFFF and the frame is accepted or rejected according to the Port Configuration register's `RB` bit [7] and the queue is handled by the `RXQ` bits [3:1].
6. Or, if DA=0x01-00-5E-00-00-XX (where XX is between 0x00 and 0xFF) the MAC DA[7:0] bits are used as a pointer to the Special Multicast Table entries in the DA-Filter table. The `Pass` and `Queue` bits determine the frames' filter and queue number.
7. Or, if (the frame is a Multicast of another type): A CRC-8bit (Polynomial: x^8+x^2+x^1+1) and the result is used as an index to the Multicast Broadcast Table entries in the DA-Filter table. The `Pass` and `Queue` bits determine the frames' filter and queue number.

In stages 2 and 4–7, If:

- The frame is *not* discarded,
- The frame is not BPDU and the Port Configuration Extend register's Span bit [1] is not enabled,
- The frame is not a Unicast frame that is accepted because of the Port Configuration register's `UPM` bit [0] setting or the frame is ARP protocol broadcast,

the resulting queue is still not the final one. The final queue is determined by one of the following scenarios:

- If the frame is a Transmission Control Protocol (TCP) frame and the Port Configuration register's `TCP_CapEn` [14] is set, the queue number is determined by the Port Configuration register's `TCPQ` bits [18:16].
- If the frame is a User Datagram Protocol (UDP) frame and the Port Configuration register's `UDP_CapEn` bit [15] is set, the queue number is determined by the Port Configuration register's `UDPQ` bits [21:19].
- If the frame is VLAN tagged or is an IPv4 frame (over Ethernet v2 or LLC/SNAP), the queue number is determined as the *highest* queue from the one determined by the three DA filters, as described in stages 2–7.
  - 802.1p if the frame is VLAN tagged (Table 511 on page 603)
  - Differentiated Services-Code-Point if frame is IPv4 over EthernetV2 or LLC/Snap.

Notes

- To not use 802.1p mapping or the Differentiated Services CodePoint (DSCP), program those registers as all zero values, which are their default value.
- Before enabling the port, the DA-Filter tables must be programmed in full, as their initial values is undefined.

15.6 Performance Aspects

The MV64360 includes three gigabit ports in the Ethernet Unit. The MV64361 includes two gigabit ports. And, the MV64362 implements one port.

When optimizing the setup for performance, the memory mapping of the descriptors and data locations are flexible. Also, internal memory and DRAM can be mapped for different data structures.

It should also be taken into account that CPU latency may exist when processing large amounts of packets-per-second (pps). Past experience in router implementation shows that the overhead associated with descriptor access to DRAM is similar to that of the data-processing in DRAM.

The internal non-blocking crossbar architecture of the MV64360/1/2 also allows concurrent transactions from any unit to both SRAM and DRAM. It is therefore recommended that descriptor structures be placed in the internal 2 Mb SRAM memory to:

- Reduce the DRAM bandwidth consumption.
- Reduce the CPU latency in descriptor accesses (that is, use the SRAM that is on chip, rather than the DRAM that is off-chip).

In the MV64360 and MV64361, the internal SRAM can hold as much as 16K descriptors.

Note

There is no internal SRAM in the MV64362.

This assumes the DDR-DRAM is operating in 133 MHz (17 Gbps raw bandwidth).

For applications where all the descriptors are in the internal SRAM, all gigabit ports can operate at full-wire-speed for 64-byte packets back-to-back (4.5 Gbps in each direction). All three elements of the chip resources are non-blocking:

- SRAM bandwidth
- Internal crossbar bandwidth
- DRAM bandwidth

However, in this case, the limiting factor is the CPU processing power, which is not able to service and generate 4.5 Gbps. Therefore, in this case it may be a viable alternative to utilize the capability to sustain a burst of received short packets from all ports. When the buffers have been prepared in advance by the CPU, transmitting frames at a sustained rate, matches the CPU's capability to prepare frames for transmission.

If all descriptors and data are located in DRAM, then the main bottleneck in 64-byte packets is the DRAM bandwidth. The Ethernet Unit can recognize a congestion case on receive by internal urgent thresholds on the ports receive FIFO. In such cases, the unit will increase the Inter-packet Gap (IPG) between transmitted packets to avoid overrun on input, as much as possible. This results in dedicating most of the bandwidth to receive, in such peak cases (until there are no buffers on receive, or until the peak load is over). In this case only one gigabit port would be able to work without loosing packets on peaks of 1.5 Mpps (million packets per second) received while transmitting. This would still leave ~35% of the bandwidth for the CPU during these peaks.

Note

For more information on IPG, see Section 15.14 “Inter-packet Gap” on page 241.

Even if all data and descriptors are in DRAM, long packets have ample bandwidth. This will consume less than 50% of the DRAM bandwidth, even at full-wire speed for send and receive on all ports.

Marvell strongly advises using the internal SRAM for descriptor traffic and other small and frequently used data structures transmitted or received by the CPU.

Another aspect that effects performance is optimization on interrupts. This is described in Section 15.7 Ethernet Interrupts.

15.7 Ethernet Interrupts

The Gigabit Ethernet unit provides many interrupt causes to the MV64360/1/2 interrupt controller.

The following registers provide the interrupt reasons for the Ethernet unit interrupt bits:

- Ethernet Unit Interrupt Cause register (EUICR) and Ethernet Unit Interrupt Mask register (EUIMR). See Table 496 on page 594 for the EUICR and Table 497 on page 595 for the EUIMR.
- Ethernet Port 0/1/2 Interrupt Cause register (PICR0/1/2) and Ethernet Port 0/1/2 Interrupt Mask register (PIMR0/1/2). See Table 527 on page 615 for the PICR0/1/2 and Table 529 on page 619 for the PIMR0/1/2.
- Ethernet Port 0/1/2 Interrupt Cause Extend register (PICER0/1/2) and the Ethernet Port Extend Interrupt Mask register (PEIMR0/1/2). See Table 528 on page 617 for the PICER0/1/2 and Table 530 on page 619 for the PEIMR0/1/2.

The Ethernet unit provides several interrupt bits to the main MV64360/1/2 interrupt controller unit. The interrupt controller unit assigns them to different places according to the programmer policy. For more information, see Section Section 25. “Interrupt Controller” on page 340.

These bits are:

- Unit level Interrupt: Sourced from the EUICR and masked by the EUIMR.
- Per Port General Interrupt (one per port): For each port, this interrupt reflects the summary of the EUICR (masked by the EUIMR).

Note

One of the bits in PICR is a summary of PICER, so effectively the per port interrupt covers both cause registers of the port (regular and extend cause).

- Three interrupts per port, that separate the cause bits in PICR and PICER into three categories:
  - Misc: Covers bits PICER [16], PICER [20:23].
  - Rx events: Cover bits PICR [18:2], PICER[17,18].
  - Tx events: Cover bits PICR [30:19], PICER[15:0,19].

This last category allows soliciting the Receive and Transmit interrupt service to multiple processors and allows for faster service for packet receive and transmit.

15.7.1 Interrupt Coalescing

Since the Gigabit Ethernet line rate provides a high packet rate, it is important to reduce the amount of interrupts that the Ethernet DMA may generate.

For this purpose, the DMA Receive and Transmit have several modes that provide the option of choosing the type of events that initiate issuing interrupts. (See also Table 528 on page 617, Table 496 on page 594, and Section 18.2 “IDMA Descriptors” on page 301).

The most intensive interrupts are the packet-level interrupts on receive and transmit. In addition to the CPU’s ability to specify, in the Receive and Transmit descriptor, which descriptor close may cause an interrupt, the MV64360/1/2 provides a programmable mechanism that allows coalescing these types of interrupts.

On a per port basis, and for Rx and Tx transactions, the MV64360/1/2 has a programmable timer in the SDMA Configuration register’s `IPG_Int_Rx` bits [21:8] (see Table 514 on page 604) for receive and `IPG_Int_Tx` (transmit) to force a minimum time between interrupts associated with the Port Interrupt Cause register’s `RxBufferQueue` bits [9:2] and the Port Tx FIFO Urgent Threshold register `IPG_Int_Tx` bits [17:4] (see Table 532 on page 620). This minimum time is programmable and may be changed dynamically during normal operation.

The flow for packet-level interrupts on receive and transmit is as follows:

Note

The following example describes the Receive flow, however, the Transmit flow is implemented in an identical fashion.

1. A non-masked `RxBufferQueue` interrupt cause is asserted. As a result, an interrupt is raised (propagated in the interrupt hierarchy etc.) and the relevant interrupt coalescing counter begins to count. From this point (after CPU read from the interrupt register) until the count-down finishes, *no new interrupts can be raised* due to new packet reception (transmission) from any of the eight queues.
2. During the countdown time, and before the next CPU read of the PICR register (or PICER for Tx), the `RxBufferQueue` events would still cause loading ‘1’ to the appropriate cause bit, but would not cause raising an interrupt at the unit, port, or port-Rx (Tx) level. Before reading the register, it is assumed that the CPU do not reset the `RxBufferQueue` cause bits.
3. Once the CPU reads the PICR (or PICER for Tx), the value of all `RxBufferQueue` interrupts that are recorded later on, accumulate in a shadow register, actually two separate registers — one for PICR `RxBufferQueue` and one for PICER TxBuffer additional events, both of which are invisible to the software.
4. When the countdown timer expires, the `RxBufferQueue` in the shadow register is loaded into the PICR (or PICER for Tx). This may cause raising an interrupt, again.

This mechanism prevents loss of interrupt indications in the time frame between the time that the CPU reads the interrupt register and the time that it starts switching off interrupt bits. During this interval, new interrupts that arrive for the same receive or transmit queue would have been lost and buffers might have gotten stuck indefinitely. The shadow registers, just described, prevents this from happening.

15.8 Transmit Weighted Round-Robin Arbitration

The MV64360/1/2 transmit port includes flexible bandwidth control distribution among eight transmit queues. For a transmit queue to be selected to transmit the next frame, the queue must be enabled by setting the corresponding bit to ’1’ in the Transmit Queue Command register’s `ENQ` bits [7:0] (Table 522 on page 613), and the queue should have a frame ready for transmit.

15.8.1 Priority Modes

Each transmit queue can be configured in two modes:

- Fixed priority mode
- Weighted-Round-Robin (WRR) priority mode

The priority mode is configured by the Transmit Queue Fixed Priority Configuration register's FIXPR bits [7:0]. Setting these bits to '1' means the transmit queue is set to the Fixed priority mode. Setting these bits to '0' means the queue is configured to a WRR priority mode. Transmit Queue Fixed Priority Configuration register bit [7] is assigned to Transmit queue 7 and bit [0] is assigned to Transmit queue 0.

The transmit port arbitrates between the queues in two modes in the following fashion: While there is an enabled non-empty Fixed priority queue, select to transmit the next frame from the Fixed priority queue or else select to transmit from the WRR priority queue(s).

15.8.2 Fixed Priority Mode

Select the Fixed Priority mode to transmit by selecting the non-empty, enabled and non-bandwidth limited queue with the highest queue number first (for example, select queue 8, then 7, etc.). A queue may be excluded from the arbitration, if it passed an optional per-queue programmable bandwidth-limitation based on the token-bucket mechanism (see Section 15.9 "Token Rate Configuration" on page 237).

15.8.3 Weighted Round-Robin Priority Mode

The port would service one of the Weighted Round Robin (WRR) queues only when the fixed priority queues have nothing to transmit.

The WRR priority mode is utilized to distribute bandwidth among transmit queues in a round-robin fashion when each WRR queue gets bandwidth portion relative to its configured Weight. This is called below "WRR arbitration".

A WRR queue may be excluded from the WRR arbitration, if it passed an optional per-queue programmable bandwidth-limitation based on token-bucket mechanism (See Section 15.8.4 "Transmit Queue Bandwidth Limitation" on page 236). Therefore, only the queues that are below the per-queue bandwidth limitation would be considered in any WRR arbitration.

Configure the queue weight (0–255) by writing to the corresponding Transmit Queue Arbiter Configuration (TQAC) register (one of eight) and setting the WRRWGT bits [7:0] (Table 543 on page 624) to the desired weight value (measure transmit ports in 256 byte units).

As used here, the "WRR bandwidth" is the available transmit bandwidth (that is not consumed by fixed-priority ports traffic).

The WRR arbitration end result is calculated by dividing the WRR bandwidth between the WRR-queues according to each queue's WRRWGT/ (Sum of all WRRGTs of the WRR-queues that are not bandwidth limited).

When several WRRWGT combinations yield the same bandwidth distribution, the user must use a combination with the smallest WRRWGT value closest to the Maximum Transmit Unit (see Section 15.8.6 "Maximum Transmit Unit" on page 236).

The WRR bandwidth distribution is determined by counting the transmitted bytes from each WRR queue and limiting the schedule of queues that used up their bandwidth portion (WRRWGT) until all other queues finish transmission.

15.8.4 Transmit Queue Bandwidth Limitation

To implement a bandwidth limitation on transmit queues, specify the maximum available bandwidth for each queue in approximate ranges (2 Mbs–1 Gbs) in 1024 step resolution.

The bandwidth limitation is implemented by the Token Bucket per queue. The tokens are added to the 'Bucket' in a constant configurable rate and drained from the Bucket when the queue transmits. A queue is only allowed to transmit when the number of 'Tokens' is larger then Maximum Transmit Unit (MTU). It is possible to configure the queue Token-Bucket size that gives control over the amount of credit (silent time) that the queue is allowed to accumulate.

The queue 'Token-Rate' is programmed by setting the corresponding queues Transmit Queue Token Bucket Rate Configuration register's `TKNRT` bits [9:0] (Table 542 on page 624). `TKNRT` is filled to the desired value in 1/64 bit per clock cycle units.

Note

To disable a bandwidth limitation, set the `TKNRT` to its maximum value.

The queue Bucket size is programmed by writing to the corresponding queue's Transmit Queue Token Bucket Rate Configuration register and setting the `MTBS` bits [25:10] in a value of 256-byte units. The tokens accumulate in the bucked until the Maximum Token Bucket Size (MTBS) setting.

The user may examine or modify the current value of a queues token bucket by read/write to the corresponding queue's Transmit Queue Token Bucket Counter register's `TKNBKT` [29:0] with a value in 1/64-byte units.

15.8.5 Transmit Port Bandwidth Limitation

The transmit port implements a bandwidth limitation on all outgoing traffic. This applies to Fixed priority and WRR priority queues.

It is possible to specify the maximum available port bandwidth in approximate ranges (2 Mbs–1 Gbs) in 1024 step resolution. The bandwidth limitation is implemented by a port Token-Bucket. The tokens are added to the Bucket in a constant configurable rate and drained from the Bucket when the port transmits. The port is allowed to transmit only when the number of Tokens is bigger than the MTU. It is also possible to configure the port Token-Bucket size for the amount of credit (silent time) the queue is allowed to accumulate.

The port 'Token-Rate' is programmed by writing to the Port Transmit Token-Bucket Rate Configuration register and setting the `PTKNRT` bits [9:0] (Table 544 on page 625) to the desired value in 1/64-bit per clock cycle units. To disable bandwidth limitation, set `PTKNRT` to its maximum value.

The port 'Bucket' size is set by writing to the Port Maximum Token Bucket Size register's `PMTBS` bits [15:0] (Table 526 on page 614 in a value of 256-byte units. The tokens accumulate in the port bucket until the setting for `PMTBS` is met.

The user may examine or modify the current value of port Token-Bucket by reading/writing to the Port Transmit Token-Bucket Counter register's `PTKNBKT` bits [29:0] (Table 544 on page 625) with a value in 1/64-byte units.

15.8.6 Maximum Transmit Unit

The MTU is a configurable value common to all transmit ports and all WRR queues. It is used for bandwidth limitation implementation.

The MTU is programmed by setting the Maximum Transmit Unit register's MTU bits [5:0] (Table 525 on page 614) in 256-byte units.

15.9 Token Rate Configuration

The relation of the desired bandwidth (BW) limitation and Token Rate programmed value is:

- TokenRate[1/64 bit/cycle] = BW[Mbit/sec]*64/SystemClock[MHz].

Table 66 shows some examples of the Token Rate bandwidth configuration values.

Table 66: Token Rate Configuration Examples

Bandwidth [Mbps]	System Clock [MHz]	Token Rate [1/64 bit/cycle]
1.953	125	1
1000	125	512
1.953	62.5	2
1000	62.5	1024

As shown in Table 66, 10 bits for the Token Rate span 1 Gbps–2 Mbps.

15.10Network Interface (10/100/1000 Mbps)

The MV64360/1/2 can be connected to a Gigabit Ethernet network using either a 10-bit interface to the available Gigabit Ethernet transceivers to fiber, or to a GMII PHY (to 1000baseT copper or 1000baseX).

The MV64360 also supports different configurations of two or three ports depending on multiplexed-interface pins (see Section 26. “Pins Multiplexing” on page 343).

To support all speeds, the MV64360/1/2 includes several MAC blocks suited for 10, 100, and 1000 Mbps with the following considerations:

- Support for half-duplex (for 10 and 100 Mbps only) and full duplex (in all speeds).
- Backpressure option in half duplex (for MII mode only).
- Flow-control option in full-duplex.

Auto-Negotiation is supported for all interface modes:

- 10-bit according to 802.3Z clause 37.
- MII and GMIII according to 802.3ab draft 5.0 using MDC/MDIO pins.

When connected to a 10-bit transceiver (commonly available for Gigabit over fiber), the MV64360/1/2 also incorporates the TBI layer for 1000baseX, including the Auto-Negotiation.

15.10.1Gigabit Ethernet MAC

The MV64360/1/2 Gigabit MAC supports the following:

- Connection to GMII PHY (for 1000baseX or 1000baseT) or 10-bit interface to Gigabit Ethernet transceiver.
- 1000 Mbps full duplex.
- Standard 802.3Z flow-control in full duplex.

The MV64360/1/2 MAC performs all of the functions of the 802.3Z such as frame formatting, frame stripping, collision handling, deferral to link traffic, etc. The MV64360/1/2 ensures that any outgoing packet complies with the

802.3 specification in terms of preamble structure. The MV64360/1/2 transmits 56 preamble bits before the Start-of-Frame Delimiter.

15.10.2 GMII Interface

The transmit and receive operations are done in full duplex and implement the standard.

15.10.2.1 GMII Transmission in Full Duplex

When the MV64360/1/2 has a frame ready for transmission and the IPG counter has expired, frame transmission begins.

Note

The Carrier Sense (CRS) and Collision Detect (COL) input pins are ignored in this mode.

The data is transmitted via pins TxD[7:0] of the transmitting port and clocked on the rising edge of TxClkOut. At the same time, signal TxEn is asserted. The `TxEr` signal is always driven LOW as there is no carrier-extension required.

15.10.2.2 GMII Reception

Frame reception starts with the PHY's assertion of `RxDv` or `RxEr` (while the MV64360/1/2 is not transmitting). Once `RxDv` or `RxEr` is asserted, the MV64360/1/2 begins sampling incoming data on pins `RxD`[7:0] on the rising edge of the `RxClk`.

Note

The `RxDv` signal is high during reception of packet-data.

15.10.3 10-bit Interface

The MV64360/1/2 implements the TBI sublayer for 1000baseX. When operating in this mode, the GMII interface is internal to the MV64360/1/2.

The 10-bit interface includes:

- A transmit clock
- A 10-bit transmit code
- A dual receive clock
- A 10-bit receive code

The Auto-Negotiation function also has an optional Bypass mode. This mode allows support for devices lacking the Auto-Negotiation functionality, without user intervention. For more details see Section 15.12.3 "Auto-Negotiation Bypass Mode" on page 240.

15.10.4 10/100 MII Interface

The MV64360/1/2 MAC allows it to be connected to a 10 Mbps or 100 Mbps network. The MV64360/1/2 interfaces to an IEEE 802.3 10/100 Mbps MII compatible PHY device. The data path consists of a separate nibble-wide stream for both transmit and receive activities.

The MV64360/1/2 can switch automatically between 10- or 100-Mbps operation depending on the speed of the network. Data transfers are clocked by the 25-MHz transmit and receive clocks in 100-Mbps operation, or by 2.5-MHz transmit and receive clocks in 10-Mbps operation. The clock inputs are driven by the PHY. The PHY controls the clock rate based on its configuration or on the Auto-Negotiation function.

15.11Interface Mode Selection

During operation, the reset configuration Gigabit Ethernet Ports TBI/GMII Interface Mode Selection pins AD[22:21] (see Table 117 on page 348) select one of the following operational interfaces:

- GMII interface
- 10-bit (PSC) interface

The interface mode for each port is set independently. Bit AD[21] sets the mode for Port 0, and bit AD[22] sets the mode for Port 1.

If Auto-Negotiation is enabled:

- For 10-bit interface mode (by the Port Serial Control register's `An_Duplex` bit [2] and `An_FC` bit [2], see Table 519 on page 607), the TBI Auto-Negotiation function performs the negotiation. The logic fully implements the 802.3Z clause 37 specification. In addition, the MV64360/1/2 supports a Bypass mode.
- For GMII or MII interface mode (by the Port Serial Control register's `An_Duplex` bit [2] and `An_FC` bit [2]), the MDC/MDIO Auto-Negotiation takes place.

15.12Auto-Negotiation Modes

15.12.1Auto-Negotiation in MII/GMII Modes

The MV64360/1/2 implements the standard IEEE Auto-Negotiation, using the Serial Management Interface (SMI), for the following:

- Detect Link status
- Duplex: half- and full-duplex operation
- Flow-control for full-duplex
- Speed

Notes

- To implement speed Auto-Negotiation, set the Port Serial Control register `AN_Speed` bit [13] (Table 519 on page 607) to '0', to switch between GMII and MII modes.
- In 10-bit interface mode, set `AN_Speed` to '1'.
- The registers and bits referred to in this sub-Section (for example, registers 4, 5, and 15 and bit 1.8) are PHY Device registers.

The MV64360/1/2 continuously reads the PHY register 1 to establish the link status, and also to determine whether or not bit 1.8 in PHY register 1 is set.

When exiting from reset, or when the link changes from up to down, the MV64360/1/2 advertises its flow control ability (if Auto-Negotiation for flow control is enabled by the Port Serial Control register's `An_FC` bit).

The MV64360/1/2 reads register bit 1.8. If this bit is reset, then the PHY does not support 1000 Mbps.

If bit 1.8 is set, the PHY supports 1000 Mbps (but the speed may still resolve to 10 or 100 Mbps at the end) and register 15 exists. The MV64360/1/2 continues to read register 15 to determine whether the PHY is 1000baseX-capable or 1000baseT capable. If it is 1000baseX, then the MV64360/1/2 regards the multiplexed speed as 1000 Mbps only and follows the 802.3Z clause 37 rules (for register 4 and 5 format) for duplex and flow-control Auto-Negotiation. If it is 1000baseT capable, then the MV64360/1/2 follows the 802.3ab rules to resolve the speed, which could be 1000 Mbps (using GMII interface) or 10/100Mbps (using MII interface), the duplex mode and the flow control.

After Auto-Negotiation is complete, the MV64360/1/2 resolves negotiated modes of operation. These values update the Port Status register fields and affect the Network port operation.

15.12.2Auto-Negotiation in 10-bit Mode

The MV64360/1/2 implements the Auto-Negotiation state machine, per the IEEE standard 802.3Z clause 37.

The advertise capabilities do not include next page and unsymmetrical pause (unsymmetrical flow control).

The MV64360/1/2 advertises duplex and flow-control capabilities according to the Port Serial Control register's `An_Duplex` bit [2] and `An_Pause` bit [4]. When the link-up is resolved, the switch's operational modes are updated according these bits.

The state machine also supports a bypass mode (controlled by the Port Serial Control register's `AN_bypass_en` bit. This bit allows the MV64360/1/2 to support devices that do not implement the Auto-Negotiation function.

15.12.3Auto-Negotiation Bypass Mode

The IEEE standard Auto-Negotiation state machine, per the 802.3X Clause 37, requires that both sides support Auto-Negotiation before the link can be established. If one side implements the Auto-Negotiation function and the other does not, two-way communication is not established, unless Auto-Negotiation is manually disabled and both sides are configured to work in the same operational modes.

When the bypass mode is enabled, the MV64360/1/2 Auto-Negotiation state machine changes from the type specified in clause 37:

When entering the "Ability_Detect" state, a timer is started to count down with an initial value of 20 times the link-timer value. Since the link-timer value is ~10 ms, the "Ability_Detect" associated timer is ~200 ms.

If the timer expires and, during this period, the receive synchronization machine stayed in synchronization and did not report RUDI(INVALID) and the state-machine is still in the Ability_Detect state, this is interpreted as a sign that the other side is "alive" but cannot send configuration codes to perform Auto-Negotiation. Therefore, the state-machine moves to a new "Bypass_Link_Up" state. In this state, the MV64360/1/2 assumes a link up and the operational mode is set to the Port Serial Control register's `An_Duplex` and `An_FC`> values are at the time.

Note

Once the other device is replaced by a device that can perform Auto-Negotiation, the Auto-Negotiation is automatically restarted.

If the other device only transmits idles during this extended timer period, the bypass is performed. In this instance, configuration codes are not idles. Therefore, a regular Auto-Negotiation device does not allow the bypass to take place.

If the bypass was performed, the MV64360/1/2 reports this via the Port Status register's `Bypass_Activated` bit, together with the Interrupt stating link change. Therefore, management can recognize whether the link was resumed due to standard Auto-Negotiation or bypass.

15.13Data Blinder

The MII Serial Parameters register's `Datablind` bits [21:17] (Table 509 on page 602) set the time period during which the port does not look at the wire to decide to defer a pending transmission, due to receive activity.

15.14Inter-packet Gap

The Inter-packet Gap (IPG) is the idle time between any two successive packets from the same port. The default (from the standard) is 96ns.

Note

Marvell Technology does not recommend reducing the IPG setting in violation of the IEEE standards. Reducing the IPG can improve test scores but can create Ethernet compatibility problems.

Use the MII Serial Parameters (Table 509 on page 602) and GMII Serial Parameters (Table 510 on page 603) to set the IPG size.

15.15Illegal Frames

For undersized frames (with or without good CRC), the MV64360/1/2 discards all illegal frames. The frames are not passed to the CPU, regardless of address filtering, and the appropriate error MIB counters are incremented. An undersized frame is determined by the Minimum Frame Size.

Oversized frames (greater than the MRU) with or without bad CRC (bad checksum) are forwarded to the DMA queue with an error summary report in the Rx-descriptor.

15.16Partition Mode

Partition mode is relevant for 10 and 100Mbps operation in half duplex.

When in Partition mode, the port continues to transmit, but it does not receive. The `Partition` bit [9] in the Ethernet Port Status register (Table 521 on page 611) is set when the port is partitioned. The port is returned to normal operation mode when a good packet is seen on the wire.

Partition mode can only occur if it was enabled (see Section 15.16.1 Enabling Partition Mode).

15.16.1Enabling Partition Mode

The Partition mode is enabled by setting the Portx Configuration Extend register's `ParEn` bit [2] (Table 508 on page 602). The default value is Partition disabled.

To permit up to 61 retransmission attempts on the same frame, the user must also set PSCR `Retr_Forever` bit [11].

Note

After the device is enabled for switching, the `ParEn` bit value must not be modified (i.e. enabled or disabled).

15.16.2Entering Partition Mode

When Partition is enabled, the network port enters Partition mode when either of the following occurs:

- more than 61 consecutive collisions are seen on the port on the same frame, while the Port Serial Control register's (PSCR) `Retr_Forever` bit [11] is set (see Table 519 on page 607).
- more than 61 consecutive collisions occur on different frames, while PSCR `Retr_Forever` bit [11] is clear.

Once the network port is in Partition mode:

- If the interrupt is not masked, the MV64360/1/2 issues an interrupt to the CPU upon entering Partition state, and sets the Port Interrupt Cause register's `Partition` bit [21] (see Table 527 on page 615).
- The port continues to transmit its pending packets, regardless of the collision detection, and does not follow the usual Backoff algorithm. While ignoring the internal collision indication, additional packets pending for transmission are transmitted. This frees the port's transmit buffers, which are otherwise full. The assumption is that Partition is a system failure situation (bad connector/cable/station).

15.16.3Exiting from Partition Mode

The port exits from Partition mode when a good packet is seen on the wire. A good packet is declared if no collisions were detected on the packet. If the interrupt is not masked, the MV64360/1/2 issues an interrupt to the CPU upon exiting from Partition mode.

15.17Backpressure Mode

Only when the network port is operating in half-duplex, MII mode will the MV64360/1/2 implement a Backpressure algorithm.

The Backpressure algorithm is enabled by setting the Port Serial Control register's `Force_BP_Mode` bits [8:7] (Table 519 on page 607).

For a port in Backpressure mode, the MV64360/1/2 waits until the medium is idle and then transmits a JAM pattern for a programmable value of time. To program the period that the JAM pattern is transmitted, set the `JAM_LENGTH` bits [1:0]. The IPG between two consecutive JAM patterns (or between the last transmit and the first JAM) is programed using the `JAM_IPG` bits [6:2], for MII, or bits [4:2], for GMII. `JAM_LENGTH` and `JAM_IPG` are set in the MII Serial Parameters register (Table 509 on page 602) or the GMII Serial Parameters register (Table 510 on page 603).

Note

Use the MII Serial Parameters register for 10 Mbps and 100 Mbps speed and in the GMII Serial Parameters register for 1000Mbps speed.

When a port in Backpressure mode has a pending packet for transmission, it halts the transmission of the JAM pattern. The JAM pattern is halted for an IPG value set through the MII or GMII Serial Parameters registers' `IPG-JAM_TO_DATA` bits [11:7], for MII, or [7:5], for GMII. After the IPG is completed, the port transmits the packet. If the port remains in Backpressure mode, it resumes the JAM pattern transmission after an IPG set by the `JAM_IPG` bits, following the packet transmission.

15.18Flow Control

The MV64360/1/2 implements the 802.3X and 802.3Z flow control in full-duplex mode, including full Auto-Negotiation.

Auto-Negotiation for flow control is enabled for:

- The multiplexed interface
- PHYs that have SMI interface (MII or GMII PHYs)
- 10-bit interface mode by the internal Auto-Negotiation state-machine.

The behavior of the MV64360/1/2 is determined by the value in Port Status register `En_Fc` bit, Port Serial Control register (PSCR) `Force_FC_Mode` bit.

The CPU may write to the PSCR `Force_FC_Mode` bit when flow-control operation is enabled in Port Status register `En_Fc` bit (which may be result either of auto-negotiation resolution for flow control, or manual setting by the CPU to enable flow-control operation, which is then reflected in the Port Status register `En_Fc` bit).

The CPU must trigger the initiation of pause disable transmission when detecting that it cannot keep up with the received traffic (This is typically done by monitoring the queue filling process).

When the CPU suspects that it may not be able to provide enough resources to the port, it must trigger the beginning of flow control packets by writing '01' value to `Force_FC_Mode`. When resources are made available the CPU must again write a '00' value to `Force_FC_Mode` to trigger transmission of pause enable packet (see the more detailed description in Section 15.18.2 "Pause Transmit Operation" on page 244. The CPU response time to congestion cases would determine if and how many packets may be lost on receive.

The value in Port Status register `En_Fc` bit can be set from the following:

- CPU programming.
- Result of Auto-Negotiation for flow control according to 802.3X/802.3Z/802.3ab standards in all modes – MII, GMII, and 10-bit.

When in MII or GMII modes, and Auto-Negotiation for Flow Control is enabled, the MV64360/1/2 writes to the relevant advertisement register in the PHY on the following events:

- Exiting from reset.
- Upon link fail detection (Link changed from up to down).

When in 10-bit mode, the MV64360/1/2 advertises the Pause capability according to the clause 37 state-machine behavior.

Auto-Negotiation for flow control for 1000BASE-X PHY advertises that the MV64360/1/2 supports Symmetric Flow-control only according to the 802.3Z standard (section 37.2.3.2).

The advertised ability of Pause support depends on the setting of the Port Serial Control register's `Pause_Adv` bit [4] (Table 519 on page 607) as follows:

- When set, the MV64360/1/2 advertises symmetric capability for Pause.
- When reset, the MV64360/1/2 advertises No Pause capability.

15.18.1Pause Receive Operation

When the MV64360/1/2 receives a Pause packet, it avoids transmitting a new packet to the port for the period of time specified in the received Pause packet.

The pause quantum is 512 bits regardless of the operation speed or the duration of the slot time.

A received packet is recognized as flow control if it was received without errors and is one of the following:

- DA = 01-80-C2-00-00-01 and type=88-08 and MAC_Control_Opcode=01.

A packet received by the MV64360/1/2 from the network port that is identified as a Pause packet is always discarded, even if the Pause function is disabled.

15.18.2Pause Transmit Operation

For enabling Pause Transmit operation or either enabling or disabling the Port Status register `En_Fc` bit must be in the active state.

It is the CPU responsibility in sensing that packets are in danger of being dropped by the receive port, according to the dynamic availability of resources. One way of doing it is monitoring how much of the descriptor chain is filled up by the port and how much is left. Another aspect is memory bandwidth should be allocated to the port via the crossbar "pizza arbiter" to avoid bandwidth shortage for the gigabit port.

Note

User should note that this mechanism does not provide hardware guarantee of zero frame-loss as it depends on CPU functionality in triggering it dynamically.

When the CPU suspects that it may not be able to provide enough resources to the port, it must trigger the beginning of flow-control, pause-disable packets transmission by writing ’01’ value to the PSCR `Force_FC_Mode` bits [6:5]. The transmit port will schedule transmission of a pause-disable frame (timer=0xFFFF) at the next possible frame boundary and will automatically retransmit it at least every 4.2 msec (GMII/10-bit), 42 msec (MII at 100 MB), or 420 msec (MII at 10 MB) as long as the value in the `Force_FC_Mode` field remains ’01’.

The other link partner is expected to stop packet transmission upon receiving the flow-control disable packets, and the retransmission of them guarantees refreshing that indication continuously.

When resources are made available, the CPU must write a ’00’ value to the PSCR `Force_FC_Mode` bits [6:5]. This will trigger transmission of a single pause enable packet (timer = 0x0000), which would enable the other link partner to resume packet transmission.

When transmitting a pause packet, the port address is put into the source address field. The 48-bit port address is located in the MAC Address Low and the MAC Address High registers.

Note

When the link goes down, the PSCR `Force_FC_Mode` bits [6:5] are always reset to ’00’ (No Pause disable frames are sent).

15.19MII/GMII Serial Management Interface (SMI)

The MV64360/1/2 MAC contains a Serial Management Interface (SMI) for a MII or GMII compliant PHYs.

This allows control and status parameters to be passed between the MV64360/1/2 and the PHY (parameters specified by the CPU) using one serial pin (MDIO) and a clocking pin (MDC), reducing the number of control pins required for PHY mode control. Typically, the MV64360/1/2 continuously queries the PHY device for the link status, without CPU intervention. The PHY addresses for the link query are programmable in the PHY Address register (Table 492 on page 592).

A CPU connected to the MV64360/1/2 can write/read to/from all PHY addresses/registers. The SMI allows the CPU to have direct control over an MII or GMII compatible PHY device via the SMI register (Table 493 on page 593). This control allows the driver software to place the PHY in specific modes such as Full-Duplex, Loop-back, Power-Down, or 1000-speed selection. It also helps control the PHY device's Auto-Negotiation function, if it exists. The CPU writes commands to the SMI register and the MV64360/1/2 reads or writes control/status parameters to the PHY device via a serial, bi-directional data pin called MDIO. These serial data transfers are clocked by the MV64360/1/2 MDC clock output.

15.19.1SMI Cycles

The SMI protocol consists of a bit stream that is driven or sampled by the MV64360/1/2 on each rising edge of the MDC clock. The SMI frame, bit-stream format starts with PRE and ends with IDLE. Its various steps are described in Table 67.

Table 67: SMI Bit Stream Format

	PRE	ST	OP	PhyAd	RegAd	TA	Data	IDLE
READ	1...1	01	10	AAAAA	RRRRR	Z0	D.D(16)	Z
WRITE	1...1	01	01	AAAAA	RRRRR	10	D.D(16)	Z

- PRE (Preamble): At the beginning of each transaction, the MV64360/1/2 sends a sequence of 32 contiguous logic '1' bits on the MDIO with 32 corresponding cycles on the MDC to provide the PHY with a pattern that it can use to establish synchronization.
- ST (Start of Frame): A Start-of-Frame pattern of 01.
- OP (Operation Code): 10 - Read; 01 - Write.
- PhyAd (PHY Address): A 5-bit address of the PHY device (32 possible addresses). The first PHY Address bit transmitted by the MV64360/1/2 is the MSB of the address.
- RegAd (Register Address): A 5-bit address of the PHY register (32 possible registers in the PHY). The first register address bit transmitted by the MV64360/1/2 is the MSB of the address. The MV64360/1/2 always queries the PHY device for status of the link by reading register 1, bit 2.
- TA (Turn Around): The turnaround time is a 2-bit time spacing between the `RegAd` field and the `Data` field of the SMI frame to avoid contention during a read transaction. During a read transaction the PHY must not drive MDIO in the first bit time and drive '0' in the second bit time. During a write transaction, the MV64360/1/2 drives a '10' pattern to fill the TA time.
- Data (Data): The data field is 16-bits long. The PHY drives the data field during read transactions. The MV64360/1/2 drives the data field during write transactions. The first data bit transmitted and received is bit 15 of the PHY register being addressed.
- IDLE (Idle): The IDLE condition on MDIO is a high impedance state. The MDIO driver is disabled and the PHY must pull up the MDIO line to a logic '1'.

15.19.2SMI Timing Requirements

When the MDIO signal is driven by the PHY, it is sampled by the MV64360/1/2 synchronously with respect to the rising edge of MDC. Per the IEEE 802.3 specification, the MDC-to-output delay must be a minimum of 0ns and a maximum of 300ns as shown in Figure 45. Further, when the MDIO signal is driven by the MV64360/1/2, the MV64360/1/2 provides a minimum of 10ns of setup time and minimum of 10ns of hold time as shown in Figure 46.

Figure 45: MV64360/1/2 MDIO Output Delay

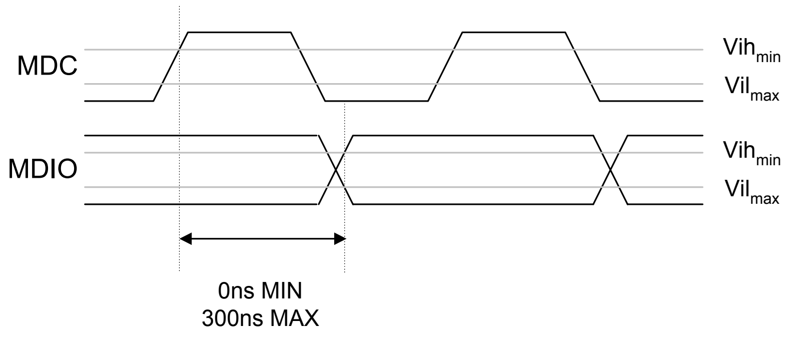
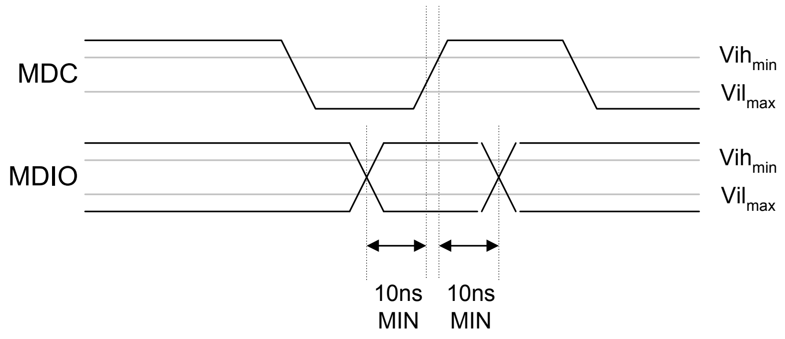


Figure 46: MV64360/1/2 Required MDIO Setup and Hold Time



15.20 Link Detection and Link Detection Bypass (ForceLinkPass*)

Note

Only supported in the MV64360 and MV64361 devices. Furthermore, references to Port 2 only applies to the MV64360.

Typically, the MV64360 and MV64361 devices continuously queries the PHY device for its link status, without CPU intervention. The PHY address used for the link query is determined by the PHY Addresses register, and it is programmable, where the default value is ’8’ for Port 0, ’9’ for Port 1, and ’A’ for Port 2 (out of a possible 32 addresses). The MV64360 and MV64361 devices reads register 1 from PHY and updates the internal link bits according to the value of bit 2 of register 1. In the case of “link is down” (bit 2 is ‘0’), that port enters link test fail state. In this state, all of the port’s logic is reset. The port exit from link test fail state only when the “link is up”, bit 2 of register 1 is read from the port’s PHY as ‘1’.

The MV64360 and MV64361 devices offers the option to disable the link detection mechanism by forcing the link state of the interface to the link test pass state. This is done by forcing the register bit, and then the link status of the port remains in the "link is up" state regardless of the Interface-PHY's link bit value. The link status of the Interface-PHY can be read through the SMI from the PHY devices (register 1, bit 2).

15.20.1Force_Link_Fail

The PSCR register's `Force_Link_Fail` bit (bit 10) has the default value of forcing the link detection on each port to link down. The user *must* set this bit, in order to get the true link status of the port, and in order to enable the port link indication to go up.

The user must not program the `Force_Link_Fail` bit and the `Force_Link_Pass` bit to be set at the same time.

15.21Network Management Interface Counters

The MV64360/1/2 incorporates a set of management counters.

For a complete description refer to Section 15.22 "Port MIB Counters" on page 247.

15.22Port MIB Counters

The MAC MIB Counters provide the necessary counters that support MAU, 802.3 and EtherLike MIB. Each port has a set of counters, which reside in consecutive address space. Some counters are 64-bits wide.

The counters are meant to provide management software to support:

1. IEEE 802.3 DTE Management objects
2. Ethernet-like interface MIB: RFC 2665
3. Interface MIB: RFC 2863
4. Remote Network Monitoring (RMON) groups 1-4: RFC 2819

Note

The MAC counters are not intended to be used for Bridge MIB nor for SMON MIB.

15.22.1Definitions

The following table summarizes the terms used in the definition of the counters.

Table 68: Definitions for MAC Counters

Term	Definition
Collision Event	A collision has been detected before 576-bit times into the transmitted packet after TxEn is asserted. Relevant to 10 Mbps and 100 Mbps speeds in half-duplex mode only.
Late Collision Event	A collision has been detected after 576-bit times into the transmitted packet after TxEn. Relevant to 10 Mbps and 100 Mbps speeds in half-duplex mode only.

Table 68: Definitions for MAC Counters (Continued)

Term	Definition
Excessive Collision Event	When a packet to be transmitted suffers from 15 consecutive collision events, therefore, it ought to be dropped according to the IEEE 802.3 specification. This does not occur if the PSCR's <`Retr_forever`> bit is set. Relevant to 10-Mbps and 100-Mbps speeds in half-duplex mode only.
MRU	Maximal Receive Unit: A programmable parameter that sets the maximal length of a valid received packet.
Rx Error Event	The Receive Error signal/symbol was asserted while a frame is received.
CRC Error Event	This event occurs whenever an Ethernet frame is received and the following conditions are satisfied: 1. Packet data length is between the Minimum Frame Size - and the MRU byte size inclusive (that is, it is a valid packet data length according to the IEEE standard). 2. Packet has an invalid CRC. 3. Collision Event has not been detected. 4. Late Collision Event has not been detected. 5. Rx Error Event has not been detected.
Undersize packet	An Ethernet frame satisfying *all* of the following conditions: 1. Packet length is less than Minimum Frame Size bytes. 2. Collision Event has not been detected. 3. Rx Error Event has not been detected. 4. Packet has a valid CRC.
Fragment	An Ethernet frame satisfying *all* of the following conditions: 1. Packet data length is less than 64 bytes, OR a packet without a Start Frame Delimiter (SFD) and the packet is less than 64 bytes in length. 2. Collision Event has not been detected. 3. Rx Error Event has not been detected. 4. Packet has an invalid CRC.
Oversize packet	An Ethernet frame satisfying *all* of the following conditions: 1. Packet length is more than the MRU byte size. 2. Collision Event has not been detected. 3. Late Collision Event has not been detected. 4. Rx Error Event has not been detected. 5. Packet has a valid CRC.
Jabber	An Ethernet frame satisfying *all* of the following conditions: 1. Packet data length is greater than the MRU. 2. Packet has an invalid CRC. 3. Rx Error Event has not been detected.
Tx Error Event	An internal error event in the transmit MAC. This is a very rare situation and when it happens, it means that there the system is misconfigured.
Bad frame	An Ethernet frame that has one of the following conditions met: CRC Error Event, Undersize, Oversize, Fragments, Jabber, Rx Error event and Tx Error Event.

Table 68: Definitions for MAC Counters (Continued)

Term	Definition
MAC Control Frame	An Ethernet frame that is not a bad frame and has a value of 88-08 in the EtherType/ Length field.
Flow Control Frame	A MAC Control Frame with an opcode equal to 00-01.
Good Flow Control Frame	A flow control frame with: 1. MAC Destination equal to 01-80-C2-00-00-01 2. 64-byte length
Bad Flow Control Frame	All flow control frames that are not good flow control frames
Good frame	An Ethernet frame that is not a bad frame NOR a MAC Control frame

The following figures illustrate the terms defined above:

Figure 47: Ethernet Frame Classification

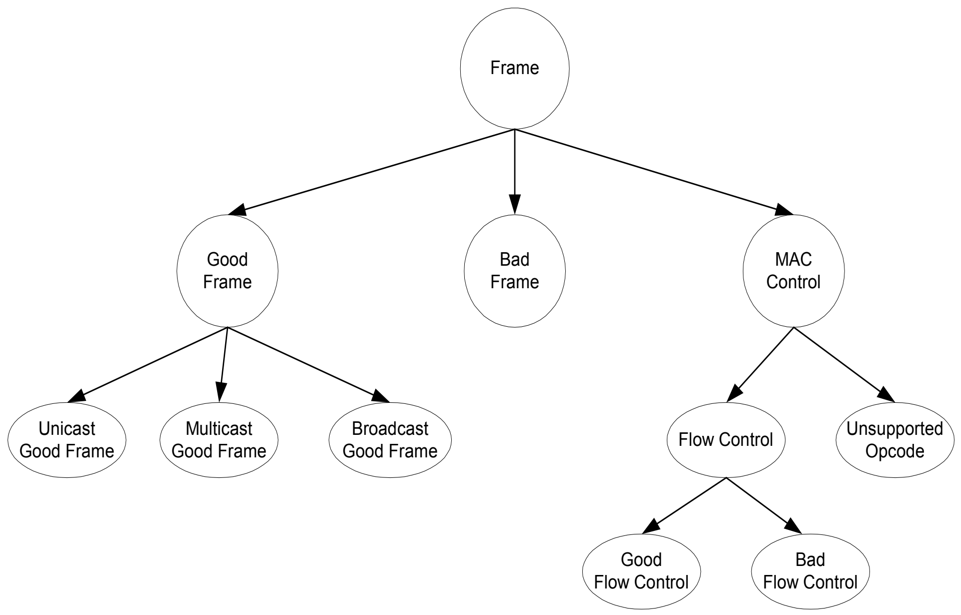
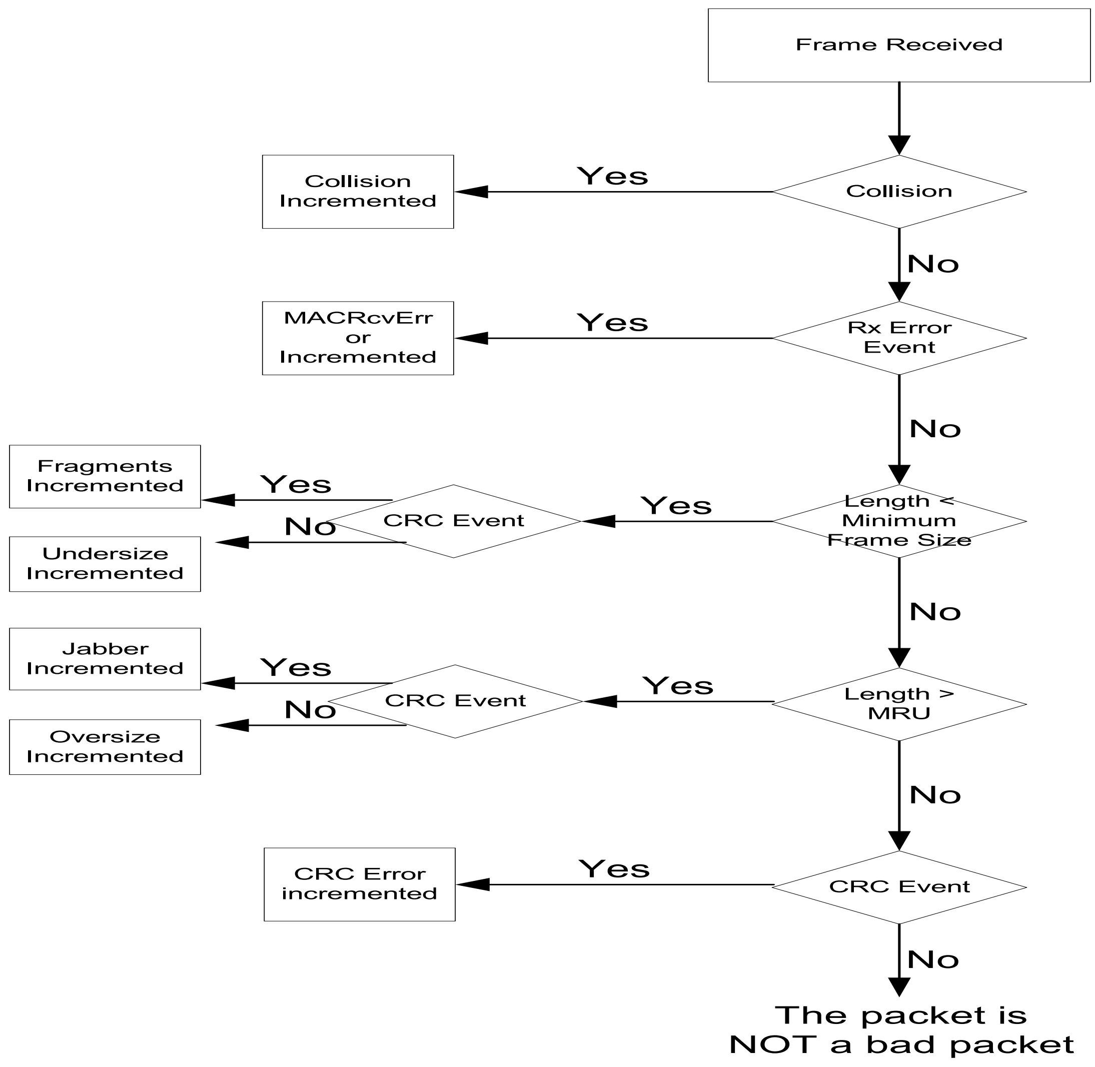


Figure 48: Bad Frame Procedure



15.22.2Per Port Counters

The counters initialize to ’0’ immediately after reset. Most counters are 32-bits, the Good Bytes received and Bytes Sent are 64-bit counters that should be read in two separate cycles.

Note

The 64-bit counters should be read from the low address and the next read from the MIB counters must be from the high address of the same counter. The MIB interface will always return the high counter data on a read from the MIBs following a read from the low address of a 64-bit counter.

Refer to Table 492 on page 592 for the address offsets of all the counters for the port. Within the counters block, the counter offset is in the address bits [6:0].

In addition to the per port counters, in Table 507 on page 600, there are some additional counters that count filtered frames for reasons like MAC address lookup results, called Port Overrun Frame Counter and Port Rx Discard Frame Counter registers (SeeTable 535 on page 621 and Table 534 on page 621, respectively). In conjunction with the counters block they provide total frames received information.

Section 16. Multi-Protocol Serial Controllers

The MV64360/1/2 includes two MPSCs that support:

- Bit oriented protocols (e.g. HDLC).
- Byte oriented protocols (e.g. BISYNC).
- Transparent protocols.
- The UART (Start/Stop) mode.

The two MPSCs can operate independently and up to a guaranteed bit rate of 55Mbps.

The transmit/receive data is organized in buffers, which are handled by a link of descriptors. Data and descriptors can be placed in any of the MV64360/1/2 interfaces (DDR SDRAM, integrated SRAM...). Data and descriptors movement is executed by dedicated SDMAs.

Note

An MPSC can also run as a simple UART debug port, without using the SDMAs.

The two MPSCs and their SDMAs are integrated in the MV64360/1/2 Serial Communication unit (or Cunit). This unit also includes the two baud rate generators, and the TWSI serial port.

16.1 MPSCs Signaling

The two MPSCs pins are multiplexed on the MV64360/1/2 Multi Purpose Pins (see Section 26. “Pins Multiplexing” on page 343). Table 69 summarizes the serial port pinout.

Table 69: MPSC Port Pinout

Pin	Type	Description
TxD	O	Serial transmit data.
RxD	I	Serial receive data.
RTS	O	Request To Send. Indicates that the MPSC is ready to transmit data.
CTS	I	Clear To Send. Indicates to the MPSC that data transmission may begin.
SCLK/OSCLK	I/O	Available as both transmit and receive clock. Also, serves as one of the input clocks to the baud generators.
		When the MPSC is not using SCLK as it’s Rx clock, Rx clock output.

Table 69: MPSC Port Pinout

Pin	Type	Description
TSCLK/OTSCLK	I/O	Available as the MPSC Tx clock when separate. Rx and Tx clocks are needed. This clock also serves as one of the input clocks to the baud rate generators.
		Tx clock output, when the MPSC is not using TSCLK as it's Tx clock.

The MPSCs' receive and transmit clocks use the baud rate generators or serial clock input signals. The routing of these signals are defined in the Rx Clock Routing register (RCR) and the Tx Clock Routing register (TCR). See H.5 "MPSCs Clocks Routing Registers" on page 642.

If using the BRG to generate the MPSC Rx clock, SCLK is an output, driving the Rx clock. If using BRG or SCLK to generate the MPSC Tx clock, TSCLK is an output, driving the Tx clock.

MPSC routing to the MPP pins is controlled via the MPSC Routing register (see Table 570 on page 642).

To enable an MPSC, the MRR register's `MR0` bits [2:0] and `MR1` bits [8:6] must be set to '0' and the MPPs must be assigned to act as MPSC signals, via the MPP Control register, see Appendix P. "Pins Multiplexing Interface Registers" on page 696.

16.2 Digital Phase Lock Loop

Each MPSC has a dedicated transmit and receive digital phase lock loop (DPLL).

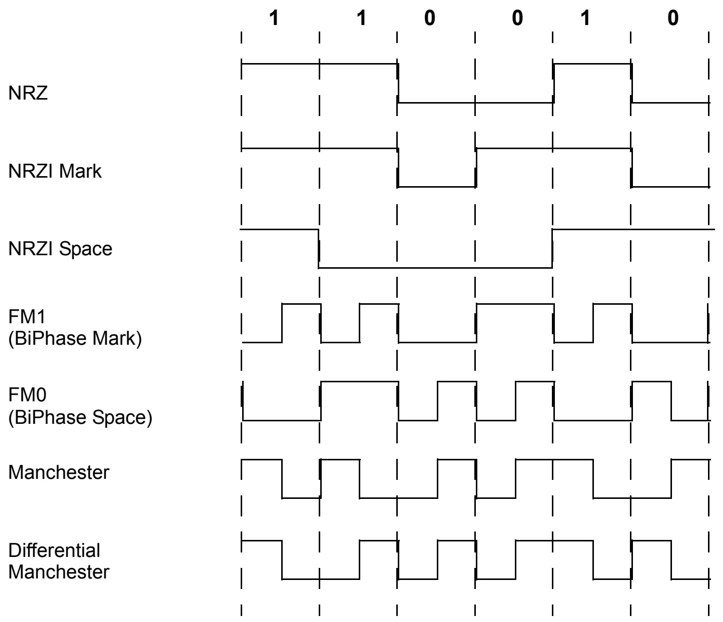
The transmit DPLL encodes the transmit bit stream to the selected code and monitors the transmit clock for glitches. If a clock glitch is detected and the Glitch Detect Enable (GDE) bit in the Main Configuration register (MMCR) is set to '1', a maskable interrupt is generated.

The receive DPLL decodes the incoming bit stream according to the selected mode. If a code violation is detected (for example, no transition in Manchester code) the DE (Decoding Error) in the receive descriptor is set. The receive DPLL also performs clock recovery from the incoming bit stream and monitors the receive clock for glitches. If a clock glitch is detected and the Glitch Detect Enable (GDE) bit in the Main Configuration register (MMCR) is set to '1', a maskable interrupt is generated.

16.2.1 Data Encoding/Decoding

Figure 49 shows the data encoding and decoding schemes The MV64360/1/2 DPLL supports.

Figure 49: MPSC DPLL Encoding/Decoding Schemes



16.2.2 DPLL Clock Source

Each received DPLL uses the MPSC receive clock input and each transmit DPLL uses the MPSC transmit clock input as its source clock.

Note

The MV64360/1/2 DPLLs can accept a clock source of up to 83MHz. This allows the MV64360/1/2 to have a bit rate of up to 5MHz using a 16X clock rate scheme.

16.2.3 Receive DPLL Clock Recovery

When a MPSC is programmed to work in UART Asynchronous mode, the DPLL encoding must be set to NRZ and the clock sampling rate to x8, x16, or x32 of the bit rate. The receive DPLL recognizes a start bit and synchronizes the clock to it.

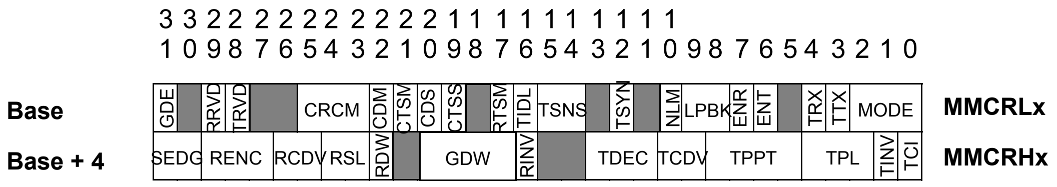
When not synchronized, the DPLL hunts for a start bit or edge. In UART mode, the DPLL hunts for start bit. In HDLC BISYNC and Transparent mode, the DPLL hunts for an edge. If hunting for a start bit (UART), the DPLL hunts for a falling edge, assuming it to be the beginning of a start bit. It then samples RxD at the middle of the bit, calculated from the falling edge of the start bit (8 ticks in x16 mode), to see that it is still ‘0’. If not, it is considered noise. A modulo 16 counter (for a 16x over-sampling rate) generates the receive clock RCLK.

In HDLC, BISYNC, and Transparent modes, the DPLL tries to lock itself on the transitions of the receive bit stream. When synchronization is achieved, the DPLL continuously monitors for rising and falling edges as defined in the MPSC Main Configuration Register (MMCR). When detecting an edge, the edge-compare logic gives the counter shift_left or shift_right commands to maintain a lock on the received data.

16.3 MPSCx Main Configuration Register (MMCRx)

Each MPSC has an MPSC Main Configuration Register. These are 64 bit register used to configure common MPSC features. It is protocol independent. The MMCRx consists of two 32 bits registers, MMCRHx and MMCRLx (see Table 575 on page 646 and Table 576 on page 650).

Figure 50: MPSC Main Configuration Register (MMCRx)



The following table summarizes the relationship between the TIDL and RTSM

Table 70: TIDL/RTSM Relationship

RTSM/TIDL	TxD	RTS#	TxD	RTS#
00	'1' Not Encoded	1	Data Encoded	0
01	'1' Encoded	1	Data Encoded	0
10	Flags/Not Encoded	0	Data Encoded	0
11	Flags/Encoded	0	Data Encoded	0

16.4 MPSCx Protocol Configuration Registers (MPCRx)

Each MPSC has a dedicated Protocol Configuration Register (MPCRx).

The MPCRx registers are located at base+08 relative to the corresponding MPSC Main Configuration Register (MMCRx). The functionality of the MPCRx is protocol dependent. Detailed descriptions of the MPCRs are given in the following protocol sections.

16.5 Channel Registers (CHxRx)

Each MPSC and ethernet controller has ten dedicated Channel Registers (CHxRx) to program the MPSC or ethernet controller.

The CHxRx registers are located at base+0xC0 through base+0x30 relative to the corresponding MPSC Main Configuration Register (MMCRx). The functionality of the CHxRx is protocol dependent. Detailed descriptions of the CHRs are given in the following protocol sections.

16.6 HDLC Mode

16.6.1 HDLC Receive/Transmit Operation

In HDLC mode, an MPSC performs the following protocol functions:

- Flag generation and stripping
- Bit stuffing and stripping
- Address recognition (up to 16 bit addresses)
- CRC generation and checking
- Line condition monitoring
- LocalTalk preamble generation
- LocalTalk trailing abort generation

Figure 51: Typical HDLC Frame

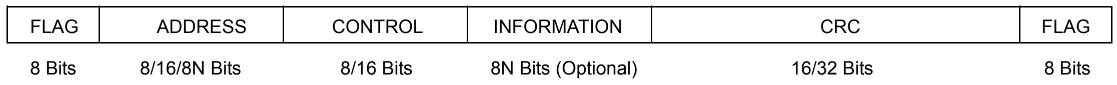
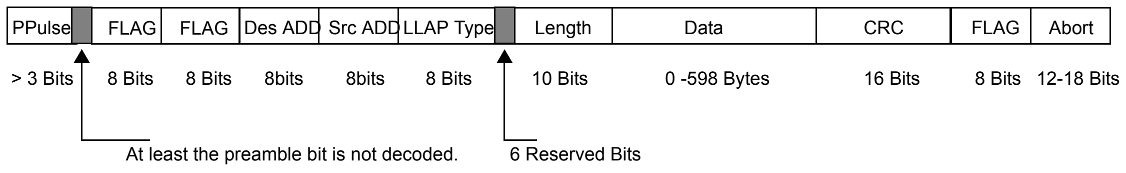


Figure 52: Typical LocalTalk Frame



16.6.2 SDMAx Command/Status Field for HDLC Mode

When an MPSC is in HDLC mode, the Command/Status field in the corresponding SDMAx descriptor has the following format:

Table 71: SDMAx Command/Status Field for HDLC Mode

Bit	Rx - Function	Tx - Function
31	O -Owner	O - Owner
30	AM - Auto Mode	AM - Auto Mode
29:24	Reserved	Reserved
23	EI - Enable Interrupt	EI - Enable Interrupt
22	Reserved	GC - Generate CRC
21:18	Reserved	Reserved
17	F - First	F - First

Table 71: SDMAx Command/Status Field for HDLC Mode (Continued)

Bit	Rx - Function	Tx - Function
16	L - last	L - Last
15	ES - Error Summary ES = CE || CDL || DE || NO || ABR || OR || MFL || SF	Error Summary ES = CTSL || UR || RL[1]
NOTE: “||” means logical OR.		
14	Reserved	Reserved
13:10	Reserved	RC-Retransmit Count (LAN HDLC mode only)
9	Reserved	COL - Collision Occurred
8	SF - Short Frame	RL - Retransmit Limit Error
7	MFL - Max Frame Length Err	Reserved
6	OR - Data Overrun/Residue[2]	UR - Data Underrun
5	Residue[1]	Reserved
4	ABR - Abort Sequence/Residue[0]	Reserved
3	NO - Non Octet Frame	D-deferred. Transmission was deferred due to busy channel.
2	DE - Decoding Error	Reserved
1	CDL - CD Loss	CTSL - CTS Loss
0	CE - CRC Error	Reserved

16.6.3 MPSCx Protocol Configuration Register (MPCRx) for HDLC

Figure 53: MPSCx Protocol Configuration Register (MPCRx) for HDLC

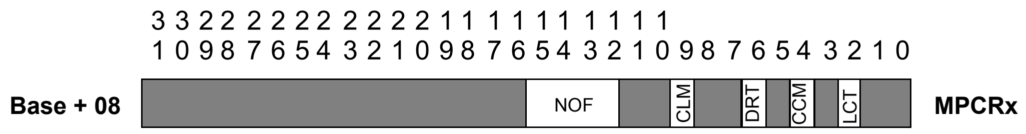


Table 72: MPSCx Protocol Configuration (MPCx) for HDLC
Offset: MPSC0 0x8008, MPSC1 0x9008

Bits	Field	InitVal	Function
1:0	Reserved	0x0	Reserved.

Table 72: MPSCx Protocol Configuration (MPCx) for HDLC (Continued)
Offset: MPSC0 0x8008, MPSC1 0x9008

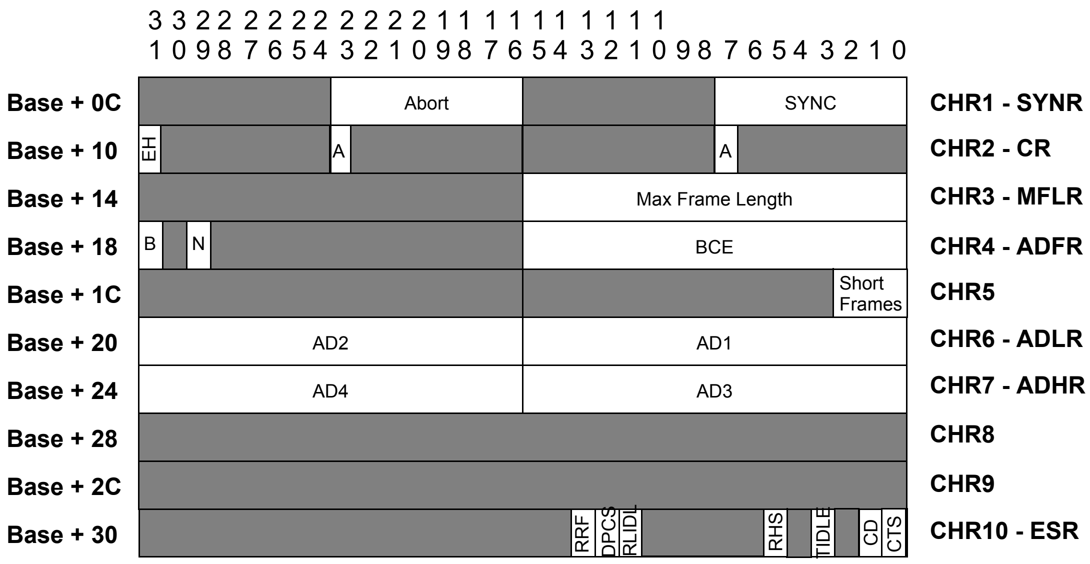
Bits	Field	InitVal	Function
2	LCT	0x0	Local Talk When set, the following LocalTalk support is added to the HDLC controller: • Two abort sequences will be generated at the end of frame following its closing flag. • A preamble will be generated. No encoding will be done for the last preamble bit When working with LocalTalk, the FM0 Encoding Scheme should be set by writing ‘010’ to RENC and TDEC in the MMCRx. The user should also set TPPT to 0xF and TPL to ‘1’ (one byte preamble). The last preamble bit is not decoded. This must be done for LocalTalk RTS frames. Setting TPL to ‘0’ leads to a frame without preamble. This can be used with LocalTalk data frames. Setting TPL to other values leads to unpredictable results.
3	Reserved	0x0	Reserved.
4	CCM		CRC Compliance Mode. In HDLC, the Tx side uses bit stuffing to prevent a data/CRC pattern from looking like an HDLC control flag. CCM tells the Rx side how to handle frames that were received with mistakes in bit stuffing, when they occur immediately before the end flag. This is a borderline condition that may or may not present a problem in actual systems. 0 = Compatible Mode If the Rx side receives a frame that is missing a stuffed bit that is supposed to be immediately before the End Flag, it marks in the descriptor that the frame has a good CRC and passes the good CRC along to the buffer. 1 = Compliance Mode If the Rx side receives a frame that is missing a stuffed bit that is supposed to immediately proceed the End Flag, it marks in the descriptor that the frame has a bad CRC and passes the problematic CRC to the buffer.
5	Reserved	0x0	Reserved.
6	DRT	0x0	Disable Rx on Tx When DRT is set to ‘1’ the Rx path is closed during Tx. This is useful in multidrop configurations when a user doesn’t want to receive its own frames.
8:7	Reserved	0x0	Reserved.
9	CLM	0x0	Collision Mode When set to ‘1’, the MPSC transceiver tries to retransmit a frame after a CTS lost. This mode allows automatic collision resolution for an ISDN LAP-D type channel.
11:10	Reserved	0x0	Reserved.

Table 72: MPSCx Protocol Configuration (MPCx) for HDLC (Continued)
Offset: MPSC0 0x8008, MPSC1 0x9008

Bits	Field	InitVal	Function
15:12	NOF	0x1	Number of Flags Specifies the number of flags transmitted between consecutive frames. Setting NOF to '0' specifies shared flag mode. In shared flag mode, the closing flag of a frame is used as the opening flag of the following frame. This setting also puts the receiver in back-to-back mode. The default value is '1'.
31:16	Reserved	0x0	Reserved.

16.6.4 Channel Registers (CHxRx) for HDLC Mode

Figure 54: Channel Registers (CHxRx) for HDLC



Unless otherwise is specified:

- '1' means set.
- '0' means not set.
- '0' is the default value after reset.

Table 73: CHR1 - Sync/Abort (SYN)
Offset: MPSC0 0X800C, MPSC1 0X900C

Bits	Field	InitVal	Function
7:0	SYNC	0x7e	Holds the synchronization pattern for the receive machine and opening/closing flag/sync-pattern for the transmit machine.

Table 73: CHR1 - Sync/Abort (SYN)
Offset: MPSC0 0X800C, MPSC1 0X900C

Bits	Field	InitVal	Function
15:8	Reserved	0x0	Reserved.
23:16	Abort_Pattern	0xFE	The abort pattern is transmitted upon receiving an abort command.
31:24	Reserved	0x0	Reserved.

Table 74: CHR2 - Command (CR)
Offset: MPSC0 0x8010, MPSC1 0x9010

Bits	Field	InitVal	Function
6:0	Reserved	0x0	Reserved.
7	A	0x0	Abort Transmission Abort transmission immediately and go to IDLE. The descriptor is not closed or incremented. **NOTE:** Command is not synchronized to byte.
22:8	Reserved	0x0	Reserved.
23	A	0x0	Abort Reception Abort receive immediately and go to IDLE. The descriptor is not closed or incremented. The processor must issue enter hunt command after abort command in order to enable reception. The bit is cleared upon entering IDLE state. After executing an Abort Reception, the CPU must disable the Tx SDMA channel. The CPU then needs to execute a normal initialization process to the MPSC.
30:24	Reserved	0x0	Reserved.
31	EH	0x0	Enter Hunt Upon receiving the Enter Hunt command, the receive machine moves to HUNT state and continuously searches for an opening flag. If enter hunt mode command is issued during frame reception, the current descriptor is closed with CRC error. The EH bit is cleared upon entering Hunt state. **NOTE:** The reception process for this purpose begins after proper address recognition is allowed. Before achieving an address match, the receiver goes to Enter Hunt state without closing the descriptor.

Note

The Main Configuration Register's `ET` bit [6] must be set to '1' before issuing the following Transmit Demand, Stop Transmission, or Abort Transmission commands, see Table 575 on page 646.

The Main Configuration Register's `ER` bit [7] must be set to '1' before issuing the Enter Hunt or Abort Reception commands.

When the `ET` or `ER` bits are de-asserted, the MPSCx transmit/receive channel is in low power mode (NO CLOCK). Issuing one of the above commands in this state leads to unpredictable results.

Table 75: CHR3 - Maximum Frame Length (MFL)
Offset: MPSC0 0x8014, MPSC1 0x9014

Bits	Field	InitVal	Function
15:0	FLBR	0xFFFF	Frame Length Buffer Register Holds the maximum allowed frame length. When a frame exceeds the number written in the FLBR, the remainder of the frame is discarded. The HDLC controller waits for a closing flag and then returns the frame status with bit 7 (MFLE) set to '1'.
31:16	Reserved	0x0	Reserved.

Table 76: CHR4 - Address Filtering (ADF)
Offset: MPSC0 0x8018, MPSC1 0x9018

Bits	Field	InitVal	Function
15:0	BCE	0x0	Bit Comparison Enable Bits Setting '1' in one of the BCE bits enables the address comparison for this bit: • For 16-bit LAP-D like address recognition, write 0xFFFF in ADFR. • For 8-bit HDLC/LAP-B like address recognition, write 0x00FF in ADFR. • For reception of a predefined address group, write '0' to the appropriate bits to disable address comparison on these bits.
28:16	Reserved	0x0	Reserved.
29	N	0x0	Null Enable Enables the reception of HDLC NULL address (0x0000 or 0x00 depending on the BCE setting)
30	Reserved	0x0	Reserved.
31	B	0x0	Broadcast Enable Enables the reception of HDLC broadcast address (0xFFFF or 0xFF, depending on the BCE setting).

Table 77: CHR5 - Short Frame (SHF)
Offset: MPSC0 0X801C, MPSC1 0X901C

Bits	Field	InitVal	Function
2:0	SHFR	0x0	Short Frame Register Setting SHFR to '1' enables the Short Frame Error report. Short Frames are frames with byte count less than 3+SHFR.
31:3	Reserved	0x0	Reserved.

Table 78: CHR6 - Address 1 and 2 (ADL)
Offset: MPSC0 0x8020, MPSC1 0x9020

Bits	Field	InitVal	Function
15:0	AD1	0x0	Address 1 A 16-bit address that can be used for receive address recognition.
31:16	AD2	0x0	Address 2 A 16-bit address used for receive address recognition.

Table 79: CHR7 - Address 3 and 4 (ADH)
Offset: MPSC0 0x8024, MPSC1 0x9024

Bits	Field	InitVal	Function
15:0	AD3	0x0	Address 3 A 16-bit address that can be used for receive address recognition.
31:16	AD4	0x0	Address 4 A 16-bit address that can be used for receive address recognition.

Table 80: CHR8 - Reserved
Offset: MPSC0 0x8028, MPSC1 0x9028

Bits	Field	InitVal	Function
31:0	Reserved	0x0	Reserved. **NOTE:** Do not access this register in the HDLC mode.

Table 81: CHR9 - Reserved
Offset: MPSC0 0X802C, MPSC1 0X902C

Bits	Field	InitVal	Function
31:0	Reserved	0x0	Reserved. **NOTE:** Do not access this register in the HDLC mode.

The ESR register holds information on the transmit/receive channel condition.

CHR10 can be read by the CPU for channel condition resolution. Some changes in the channel condition can generate maskable interrupts, as shown below.

Table 82: CHR10 - Event Status (ES)
Offset: MPSC0 0x8030, MPSC1 0x9030

Bits	Field	InitVal	Function
0	CTS	0x1	Clear To Send Signal Generates an interrupt when this signal is de-asserted during transmit.
1	CD	0x1	Carrier Detect Signal Generates and interrupt when this signal is de-asserted during receive.
2	Reserved	0x0	Reserved.

Table 82: CHR10 - Event Status (ES) (Continued)
Offset: MPSC0 0x8030, MPSC1 0x9030

Bits	Field	InitVal	Function
3	TIDLE	0x0	Tx in IDLE state Generates an interrupt upon entering IDLE state.
4	Reserved	0x0	Reserved.
5	RHS	0x0	Rx in HUNT state.
10:6	Reserved	0x0	Reserved.
11	RLIDL	0x0	1 = Rx IDLE Line
12	DPCS	0x0	1 = DPLL Carrier Sense.
13	RRF	0x0	1 = Rx Receiving Flags.
31:14	Reserved	0x0	Reserved.

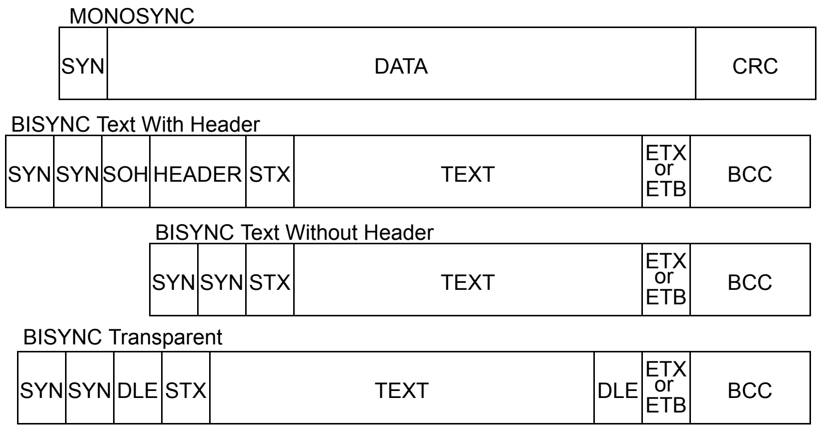
16.7 BISYNC Mode

The MV64360/1/2 BISYNC controller was designed to reduce CPU overhead by executing most of the protocol requirements for BISYNC/MonoSYNC mode without CPU interference.

When Auto Transparent mode is enabled, the MV64360/1/2 automatically switches to the transparent receive mode upon receiving a DLE STX sequence.

Other features are controlled by programming the bank of control registers.

Figure 55: Typical BISYNC/MonoSYNC Frames



16.7.1 BISYNC Transmit Operation

In BISYNC mode an MPSC handles the following protocol functions:

- Leading SYNC character transmission before a buffer with F bit set.

- Optional 32-bit transmission before the SYNC transmission.
- DLE transmission before a buffer with the TD bit set.
- BCC generation:
  - BCC (CRC-16, VRC/LRC and VRC/CRC-16) is calculated.
  - Buffers with BCE set to ‘0’ are excluded from BCC calculation.
  - CRC reset is controlled from the RC bit in Tx descriptor.
  - The calculation of BCC is sent or discarded according to the GC bit in the Tx descriptor.
- Automatic stuffing of DLE when transmitting a transparent buffer (buffer with TR bit set).
- SYNC transmission if underrun occurs.

BISYNC transmission is descriptor chain oriented. Transmission starts when the CPU issues a Transmit Demand command and continues until the channel’s SDMA reaches a NULL pointer or a ‘not owned’ descriptor.

16.7.2 BISYNC Receive Operation

There are two major operating modes in the BISYNC receiver.

Table 83: BISYNC Receiver Operating Modes

Mode	Function
Normal Mode	The CPU must monitor each received byte and manage each BISYNC operation (e.g., moving into transparent mode) manually.
Auto Transparent Mode	The MV64360/1/2 handles transparent mode automatically. This mode reduces the CPU burden since it can monitor the incoming data buffer-by-buffer and not byte-by-byte.

16.7.2.1 BISYNC Normal Receive Mode

In Normal Mode, the BISYNC receiver handles the following protocol functions:

- BISYNC, MonoSYNC, NibbleSYNC or External SYNC synchronization.
- Auto SYNC stripping in text mode.
- Auto DLE-SYNC stripping in transparent text mode.
- Auto SYNC stripping after receiving DLE ITB in transparent mode.
- Automatic exit of transparent mode after receiving DLE-ETX/ETB (if RTR bit in the MPCRx was cleared).
- Marking of buffers that contain transparent data by setting the TB bit in the descriptor.
- BCC generation:
  - BCC (CRC-16, VRC/LRC and VRC/CRC-16) is calculated.
  - In transparent text mode, CRC-16 always overrides the VRC.
    - SYNC (DLE-SYNC) is not included in the BCC calculation.
- Buffer closing at the reception of ETX, ETB, ITB and ENQ.
- Maintaining SYNC (stay in text mode) after ITB.
- Protocol correctness checking:
  - Test for ‘1’ padding at the end of block reception. (The CPU should ignore a padding error reported after ITB, and can use it when testing for proper NAK or EOT.)
  - Test for DLE-CTL after receiving DLE-ITB in transparent text mode. If another sequence arrives (except SYNCs), buffer is closed with DLE error.

16.7.2.2 BISYNC Auto Transparent Receive Mode

In Auto Transparent Mode, the BISYNC receiver handles the following protocol functions:

- BISYNC, MonoSYNC, NibbleSYNC, or External SYNC synchronization.
- Auto SYNC stripping in text mode.
- Auto DLE-SYNC stripping in transparent text mode.
- Auto SYNC stripping after receiving DLE ITB in transparent mode.
- Automatic switch to transparent mode after receiving DLE-STX.
- Automatic exit of transparent mode after receiving DLE-ETX/ETB.
- Marking of buffers that contain transparent data by setting the TB bit in the descriptor.
- BCC generation:
  - BCC (CRC-16, VRC/LRC and VRC/CRC-16) is calculated.
  - In transparent text mode, CRC-16 always overrides the VRC.
  - SYNC (DLE-SYNC) is not included in the BCC calculation.
    - Opening STX/SOH (DLE-STX) are discarded from BCC calculations.
- Buffer closing at the reception of ETX, ETB, ITB, and ENQ.
- Maintaining SYNC (stay in text mode) after ITB.
- Buffer closing after SYN-SYN-DLE-CHAR (when char is not STX).
- Protocol correctness checking:
  - Test for ‘1’ padding at the end of block reception. (The CPU should ignore a padding error reported after ITB, and can use it when testing for proper NAK or EOT.)
    - Test for DLE-CTL (CTL is a control character with B or H set) after receiving DLE-ITB in transparent text mode. If another sequence arrives (except SYNCs), buffer is closed with a DLE error.

The BISYNC receive process is block oriented. A block starts after a buffer was closed due to control character reception, overrun, protocol error, parity error, or line error (i.e. CD de-assertion).

The first descriptor in a block is marked with F bit set to ‘1’. The last descriptor in block is marked with L bit set to ‘1’. The last descriptor also includes the actual status report for the block. Intermediate descriptors can be recognized by having both F and L bit set to ‘0’.

16.7.3 SDMAx Command/Status Field for BISYNC Mode

When an MPSC is in BISYNC mode the Command/Status field in the corresponding SDMAx descriptor has the following format:

Table 84: SDMAx Command/Status Field for BISYNC Mode

Bits	Rx - Function	Tx - Function
0	CE - CRC/LRC Error	Reserved
1	CDL - CD Loss	CTSL - CTS Loss
2	DE - Decoding Error	Reserved
3	DLE - DLE Error. While in transparent mode, this indicates a DLE was received and the following byte was not a valid control character.	Reserved
4	PR - Parity Error. Last byte in buffer has parity error.	Reserved

Table 84: SDMAx Command/Status Field for BISYNC Mode (Continued)

Bits	Rx - Function	Tx - Function
5	Reserved	Reserved
6	OR - Data Overrun	Reserved
7:8	Reserved	Reserved
9	PDR - Pad Report. This is set if there were no four consecutive ‘1’s after the block reception.	Reserved
10	Reserved	Reserved
11	TB - Transparent Buffer. Buffer contains transparent data.	Reserved
12	Reserved	Reserved
13	C - Last bytes in buffer is a user defined control character.	Reserved
14	B - Last bytes in buffer are BCC.	Reserved
15	ES - Error Summary ES = CDL || DE || DLE || PR || OR	ES - Error Summary ES = CTSL
16	L - Last	L - Last **NOTE:** Transmit Bit 22 is used only if L bit is set to ‘1’. If L bit is set to ‘0’, no BCC is sent at the end of this buffer transmission.
17	F - First	F - First
18	Reserved	TR - Transparent mode. • 0 - Normal mode. SYNC will be sent in case of underrun • 1 - Transparent Mode. DLE-SYNC will be sent in case of underrun. CRC-16 will be used.
19	Reserved	TD - Transmit DLE before transmitting the buffer. This bit is valid only for transparent buffers. The preceding DLE is not included in the BCC calculations.
20	Reserved	BCE - BCC Enable • 0 - Buffer must be excluded from BCC calculations • 1 - Buffer must be included in BCC calculation
21	Reserved	RC - Reset BCC • 0 - BCC/LRC is accumulated. • 1 - BCC/LRC is reset.

Table 84: SDMAx Command/Status Field for BISYNC Mode (Continued)

Bits	Rx - Function	Tx - Function
22	Reserved	GC - Generate BCC/LRC.
23	EI - Enable Interrupt	EI - Enable Interrupt
29:24	Reserved	Reserved
30	AM - Auto Mode	AM - Auto Mode
31	O - Owner	O - Owner

16.7.4 MPSCx Protocol Configuration Register (MPCRx) for BISYNC

Figure 56: MPSCx Protocol Configuration Register (MPCRx) for BISYNC

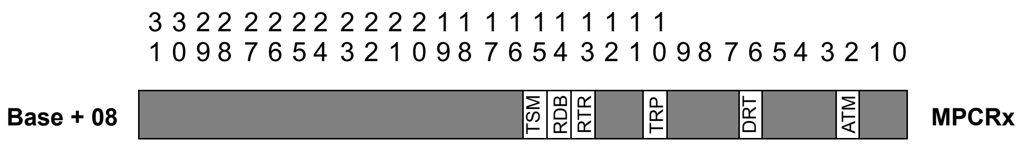


Table 85: MPSCx Protocol Configuration (MPCRx) for BISYNC

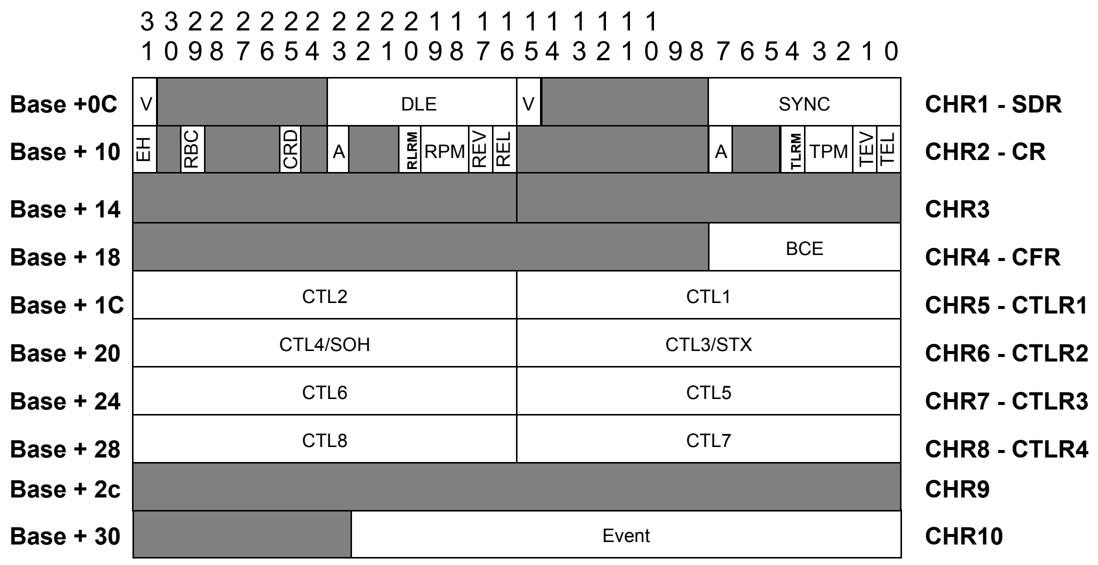
Bits	Field	InitVal	Function
1:0	Reserved	0x0	Reserved.
2	ATM	0x0	Auto Transparent Mode 0 = Normal Mode. 1 = Receiver switches to transparent mode after receiving DLE-STX and exits transparent mode upon receiving a DLE-ETB or DLE-ETX sequence. When switching to transparent mode, new buffers are opened for transparent data. When the ATR bit is set to '1' the following characters should be programed into CTL1-8: • CTL3 - STX • CTL4 - SOH **NOTE:** When entering transparent mode either automatically or by issuing an RTR command, the Receiver will strip automatically leading DLEs. The TB bit in the descriptor will be set to signal the software that the buffer contains transparent data.
5:3	Reserved	0x0	Reserved.
6	DRT	0x0	Disable Rx on Tx When DRT is set to '1' the Rx path is closed during Tx. This is useful in a multidrop configuration when a user doesn't want to receive its own frames.
9:7	Reserved	0x0	Reserved.

Table 85: MPSCx Protocol Configuration (MPCRx) for BISYNC (Continued)

Bits	Field	InitVal	Function
10	TRP	0x0	Trailing Pad When set, the BISYNC transmitter sends a PAD character (0xFF) at the end of each outgoing frame (i.e. after a buffer with L bit set.)
12:11	Reserved	0x10	Reserved.
13	RTR	0x0	Receive Transparent Mode 0 = The receiver is placed in normal mode with sync stripping and control character recognition operative. 1 = The receiver is placed in transparent mode. Syncs DLEs and control characters are recognized only after leading DLE characters. CRC16 is calculated even in VRC/LRC mode while in transparent mode. **NOTE:** When entering transparent mode either automatically or by issuing an RTR command, the receiver automatically strips leading DLEs. The TB bit in the descriptor is set to signal the software that the buffer contains transparent data.
14	RDB	0x0	Receive Discard From BCC When this bit is set, the received byte is not included in the BCC. The software must set this bit within the byte time window that starts when the character is in the Rx machine internal buffer. (The software can use the BISYNC interrupts for proper synchronization.) This bit is used in software to control BISYNC. The MV64360/1/2 clears the RDBCC bit after discarding the required byte from BCC.
15	TSM	0x0	Tx SYNC Mode 0 = Two SYNC characters are transmitted. 1 = 32 SYNC characters are transmitted. **NOTE:** The Tx machine sends at least two bytes even in MonoSYNC or NibbleSYNC modes.
31:16	Reserved	0x0	Reserved.

16.7.5 Channel Registers (CHxRx) for BISYNC Mode

Figure 57: Channel Registers (CHxRx) for BISYNC



Unless otherwise is specified:

- ‘1’ means set
- ‘0’ means unset.
- ‘0’ is the default value after reset.

16.7.5.1 CHR1 - SYNC/DLE Register (SDR)

CHR1 bits [7:0] holds the SYNC character and CHR1 [23:16] holds the DLE character for the channel. After reset it holds the value of 7E in the SYNC field and FE in the DLE field. The user must write the appropriate values before enabling the Rx/Tx machines.

If bit 15 is set, the BISYNC receive machine discards the SYNC patterns received in a middle of a message.

Note

This usually happens when the transmitter experiences underrun.

If bit [15] is ‘0’ the SYNC characters is transferred to the receive buffer.

If bit 31 is ‘1’, the first DLE received in transparent mode is discarded. If bit 31 is ‘0’, the BISYNC receiver is not discard DLE in transparent mode.

A BISYNC transmitter always stuffs the leading DLE before transmitting the DLE that is part of a transparent buffer (transmit descriptor with TR bit set). In order to send DLE ETX, for example, the CPU must either prepare a buffer that contains DLE ETX and set TR=’0’, or prepare a buffer with ETX and program the transmitter to send a leading DLE by setting the TD bit in the descriptor.

A BISYNC transmitter always transmits SYNC-SYNC at the beginning of a frame. This is true in MonoSYNC and NibbleSYNC modes.

When a BISYNC transmitter experiences underrun it transmits continuous SYNC patterns in text mode or DLE-SYNC in transparent mode. The BISYNC transmitter exits this state upon receiving new data or when the CPU issues a Stop or Abort command.

The receiver SYNC length is programmable. The actual length is determined according to the value of the RSYL bits in the MMCRx. If the RSYL bits equal #00b, the synchronization is done externally and the receiver will start receiving when CD# is asserted.

In NibbleSync mode, bits [7:4] are used by the receiver for sync recognition. Bits [3:0] should return the SYNC pattern in order to assure proper SYNC transmission.

16.7.5.2 CHR2 - Command Register (CR)

Table 86: CHR2 - Command (CR)

Bits	Field	InitVal	Function
0	TEL	0x0	Tx Enable Longitudinal Redundancy Check 0 = LRC is disabled. 1 = LRC is enabled. (TEL default value is 0 and the CPU must write “1” to it in order to enable LRC). When set, TEL **overrides** the CRC mode that was programed in the CRCM field in the MMCRx.
1	TEV	0x0	Tx Enable Vertical Redundancy Check (Parity Bit) 0 = VRC is disabled. 1 = VRC is enabled. (TEV default value is ‘0’ and the CPU must write ‘1’ to it in order to enable VRC).
3:2	TPM	0x0	Transmit Parity Mode 00 = Odd 01 = Low (always “0”) 10 = Even 11 = High (always “1”)
4	TLRM	0x0	Transmit Longitudinal Redundancy Mode 0 = Odd 1 = Even
6:5	Reserved	0x0	Reserved.
7	A	0x0	Abort Transmission Abort transmission immediately and go to IDLE. The descriptor is not closed or incremented. **NOTE:** Command is not synchronized to byte.
15:8	Reserved	0x0	Reserved.
16	REL	0x0	Rx Enable Longitudinal Redundancy Check. 0 = LRC is disabled. 1 = LRC is enabled. This is the normal mode for BISYNC. When set, REL **overrides** the CRC mode that was programed in the CRCM field in the MMCRx.

Table 86: CHR2 - Command (CR) (Continued)

Bits	Field	InitVal	Function
17	REV	0x0	Rx Enable Vertical Redundancy Check (parity bit). 0 = VRC (parity) is disabled. 1 = VRC is enabled. This is the normal mode for BISYNC.
19:18	RPM	0x0	Receive Parity Mode 00 = Odd 01 = Low (always '0') 10 = Even 11 = High (always '1')
20	RLRM	0x0	Receive Longitudinal Redundancy Mode 0 = Odd 1 = Even
22:21	Reserved	0x0	Reserved.
23	A	0x0	Abort Reception Abort receive immediately and go to IDLE. The descriptor is not closed or incremented. The processor must issue an enter hunt command after an abort command to enable reception. The A bit is cleared upon entering IDLE state.
24	Reserved	0x0	Reserved.
25	CRD	0x0	Close Rx Descriptor When the CPU issues a CRD command the current receive descriptor is closed and the following received data is SDMA'd into a new buffer. If there is no active receive in process, no action takes place.
28:26	Reserved	0x0	Reserved.
29	RBC	0x0	Reset BCC The CPU issues an RBC command to manually reset the CRC-LRC/VRC generator. The BCC calculation starts with the next byte. The MV64360/1/2 clears the RBC bit after resetting BCC.
30	Reserved	0x0	Reserved.
31	EH	0x0	Enter Hunt Upon receiving an enter hunt command, the receive machine moves to a hunt state and continuously searches for an opening SYNC or external SYNC. If an enter hunt mode command is issued during frame reception, the current descriptor is closed with a CRC error. The EH bit is cleared upon entering a hunt state.

The Main Configuration register's `ET` bit [6] (see Table 575 on page 646) must be set to '1' before issuing any of the following commands:

- Transmit Demand.
- Stop Transmission.
- Abort Transmission.

The Main Configuration register's `ER` bit [7] must be set to '1' before issuing any of the following commands:

- Enter Hunt

- Reset BCC
- Close Rx Descriptor
- Abort Reception.

When the ET or ER bits are de-asserted, the MPSCx transmit/receive channel is in low power mode (NO CLOCK).

Note

Issuing one of the above commands in this state will lead to unpredictable results.

Setting `TEL` to ’0’, `TEV` to ’1’ and `CRCM` to ’001’, or setting `REL` to ’0’, `REV` to ’1’ and `CRCM` to ’001’, will set the BISYNC transmitter/receiver to work in VRC+CRC16 mode. The calculated parity bit is considered part of the data that the CRC-16 checks.

When a BISYNC transmitter transmits a transparent buffer, it automatically switches to the CRC that was programmed in the CRCM field in MMCRx. When a receiver enters transparent mode, it automatically switches to the CRC that was programed in CRCM field in MMCRx. In both cases, CRCM must be programed to ‘001’ in order to meet the BISYNC CRC-16 specifications.

16.7.5.3 CHR4 - Control Filtering Register (CFR)

Bits 7:0 of the CFR register are the Bit Comparison Enable bits. Setting ‘1’ in one of the BCE bits enables the control comparison for this bit

16.7.5.4 CHR5-8 - BISYNC Control Character Registers

Figure 58 shows a BISYNC control register format.

The CHAR field holds the pattern for the control character while bits 8-15 are used to control the MV64360/1/2 behavior when the control character is recognized.

Figure 58: BISYNC Control Character Register Format

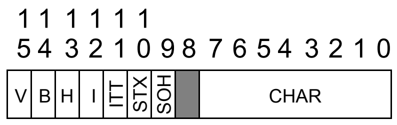


Table 87: BISYNC Control Character Format

Bits	Field	InitVal	Function
7:0	CHAR	0x0	The Control Character To Sync On **NOTE:** Bit 7 must be programmed according to the parity method in use. See Table 86 on page 270.
10:8	Reserved	0x0	Reserved.
11	ITT	0x0	Ignore While Receiving in Text Mode 0 = Normal control character. 1 = Ignore this character after entering text mode (i.e. after receiving SYN-SYN-STX/SOH).

Table 87: BISYNC Control Character Format (Continued)

Bits	Field	InitVal	Function
12	I	0x0	Interrupt 0 = No interrupt. 1 = Generate interrupt upon receiving this CHAR.
13	H	0x0	Hunt 0 = Close buffer and maintain SYNC. 1 = Close buffer and move to HUNT state.
14	B	0x0	BCC Next 0 = Close buffer. 1 = BCC is next. Receive BCC and than close buffer.
15	V	0x0	Valid. 0 = Entry is not valid. 1 = Entry is valid.

The BISYNC Control Character programming recommendations for Auto Transparent Mode and CPU Controlled Operation are shown in the following tables.

Table 88: Auto Transparent Programming

Control Character	V	B	H	I	ITT	STX	SOH
STX **NOTE:** CTL3 must be use to hold STX.	1	0	0	X	1	1	0
SOH **NOTE:** CTL4 must be use to hold SOH.	1	0	0	X	1	0	1
ETX	1	1	1	X	0	0	0
ITB	1	1	0	X	0	0	0
ETB	1	1	1	X	0	0	0
ENQ	1	0	1	X	0	0	0
EOT	1	0	1	X	1	0	0
NACK	1	0	1	X	1	0	0

Table 89: CPU Controlled Operation

Control Character	V	B	H	I	ITT	STX	SOH
ETX	1	1	1	X	0	0	0
ITB	1	1	0	X	0	0	0
ETB	1	1	1	X	0	0	0
ENQ	1	0	1	X	0	0	0
EOT	1	0	1	X	1	0	0

Table 89: CPU Controlled Operation

Control Character	V	B	H	I	ITT	STX	SOH
NACK	1	0	1	X	1	0	0

16.7.5.5 CHR9 - Reserved

This register is reserved.

Do not access this register in the BISYNC mode.

16.7.5.6 CHR10 - BISYNC Event Status Register (ES)

The ESR register holds information on the transmit/receive channel condition.

CHR10 can be read by the CPU for channel condition resolution. Some changes in the channel condition can generate maskable interrupts, as shown below.

Table 90: CHR10 - BISYNC Event Status (ES)

Bits	Field	Event
0	CTS	Clear To Send Signal **NOTE:** Interrupt is generated when signal is de-asserted during transmit.
1	CD	Carrier Detect Signal **NOTE:** Interrupt is generated when signal is de-asserted during receive.
2		Reserved
3	TIDLE	Tx in Idle State **NOTE:** Interrupt is generated upon entering IDLE state
5	RHS	Rx in HUNT state
6-10		Reserved
11	RLIDL	1 = Rx IDLE Line **NOTE:** Interrupt is generated upon change in line status
12	DPCS	1 = DPLL Carrier Sense
13-15		Reserved.
16-23	RCRn	Received Control Character n When the BISYNC receiver recognizes a control character it sets the corresponding RCRn bit. Bit 16 (RCR1) corresponds to CTL1. Bit 23 (RCR8) corresponds to CTL8. RCRn bits are cleared by writing ‘1’ to the bit. RCRn is set if the corresponding control character arrives, and both its Valid bit and Interrupt bit are also set (e.g., bit 16 will be set if CTL1 arrives, and both CTL1’s “V” bit is set, and CTL1’s “I” bit is also set.)

Note

PERR is set in transparent mode during SYN stripping when a non-DLE or SYN character is received. This is a protocol violation. The receiver moves to the hunt mode and a maskable interrupt is generated. The received character is discarded.

16.8 UART Mode

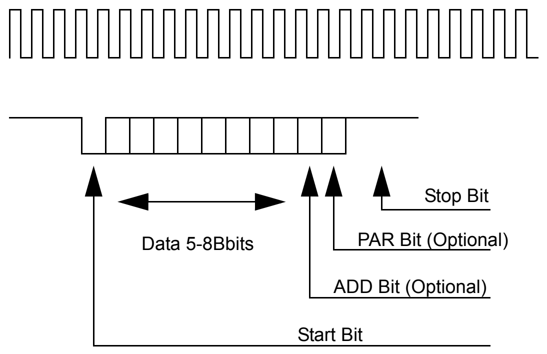
16.8.1 UART Receive/Transmit Operation

In UART mode an MPSC performs the following protocol functions:

- Start/Stop bit framing.
- Programmable data lengths (5-8 bits).
- Synchronous and asynchronous support.
- Message oriented data support.
- Parity detection and generation.
- Frame error, noise error, break, and idle detection.
- Support for HDLC over asynchronous control-octet transparency protocol.
- Multidrop operation with address recognition of up to two different addresses.

Figure 59 shows a typical UART frame format. A frame with a start bit is followed by 5-8 data bits. The address and parity bits are optional.

Figure 59: Typical UART Frame



At the end of a frame there are 1–2 stop bits before the transmitter can start to transmit the next frame. If there is nothing to transmit, a continuous ‘1’ is transmitted. This indicates that the line is idle.

The MV64360/1/2’s UART samples each bit three times near its central point to define the bit value. A new start bit can be recognized only after the last stop bit sample is received. For example, at a 16x clock rate, the receiver can receive a start bit after a 9/16 bit time long stop bit.

When in UART mode, the RDW bit in the MMCRx should be set to configure the MPSCx data path to 8 bits.

A UART transceiver can work in asynchronous or is synchronous modes.

16.8.1.1 Asynchronous Mode

In Asynchronous mode, the DPLL sampling rate is set to 8x, 16x, or 32x of the data rate. The DPLL is synchronized by the falling edge of the start bit. If no error occurs, it maintains synchronization until the last bit in a frame is received.

Each bit is sampled three times around it’s middle point. The bit value is determined by a majority vote. This feature helps to filter out noise from received data.

16.8.1.2 Isochronous Mode

In Isochronous mode, the DPLL sampling rate will be 1x the data rate. The receive data must be synchronized to the receive clock.

16.8.2 SDMAx Command/Status Field for UART Mode

When an MPSC is in UART mode the Command/Status field in the corresponding SDMAx descriptor has the following format:

Table 91: SDMAx Command/Status Field for UART Mode

Bit	Rx - Function	Tx - Function
0	PE - Parity Error. Last byte in buffer has parity error.	Reserved
1	CDL - CD Loss	CTSL - CTS Loss
2	Reserved	Reserved
3	FR - Framing Error	Reserved
5:4	Reserved	Reserved
6	OR - Data Overrun	Reserved
8:7	Reserved	Reserved
9	BR - Break Received while receiving data into this buffer	Reserved
10	MI - Max Idle. Buffer was closed due to Max_Idle timer expiration. **NOTE:** When this bit is set, the status of bit 0 is disregarded.	Reserved
11	A - Address. First byte in the buffer is an address. (Valid only in multidrop mode, ‘00’ in point to point mode.)	Reserved
12	AM - Address match. This bit will be set to ‘1’ when a match occurred even if the V bit of the address is disabled.	Reserved
13	CT - The last byte in the buffer was precede by a transparency control octet.	Reserved
14	C - The last byte in a buffer is a user define control character.	Reserved
15	ES - Error Summary ES = PE || CDL || FR || OR	ES - Error Summary ES = CTSL
16	L - Last	L- Last
17	F - First	F - First

Table 91: SDMAx Command/Status Field for UART Mode (Continued)

Bit	Rx - Function	Tx - Function
18	Reserved	P - Preamble. When set, the UART will send an IDLE preamble before buffer data. If data length is 0, only preamble IDLE will be send.
19	Reserved	A - Address. When set, buffer content will be sent with address bit on. Valid only in multidrop mode.
20	Reserved	NS - No Stop Bit. When set, data will be sent without stop bit.
22:21	Reserved	Reserved
23	EI - Enable Interrupt	EI - Enable Interrupt
29:24	Reserved	Reserved
30	AM - Auto Mode	AM - Auto Mode
31	O - Owner	O - Owner

16.8.3 MPSCx Protocol Configuration Register (MPCRx) for UART Mode

Figure 60: MPSCx Protocol Configuration Register (MPCRx) for UART Mode

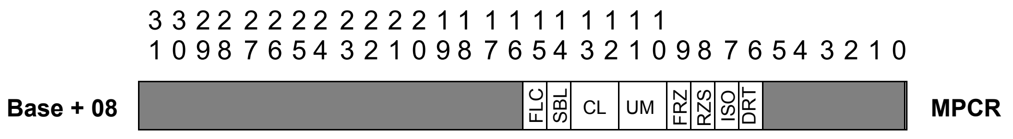


Table 92: MPSCx Protocol Configuration (MPCRx) for UART Mode

Bits	Field	InitVal	Function
5:0	Reserved	0x0	Reserved.
6	DRT	0x0	Disable Rx on Tx. When DRT is set to '1' the Rx path is closed during Tx. This is useful in multidrop configurations when a user doesn't want to receive its own frames
7	ISO	0x0	Isochronous Mode 0 = Asynchronous Mode Start and stop bits are expected. RENC in the MMCRx should be programmed to NRZ and RCDV should be programmed to x8, x16 or x32 mode. (x16 is recommended for most applications). 1 = Isochronous Mode The receive bit stream is assumed to be synchronous to the receive clock. RCDV should be programmed to x1 mode.

Table 92: MPSCx Protocol Configuration (MPCRx) for UART Mode (Continued)

Bits	Field	InitVal	Function
8	RZS	0x0	Receive Zero Stop Bit 0 = Normal Mode At least one stop bit is expected. 1 = Zero Stop Bit The receiver continues reception when a stop bit is missing. If a ‘0’ is received when stop bit is expected, this bit is considered a start bit. The FE (Framing Error) bit is set and the next bit to be received is considered to be data.
9	FRZ	0x0	Freeze Tx 0 = Restart Tx after freeze (normal operation). Transmission continues from the place it stopped. 1 = Freeze Tx at the end of the current character.
11:10	UM	0x0	UART Mode 00 = Normal Mode Multidrop is disabled and IDLE line wake up is selected. A UART receiver wakes up after entering hunt mode upon receiving an IDLE character (all one character). 01 = Multi Drop Mode In multidrop mode, there is an additional Address/Data bit in each character. Upon receiving an address character, the UART receiver compares it to two 8-bit addresses stored in it’s channel registers. If a match occurs, the receiver transfers the address and the following characters into a new buffer. If there is a no match, the character is discarded and the receiver is set to the hunt mode. If none of the addresses is valid (V bit in both address register is set to ‘0’), there is always a match and all the characters are transferred into the DRAM. Addresses are always be placed in a new buffer (Regardless of the V bit). The receiver receives characters until a new address is received, an abort character is received, an enter hunt command is issued, or until max idle counter expiration. Upon max idle counter expiration, the receiver is set to the hunt mode. 10 = Reserved. 11 = Reserved.
13:12	CL	0x01	Character Length 00 = 5 data bits 01 = 6 data bits 10 = 7 data bits 11 = 8 data bits
14	SBL	0x0	Stop Bit Length 0 = One stop bit 1 = Two stop bits

Table 92: MPSCx Protocol Configuration (MPCRx) for UART Mode (Continued)

Bits	Field	InitVal	Function
15	FLC	0x0	Flow Control 0 = Normal Mode The CTSM bit in the MMCRx determines the CTS# pin behavior. 1 = Asynchronous Mode When CTS# is negative, transmission stops at the end of the current character. When CTS# is asserted again, the transmission starts from the place it stopped. No CTS# lost is reported. Line is IDLE (MARK) during CTS# de-assertion period.
31:16	Reserved	0x0	Reserved.

Note

When CD# is de-asserted during frame reception UART behavior is different for multidrop and normal modes. In normal mode the UART hunts for an IDLE character (hunting starts when CD# is asserted again) before receiving valid start bit. In this mode, transmitting from a MV64360/1/2 model to another should be with the 'P' bit in the buffer descriptor set. In multidrop mode, the UART receiver hunts for a start bit as soon as CD# is asserted again.

16.8.4 UART Stop Bit Reception and Framing Error

The UART receiver always expects to find a stop bit at the end of a character. If no stop bit is detected, the Framing Error (FE) bit is set in the receive descriptor. After a framing error, the reception process is controlled by the RZS and UM bits in the UART MPCRx. The various options are summarized in the table bellow.

Table 93: UART Stop Bit Reception and Framing Error

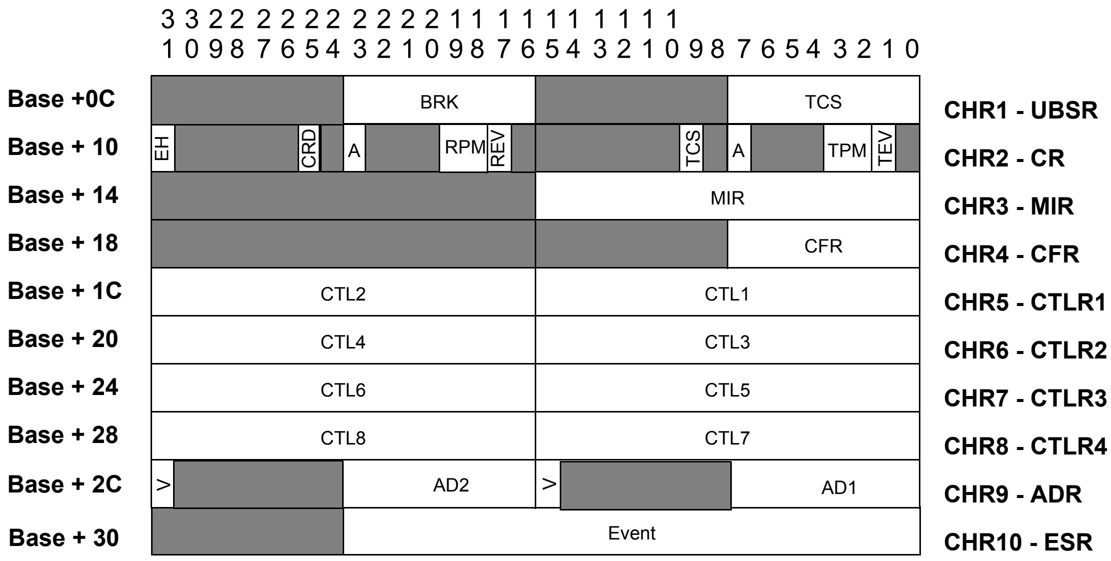
UM	RZS	Operation	Break Recognition
00	0	Go to hunt after missing a stop bit. The receiver is enabled after receiving a new IDLE char.	Single Break
00	1	The receiver tries to synchronize itself. The missing stop bit is considered as the following start bit and the reception process continues.	Two Break Sequence
01	0	Goes to hunt after missing stop bit. The receiver is enabled after receiving new address character.	Single Break
01	1	The receiver tries to synchronize itself. The missing stop bit is considered as the following start bit and the reception process continues.	Two Break Sequence

16.8.5 Channel Registers (CHxRx) for UART Mode

The MPSCx Channel Registers (CHxRx) are protocol dependent.

Figure 61 shows the CHxRx format in UART mode.

Figure 61: Channel Registers (CHxRx) for UART Mode



Unless otherwise is specified:

- ‘1’ means set.
- ‘0’ means unset.
- ‘0’ is the default value after reset.

16.8.5.1 CHR1 - UART Break/Stuff Register (UBSR)

The UART Break/Stuff register has two fields: Break Count (BRK)(CHR1[23:16]) and Control Stuff Character (TCS) (CHR1[7:0]).

With the BRK field, the UART transmitter will starts to transmit break characters after receiving an abort command. The number for the break character to send is programmed into the BRK field.

For example, when BRK equals ‘0’, no break character is transmitted. When BRK equals ‘1’, one break character is transmitted.

A break character is a character with all ‘0’s including it’s stop bit.

Upon issuing a TCS command, the transmitter sends a TCS character after the current transmitting character. This allows a transmitter to bypass the normal pipeline when a special control character must be send (e.g. XON/ XOFF).

Upon receiving a break character, the UART stops the reception process and moves to the hunt state. In a point to point configuration, the receiver is hunting for a new IDLE character. In a multidrop configuration, the receiver hunts for a new address character.

When the UART is in RZS=0 mode after receiving a break sequence, the descriptor is closed with BR bit (bit 9) set. In addition, a “break descriptor” also has the FE bit (bit 3) set and, if in odd parity, the PE bit (bit 0) is also set.

When the UART in RZS=1 mode, two consecutive break sequences are needed for proper break recognition. The first break character is not recognized. Instead, the UART receiver closes the descriptor with the FE bit (bit 3) set and, if in odd parity, the PE bit (bit 0) will also be set. The second break will be recognized as a break and a descriptor will be closed with the BR bit (bit 9) set. In addition, a “break descriptor” will also have the FE bit (bit 3) set and, if in odd parity, the PE bit (bit 0) will also be set.

16.8.5.2 CHR2 - Command (CR)

Table 94: CHR2 - Command (CR)

Bits	Field	InitVal	Function
0	Reserved	0x0	Reserved.
1	TEV	0x0	Tx Enable Vertical Redundancy Check 0 = VRC (parity) is disabled. 1 = VRC is enabled.
3:2	TPM	0x0	Transmit Parity Mode 00 = Odd 01 = Low (always 0) 10 = Even 11 = High (always 1)
6:4	Reserved	0x0	Reserved.
7	A	0x0	Transmit Abort Aborts the transmission immediately (on byte boundaries) and goes to IDLE. The descriptor is not closed or incremented. After receiving an abort command, the MV64360/1/2 halts the transmit process and starts sending a break sequence according to the BRK field in CHR1. **NOTE:** Command is not synchronized to byte.
8	Reserved	0x0	Reserved.
9	TCS	0x0	Transmit TCS Character. The TCS character is transmitted after the current transmitted character. The transmitter then continues with the normal Tx sequence. The TCS command can be used to send out of band characters such as XOFF and XON.
16:10	Reserved	0x0	Reserved.
17	REV	0x0	Rx Enable Vertical Redundancy Check 0 = VRC (parity) is disabled. 1 = VRC is enabled.
19:18	RPM	0x0	Receive Parity Mode. 00 = Odd 01 = Low (always ‘0’) 10 = Even 11 = High (always ‘1’)
22:20	Reserved	0x0	Reserved.
23	A	0x0	Receive Abort Abort receive immediately and go to IDLE. The descriptor is not closed or incremented. The processor must issue a enter hunt command after an abort command in order to enable reception. The A bit is cleared upon entering IDLE state.

Table 94: CHR2 - Command (CR) (Continued)

Bits	Field	InitVal	Function
24	Reserved	0x0	Reserved.
25	CRD	0x0	Close Rx Descriptor When the CPU issues a CRD command, the current receive descriptor is closed and subsequent received data is DMA'd into a new buffer. If there is no active receive process, no action takes place. The MV64360/1/2 clears the CRD bit upon closing the buffer status. **NOTE:** It usually takes a few cycles from the time the CRD bit is closed until the SDMAx actually closes the buffer. If programed to, the SDMAx generates a maskable interrupt when closing a buffer.
30:26	Reserved	0x0	Reserved.
31	EH	0x0	Enter Hunt Upon receiving an enter hunt command, the receive machine moves to a hunt state and continuously searches for an opening character. An opening character is considered an IDLE char in point to point mode (UM=00) or a matched address in multidrop mode. The EH bit is cleared upon entering a hunt state.

The Main Configuration register's ET bit [6] must be set to '1' before issuing any of the following commands:

- Transmit Demand
- Stop Transmission
- Transmit TCS Character
- Abort Transmission

The Main Configuration register's ER bit [7]must be set to '1' before issuing any of the following commands:

- Enter Hunt
- Close Rx Descriptor
- Abort Reception

When the ET or ER bits are de-asserted, the MPSCx transmit/receive channel is in low power mode (NO CLOCK).

Note

Issuing one of the above commands in this state leads to unpredictable results.

The CRCM in the MMCR must be set to 011 for LRC/VRC mode.

16.8.5.3 CHR3 - Max Idle Register (MIR)

This 16-bit value (CHR3[15:0]) defines the number of IDLE characters the receiver waits before it closes a descriptor and a maskable interrupt is generated.

When set to '0', the counter is disabled.

The counter is pre-loaded every time a non-IDLE character is received.

16.8.5.4 CHR4 - Control Filtering Register (CFR)

Bits 7:0 of the CFR register are the Bit Comparison Enable bits.

Setting a ‘1’ in one of the BCE bits enables the control comparison for this bit.

16.8.5.5 CHR5-8 - UART Control Character Registers

Figure 62 shows a UART control register format.

The CHAR field holds the pattern for the control character while bits 8-15 are used to control the MV64360/1/2’s behavior when the control character is recognized.

Figure 62: UART Control Character Register Format

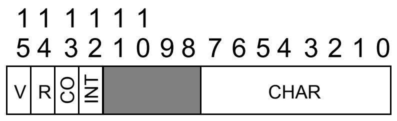


Table 95: UART Control Character Format

Bits	Field	InitVal	Function
7:0	CHAR	0x0	The control character to sync on.
11:8	Reserved	0x0	Reserved.
12	INT	0x0	Interrupt 0 = No interrupt. 1 = Generate interrupt upon receiving this CHAR.
13	CO	0x0	Control Octet (ISO 3309 Control Octet) Upon receiving a control octet, the control octet is discarded and the 6th bit (i.e. bit 5 in CHR5) of the following octet is complemented. The current buffer is closed with the CO bit asserted. **NOTE:** When the CO bit is set, the CHAR field must be programmed with ‘10111110’ in order to be ISO-3309 compatible.
14	R	0x0	Reject 0 = Receive character and close the buffer. 1 = Reject character. The character is discarded, the buffer is closed and a maskable interrupt is generated.
15	V	0x0	Valid. 0 = Entry is not valid 1 = Entry is valid

16.8.5.6 CHR9 - Address Register (ADR)

CHR9 holds the UART addresses for multidrop operation. The MV64360/1/2 UART supports up to 2 addresses.

Upon receiving an address, the UART transfers the previous frame status to the SDMA. This causes the SDMA to close the previous frame descriptor and to locate the address in a new buffer.

There are two modes for address recognition operation. The first mode, setting of ‘1’, allows the address and following characters to be transferred to the SDMA only if there is a match. The second mode, setting of ‘0’, allows all frames to be passed to the SDMA. The CPU can use the M bit in the last frame descriptor to check if a match occurred.

16.8.5.7 CHR10 - UART Event Status Register (ESR)

The ESR register holds information on the transmit/receive channel condition. CHR10 can be read by the CPU for channel condition resolution.

Some changes in the channel condition can generate maskable interrupts, as shown in Table 96.

Table 96: CHR10 - UART Event Status Register (ESR)

Bits	Field	Event
0	CTS	Clear To Send Signal **NOTE:** Interrupt is generated when signal is de-asserted during transmit.
1	CD	Carrier Detect Signal **NOTE:** Interrupt is generated when signal is de-asserted during receive.
2		Reserved.
3	TIDLE	Tx in Idle State **NOTE:** Interrupt is generated upon entering IDLE state.
4		Reserved.
5	RHS	Rx in HUNT State
6		Reserved.
7	RLS	Rx Line Status
10:8		Reserved.
11	RLIDL	1 = Rx IDLE Line **NOTE:** Interrupt is generated upon change in line status.
15:12		Reserved.
23:16	RCRn	Received Control Char n When the UART receiver recognizes a control character it sets the corresponding RCRn bit. (bit 16 (RCR1) corresponds to CTL1... bit 23 (RCR8) corresponds to CTL8). RCRn bits are cleared by write a 1 to the bit.

16.9 Transparent Mode

In transparent mode, the MV64360/1/2 does not perform any protocol dependent data processing.

However, it gives the processor hardware assistance for bit reception, using the MV64360/1/2's powerful SDMA engines, and some assistance in synchronization, interrupt generation, and frame construction. The CPU also uses the built-in CRC engine for CRC generation and checking. In any case, CRC bits are transferred into memory for CPU use.

In transparent mode, the channel is fully configured from the MMCRx and no mode is defined by the channel registers.

A transparent channel is synchronous.If it is not serviced on time, underrun and overrun errors can occur.

The receiver can use external sync using the CD# input or synchronize itself on a SYNC sequence according to the RSYL bits in the MMCRx.

16.9.1 SDMAx Command/Status Field for Transparent Mode

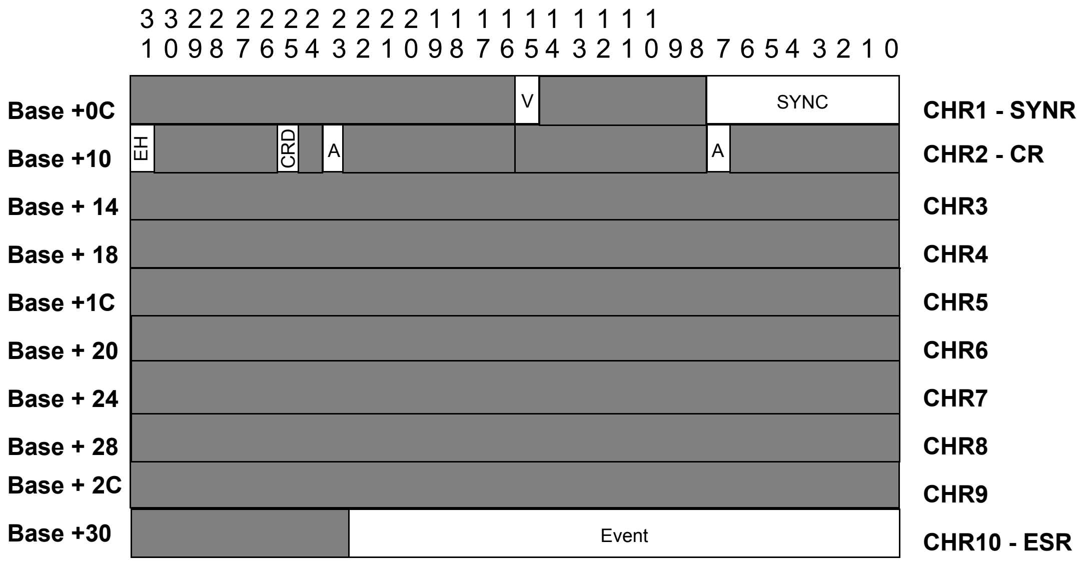
When an MPSC is in Transparent mode the Command/Status field in the corresponding SDMAx descriptor has the following format:

Table 97: SDMAx Command/Status Field for Transparent Mode

Bit	Rx - Function	Tx - Function
0	CE - CRC/LRC Error	Reserved.
1	CDL - CD Loss	CTSL - CTS Loss
2	DE - Decoding Error	Reserved.
3	Reserved.	Reserved.
4	Reserved.	Reserved.
5	Reserved.	Reserved.
6	OR - Data Overrun	UR - Data Underrun
14:7	Reserved.	Reserved.
15	ES - Error Summary ES = CE || CDL || DE || OR	ES - Error Summary ES = CTSL || UR
16	L - Last	L - Last
17	F - First	F - First
21:18	Reserved.	Reserved.
22	Reserved.	GC - Generate BCC/LRC.
23	EI - Enable Interrupt	EI - Enable Interrupt
29:24	Reserved.	Reserved.
30	AM - Auto Mode	AM - Auto Mode
31	O - Owner	O - Owner

16.9.2 Channel Registers (CHxRx) for Transparent Mode

Figure 63: Channel Registers (CHxRx) for Transparent Mode



Unless otherwise is specified:

- ‘1’ means set.
- ‘0’ means unset.
- ‘0’ is the default value after reset.

16.9.2.1 CHR1 - SYNC Register (SYNR)

The SYNC Register holds the synchronization for the channel receiver. After reset it holds the value of 7E in the SYNC field. The user should right the appropriate values before enabling the Rx/Tx machines.

There are two basic synchronization options for a transparent channel: Transparent Mode Synchronization and Transmitter Synchronization. The Transparent Mode Synchronization has two synchronization options, selected by setting RSYL[24:23] in the MMCRx.

The Transparent Mode Synchronization has two synchronization options. They are also selected by setting the RSYL [24:23] bits in the MMCRx.

Note

For more information about setting `RSYL`[24:23], see Table 576 on page 650.

Table 98: Transparent Mode Synchronization Options

Synchronization Option	Function
External Synchronization	(RSYL = '00') The receiver starts to receive data whenever CD# is asserted and stops receiving data when CD# is de-asserted (if CDM=0) or when the CPU issues an Enter Hunt Command.
Sync Hunt	RSYL = '01', '10', or '11' (nibble, byte or two bytes sync) The receiver hunts for the sync pattern, as defined by RSYL. When the synch pattern is recognized, the receiver starts to receive data. The receive process stops when CD# is de-asserted and CDM=0 or when the CPU issues an enter hunt command. If bit 15 is set, there is no transfer of the SYNC characters to the receiver. The syncs are stripped until the first data character is received, and are not calculated in the packet CRC. If bit 15 is reset, sync characters that appear after the sync pattern is recognized are regarded as data. On the transmitter side, in sync hunt mode, two sync characters are always sent at the beginning of a frame. **NOTE:** When RSYL equals 01, the Sync pattern is defined by bits [7:4] of the Sync Register.

There are two mode of transmit synchronization in transparent mode. They are selected by setting TSYN[12} in the MMCRx, see Table 575 on page 646.

Table 99: Transmitter Mode Synchronization Options

Synchronization Option	Function
TSYN = 0	Synchronization is achieved whenever CTS# is asserted.
TSYN=1	Synchronization is achieved after receiver synchronization and CTS# is asserted. The transmitter always starts to transmit on the receive byte boundaries. In external synchronization, when CTS# is asserted, the transmitter starts to transmit 8 bits after CD# assertion. In sync hunt mode, when CTS# is asserted, the transmitter starts to transmit 8 bits after sync recognition. If CTS# is de-asserted after the receiver gains synchronization, the transmitter waits to the byte boundary before it starts to transmit.

16.9.2.2 CHR2 - Command Register (CR)

Table 100: CHR2 - Command (CR)

Bits	Field	InitVal	Function
6:0	Reserved	0x0	Reserved.
7	A	0x0	Abort Transmission Aborts the transmission immediately (on byte boundaries) and goes to IDLE. The descriptor is not closed or incremented. Command is not synchronized to byte.
22:8	Reserved	0x0	Reserved.
23	A	0x0	Abort Reception Abort receive immediately and go to IDLE. The descriptor is not closed or incremented. The processor must issue an enter hunt command after an abort command in order to enable reception. The A bit is cleared upon entering IDLE state.
24	Reserved	0x0	Reserved.
25	CRD	0x0	Close Rx Descriptor When the CPU issues a CRD command the current receive descriptor is closed and the following received data is SDMA'd into a new buffer. If there is no active receive in progress, no action takes place.
30:26	Reserved	0x0	Reserved.
31	EH	0x0	Enter Hunt Upon receiving an enter hunt command, the receive machine moves to a hunt state and continuously searches for an opening sync or an external sync. If the enter hunt command is received during a frame reception, the current descriptor is closed with a CRC error. The EH bit is cleared upon entering a hunt state.

The Main Configuration register's ET bit [6] must be set to '1' before issuing any of the following commands:

- Transmit Demand
- Stop Transmission
- Abort Transmission

The Main Configuration register's ER bit [7] must be set to '1' before issuing any of the following commands:

- Enter Hunt
- Close Rx Descriptor
- Abort Reception

When the ET or ER bits are de-asserted, the MPSCx transmit/receive channel is in low power mode (NO CLOCK).

Note

Issuing one of the above commands in this state leads to unpredictable results.

16.9.2.3 CHR10 - Transparent Event Status Register (ESR)

The ESR register holds information on the transmit/receive channel condition. CHR10 can be read by the CPU for channel condition resolution. Some changes in the channel condition can generate maskable interrupts, as shown Table 101.

Table 101: CHR10 - Transparent Event Status (ES)

Bits	Field Name	Event
0	CTS	Clear To Send Signal **NOTE:** Interrupt is generated when signal is de-asserted during transmit.
1	CD	Carrier Detect Signal **NOTE:** Interrupt is generated when signal is de-asserted during receive.
2		Reserved.
3	TIDLE	Tx in Idle State **NOTE:** Interrupt is generated upon entering IDLE state.
4		Reserved.
5	RHS	Rx in HUNT state
11:6		Reserved.
12	DPCS	1 = DPLL Carrier Sense
15:13		Reserved.
23:16	RCRn	Received Control Character n When the transparent receiver recognizes a control character it sets the corresponded RCRn bit (bit 16 (RCR1) corresponds to CTL1... bit 23 (RCR8) correspond to CTL8). RCRn bits are cleared by writing '1' to the bit.
31:24		Reserved

16.10MPSC Cause and Mask Registers

Each MPSC has a dedicated interrupt cause register and an interrupt mask register (see H.6 "MPSC Cause and Mask Registers" on page 643). Unless states differently, the interrupt cause bits are set upon hardware events, and are cleared by software writing '0'. The Mask register's bits do not affect the actual setting of the interrupt cause bits. They only determine upon which cause event, should an interrupt pin be asserted.

16.11SDMA Operation

There are two SDMA channels dedicated for moving data between the MPSCs and memory buffers. Each SDMA channel consists of a DMA engine for receiving and one for transmitting.

Each SDMA channel has two dedicated FIFOs for data buffering. All FIFOs are 256 bytes deep.

For receive operations, the MPSC moves received data into the dedicated FIFO of the corresponding SDMA. Then, using descriptors set up by the user, the SDMA moves the data into memory buffers. For transmit operations, the SDMA uses descriptors set up by the user to move data out of buffers into the dedicated FIFO. The MPSC moves the data down to the serial communications link.

The SDMA channel descriptors use a chained data structure. They work without CPU interference after appropriate initialization. SDMA channels can be programed to generate interrupts on buffer or frame boundaries.

When enabled, the receive SDMAs run freely and expect to find a valid descriptor, when one is required. When a receive SDMA channel accesses an invalid descriptor, the receive SDMA process halts with a resource error status indication.

When enabled, the transmit SDMAs run freely until the end of the descriptor chain is reached. When a transmit SDMA accesses an invalid descriptor and the last descriptor was not marked as an end of frame descriptor, the transmit SDMA process halts with resource error status indication.

The SDMAs arbitrate for accessing the descriptors and buffers. A standard round-robin scheme is used for arbitration between them.

16.11.1Address Decoding

The two SDMAs share four address windows that can be individually configured. With each SDMA transaction, the address is first compared against the address decoding registers. Each window can be configured to a different target interface. Address comparison is done to select the correct target interface (DRAM, PCI, etc.).

Note

Unlike the GT-64240/60 devices, each of the MV64360/1/2 MPSCs SDMAs uses its own address decoding map, de-coupled from the CPU interface address decoding windows.

If the address does not match any of the address windows, an interrupt is generated and the port halts.

The PCI interface supports 64-bit addressing. Two out of the four address windows have an upper 32-bit address register. To access the PCI bus with 64-bit addressing cycles (DAC cycles), assign these windows to target the PCI bus. The address generated on the PCI bus is composed of the window base address and the High Remap register.

For an SDMA to not access an incorrect address space, an incorrect access due to a programing bug, each MPSC has it’s own access protection logic. This logic prevents the SDMA from read/write access to specific address windows. In case of an access violation, the port halts and an interrupt is asserted.

For full details on address the decoding scheme, see Section 7. “Address Space Decoding” on page 82.

16.11.2Big and Little Endian

The MV64360/1/2 supports both Little and Big Endian serial data.

The transmit and receive data endianess is determined by the SDMA Configuration register’s `BLMT` bit [7] and `BLMR` bits [6] (Table 577 on page 652). The SDMA engine organizes the read data in its buffer, based on this setting.

Additionally, the MV64360/1/2 supports access to Big and Little Endian devices on the PCI bus. PCI endianess is controlled through the PCI Command register’s `MSwap` and `MWswap` bits(Table 378 on page 535). Also, each address decoding window that is targeted to the PCI can be configured to a different endianess setting, in case the DMA channels target different endianess orientation PCI devices (see 13.10 “Data Endianess” on page 172).

16.12SDMA Descriptors

All SDMA data transfers are completed via a chained link of descriptors. The following rules must be followed when using the MV64360/1/2 SDMA descriptors:

- Descriptor length is 4LW and it must be 4LW aligned (i.e. Descriptor_Address[3:0]=0000).
- Descriptors may reside anywhere in CPU address space except NULL address, which is used to indicate end of descriptor chain.
- In normal mode (HDLC and Transparent) Rx buffers associated with Rx descriptors must be 64-bit aligned. Minimum size for Rx buffers is 8 bytes. In low latency, or byte, mode (BISYNC, UART, and Transparent) Rx buffers have no alignment restrictions.
- Tx buffers associated with Tx descriptors can start in any byte location.
- SDMA Rx and Tx buffers are limited to 64 KBs.

Figure 64: SDMA Descriptor Format

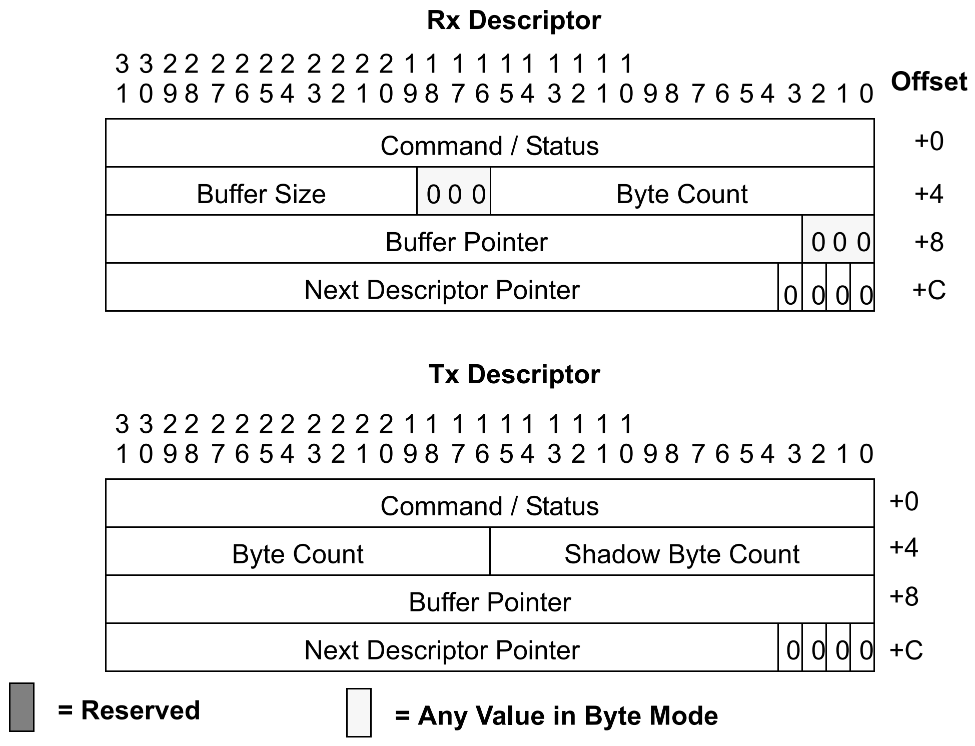


Table 102 through Table 106 provide detailed information about the descriptor fields.

Table 102: SDMA Descriptor - Command/Status Word

Bits	Field	Function
Contains commands bits that instruct the SDMA how to process a buffer and status bits that the SDMA updates upon closing a descriptor. The CPU uses the status bits to evaluate the buffer status. Except for bits 31, 30, 23, 17, and 16, the definition of the bits vary depending on which mode is being used. See: • 16.6.2 “SDMAx Command/Status Field for HDLC Mode” on page 256 • 16.7.3 “SDMAx Command/Status Field for BISYNC Mode” on page 265 • 16.8.2 “SDMAx Command/Status Field for UART Mode” on page 276 • 16.9.1 “SDMAx Command/Status Field for Transparent Mode” on page 285		
15:0		Determined by the mode selected.
16	L	Last Bit Indicates last buffer of a frame.
17	F	First Bit Indicates first buffer of a frame.
22:18		Determined by the mode selected.
23	EI	Enable Interrupt The MV64360/1/2 generates a maskable interrupt when closing descriptor with EI bit set. **NOTE:** If the RIFB bit is set in the SDMA configuration register, a Rx interrupt is generated only if this is the last descriptor associated with a received frame. In this case, EI bit setting is masked for intermediate descriptors.
29:24	Reserved	Determined by the mode selected.
30	AM	Auto Mode When set, the SDMA won’t clear the Owner bit of the descriptor at the end of buffer processing.
31	O	Owner Bit When set to’1’, the buffer can be processed by the MV64360/1/2. When set to ‘0’, the buffer can be processed by the CPU. An SDMA process will halt when a descriptor with owner bit set to ‘0’ is fetched.

Table 103: SDMA Descriptor - Buffer Size, Byte Count (Rx Descriptor)

Bits	Field	Function
15:0	Byte Count	The number of bytes that were actually written by the SDMA into the buffer. This number is never greater than Buffer Size. The CPU must initialize the Byte Count field with 0x0000.
31:16	Buffer Size	The buffer size field is valid only in receive descriptors and is reserved in transmit descriptors.The field is written by the CPU and read by the MV64360/1/2. When the buffer byte counter of a SDMA receive channel reaches the buffer size value, the SDMA will close the buffer descriptor and will move to the next buffer. Buffer Size must be a multiple of 8 when the MPSC is programmed to work in normal mode (HDLC and Transparent). Buffer Size can be arbitrary when working in low bandwidth mode (BISYNC, UART, and Transparent).

Table 104: SDMA Descriptor - Byte Count, Shadow Byte Count (Tx Descriptor)

Bits	Field	Function
15:0	Shadow Byte Count	The CPU must initialize this field with a value identical to the Byte Count field. The MV64360/1/2 subtracts the number of bytes actually transmitted from this parameter. Usually the MV64360/1/2 writes '0' in this field when closing a descriptor. However, when the transmit SDMA halts due to a transmit error, this number can be used to determine the number of bytes that were fetched into the MV64360/1/2. Setting both the Byte Count and Shadow Byte Count to '0' will cause the SDMA to close the descriptor and move to the next descriptor, if both or neither of the F and L bits are set. Setting Byte Count and Buffer Size to '0' in transmit descriptors with one of the F or L bits set will lead to unpredictable behavior.
31:16	Byte Count	Byte count is the number of bytes to be transmitted. Zero byte counters are not supported with retransmission. Do not use zero byte buffers with LAP-D protocol.

Table 105: SDMA Descriptor - Buffer Pointer

Bits	Field Name	Function
31:0	Buffer Pointer	32-bit pointer to the beginning of the buffer associated with the descriptor. The buffer can reside anywhere in memory or PCI address space.

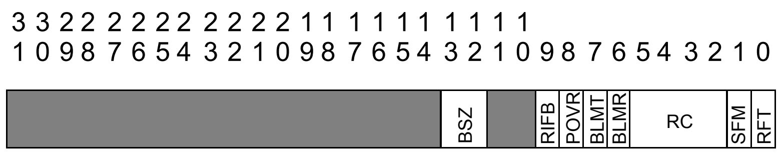
Table 106: SDMA Descriptor - Next Descriptor Pointer

Bits	Field Name	Function
31:4	Next Descriptor Pointer	32-bit Next Descriptor pointer to the beginning of the next descriptor in the chain. A descriptor can reside anywhere in memory or PCI space. Bits [3:0] must be set to'0'. DMA operation is stopped when a NULL value in the Next Descriptor Pointer is encountered.

16.13SDMA Configuration Register (SDC)

Each SDMA has a dedicated configuration register (SDCx), see Table 577 on page 652. The SDC must be initialized before enabling the SDMA channel.

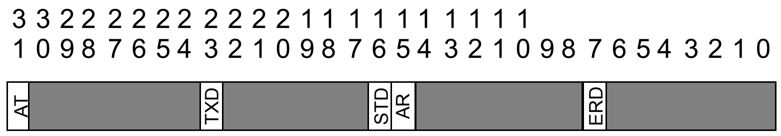
Figure 65: SDMAx Configuration Register (SDCx)



16.14SDMA Command Register (SDCMx)

Each SDMA has a dedicated SDMA Command Register (SDCMx) register to control its DMA process, see Table 578 on page 654.

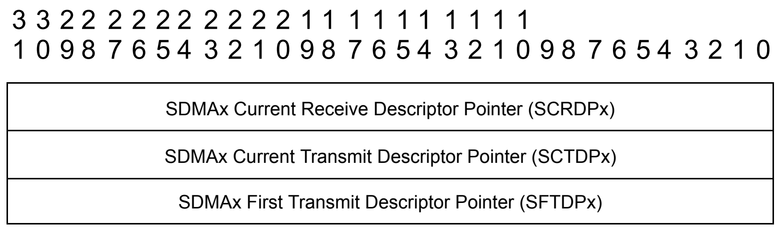
Figure 66: SDMA Command Register (SDCMx)



16.15SDMA Descriptor Pointer Registers

Each SDMA channel has three 32-bit registers that reside in a special descriptor's Dual Port memory located in the internal address space of the MV64360/1/2.

Figure 67: SDMA Descriptor Pointer Registers


3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
SDMAx Current Receive Descriptor Pointer (SCRDPx)
SDMAx Current Transmit Descriptor Pointer (SCTDPx)
SDMAx First Transmit Descriptor Pointer (SFTDPx)

16.15.1SDMA Current Receive Descriptor Pointer (SCRDP)

SCRDPx points to the current receive descriptor in memory. The CPU must write this register with the first descriptor address before enabling the SDMA receive channel. When a SDMA receive channel is enabled it will fetch the first descriptor pointed to by SCRDPx as part of its SDMA starting procedure.

16.15.2SDMA Current Transmit Descriptor Pointer (SCTDP)

SCTDPx points to the current transmit descriptor in memory. The CPU must write this register with the first descriptor address before enabling the SDMA transmit channel. When a SDMA transmit channel is enabled it will fetch the first descriptor pointed to by SCRDPx as part of its SDMA starting procedure.

16.15.3SDMA First Transmit Descriptor Pointer (SFTDP)

SFTDPx points to the first descriptor in a transmit frame. The CPU must write this register with the first descriptor address before enabling the SDMA transmit channel. The SDMA transmit controller uses the SFTDP when it needs to restart a transmission after collision (HDLC mode only). The MV64360/1/2 updates the content of SFTDP each time it fetches a descriptor with the F (first) bit set to '1'.

Note

The CPU must write the same value to both SCTDP and SFTDP before enabling the corresponding SDMA transmit channel.

16.16Transmit SDMA

16.16.1Transmit SDMA Definitions

- **SOF** (Start Of Frame descriptor): Descriptor with F (First) bit set to ‘1’.
- **EOF** (End Of Frame descriptor):   Descriptor with L (Last) bit set to ‘1’.

F and L bits are set by the CPU before releasing a descriptor to the MV64360/1/2 for transmission.

A frame starts with a SOF descriptor and ends with a EOF descriptor. A frame can consist of one buffer or split over many buffers. If a frame is stored in one buffer, the associated descriptor will have both the F and L bits set to ‘1’. In a non-frame oriented protocol (e.g. BISYNC or UART), it is recommended that both F and L bits be set to ‘1’ for each buffer.

16.16.2Transmit SDMA Flow

The following steps are executed during a normal transmit SDMA process:

1. Before enabling a SDMA Tx channel the CPU must prepare a valid descriptor with the owner bit set to ‘1’.
2. The CPU must then write the first descriptor address to both SCTDP and SFTDP registers.
3. The CPU issues a Transmit Demand command. The SDMA controller will then fetch the first descriptor and will start the SDMA process.
4. When buffer transmission is completed, the SDMA will close the buffer descriptor by setting the correct transmit status and writing ‘0’ in the Owner Bit, returning the buffer to the CPU.

16.16.3Retransmit in HDLC (LAP-D) mode

When working in collision mode (see MPSC section), the MV64360/1/2 retransmits if collision occurs before the SDMA fetches the 3rd descriptor. If the frame consists of more than two buffers, the user must assure that there is enough data in the first two buffers to compensate for this behavior. The MV64360/1/2 can buffer up to 256 bytes in its internal Tx FIFO. This should be considered when preparing a LAP-D transmit frame.

16.16.4Transmit SDMA Notes

The transmit SDMA process is *frame oriented*.

The Transmit SDMA does not clear the frame’s first descriptor ownership bit until the last descriptor associated with this frame is closed. The transmit SDMA then writes ‘0’ to the first descriptor Owner bit and generate an interrupt if the SDMA Descriptor register’s EI bit [23] of the first descriptor is set.

The transmit SDMA stops the DMA process whenever it reaches a descriptor with NULL (0x00000000) value in the NDP field or when it fetches a descriptor with Owner Bit set to ‘0’. In such cases, the SDMA controller clears the TxD bit before returning to IDLE state.

In normal operation, the transmit SDMA never expects to find a NULL NextDescriptorPointer or Not-Owned descriptor in the middle of a frame. When this occurs, the transmit SDMA controller aborts, the TxD bit is cleared, and a Tx RESOURCE ERROR maskable interrupt is generated.

Note

In collision mode, if a collision occurs exactly one clock cycle after a resource error, the MV64360/1/2 ignores the resource error and retransmit the frame.

When the CPU wants to interfere with the transmit process without corrupting the ongoing transmit process, it can issue a STOP command by writing ‘1’ to the STD bit in the SDMA command register. The transmit SDMA controller stops after completing the transmission of the active frame.

When issuing an STD command TXD is reset to ‘0’ upon entering IDLE state. The CPU can then issue a new Transmit Demand command to restart the SDMA process.

16.17Receive SDMA

16.17.1Receive SDMA Definitions

Table 107: SDMA Definitions

Term	Definition
SOF	Start Of Frame descriptor Descriptor with F (First) bit set to ‘1’.
EOF	End Of Frame descriptor Descriptor with L (Last) bit set to ‘1’.

F and L bits are set by the CPU before releasing a descriptor to the MV64360/1/2.

A frame starts with an SOF descriptor and ends with an EOF descriptor. A frame can be contained in one buffer or split over many buffers. If a frame is stored in one buffer, the associated descriptor will have both F and L bits set to ‘1’.

16.17.2Receive SDMA Flow

The following steps are executed during a normal transmit SDMA process:

1. Before enabling a SDMA Rx channel the CPU must prepare a valid descriptor with the owner bit set to ‘1’.
2. The CPU must then write the descriptor address to the SCRDP register before enabling the receive SDMA channel.
3. The CPU writes ‘1’ to the ERD bit in the SDCM register, enabling the receive SDMA channel.
4. Normally the receive SDMA controller will then run continuously, processing received data from the MPSC.

Note

The receive SDMA controller never expects to encounter a descriptor with owner bit set to ‘0’ or a NULL value (0x00000000) in the NDP field. If this occurs, the receive SDMA aborts and a maskable Rx RESOURCE ERROR interrupt is generated.

Use the receive abort command for the CPU to stop the receive SDMA. It is the CPU’s responsibility to properly restart the descriptor chain.

16.18SDMA Interrupt Cause and Mask Register (SDI and SDM)

Each SDMA channel has two types of interrupt cause events - resource error events and descriptor closed events. See

When a receiving SDMA encounters a NULL descriptor pointer or a not owned descriptor, a Resource Error interrupt is generated. A Resource Error interrupt is generated whenever a transmit SDMA encounters a NULL descriptor pointer or a not-owned descriptor in a middle of a frame.

Note

When the MV64360/1/2 encounters a descriptor with an Owner bit set to ‘0’, it still expects to find that all the other fields of the descriptor are legitimate. A descriptor with an Owner bit set to ‘0’, with non-legitimate fields (such as Start Of Frame descriptor with F (First) bit not set to ‘1’), can lead to unpredictable behavior.

When a SDMA channel closes a descriptor with the SDMA Descriptor register’s EI bit [23] (see Table 102 on page 292) set to ‘1’, a Descriptor Closed interrupt is generated.

Notes

- If the SDMA Configuration register’s RIFB bit [9] (see Table 577 on page 652) is set, an interrupt is generated by the Rx channel only on receive frame boundaries.
- The correct operation of the frame level interrupt requires all Rx descriptors to have their EI bit set.

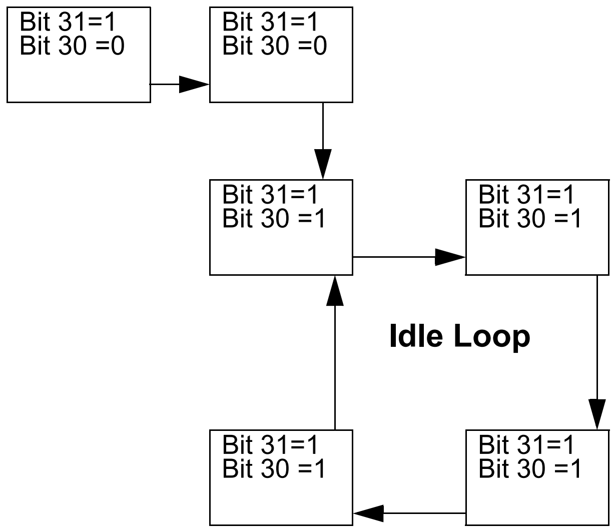
The interrupt cause bits are set upon hardware events and are cleared by the software writing ‘0’. The Mask register’s bits do not effect the actual setting of the interrupt cause bits. The Mask register’s bits only determine upon which cause event, should an interrupt pin be asserted.

16.19SDMA in Auto Mode

The CPU can set bit 30 in the command/status field of transmit or receive descriptors directing the MV64360/1/2 to work in Auto Mode.

When working with an Auto Mode descriptor, the MV64360/1/2 SDMA works as usual except that it does not clear the Ownership bit when closing the descriptor. The CPU can use this for example to cause the MV64360/1/2 to transmit endlessly (until CPU intervention).

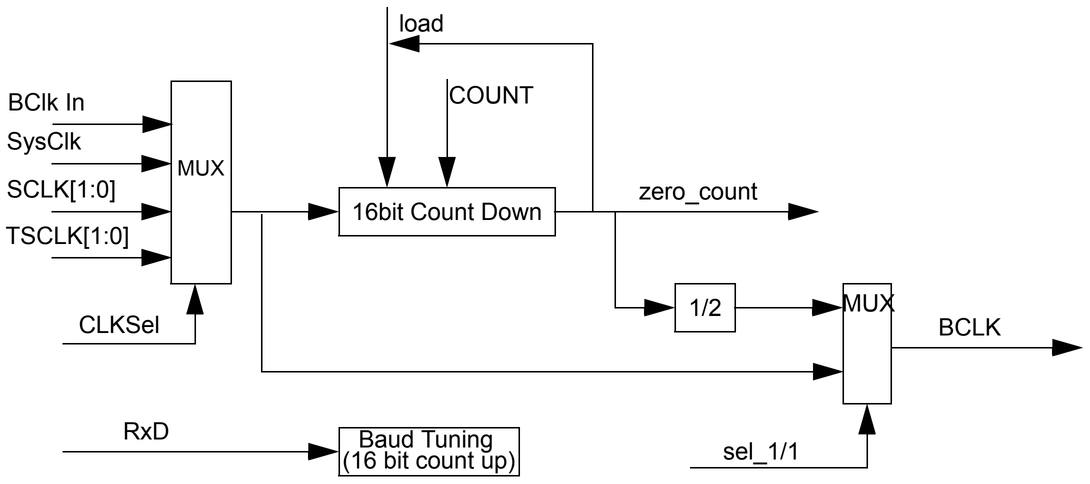
Figure 68: Using Auto Mode to Create Idle Loop



Section 17. Baud Rate Generators (BRG)

There are two baud rate generators (BRGs) in the MV64360/1/2. Figure 69 shows a BRG block diagram.

Figure 69: Baud Rate Generator Block Diagram



17.1 BRG Inputs and Outputs

There are five clock inputs to the baud rate generators (BRGs). One MPP pin can be programmed to function as clock input to the BRGs. Additionally, each of the MPSCs input clocks can be used as a BRG clock. Finally, SysClk is also an option.

Note

When using the BRG as the clock source of an MPSC, this BRG can not use this MPSC SCLK or TSCLK as reference clock.

When a BRG is enabled, it loads the Count Down Value (CDV), from the BRG configuration register, into its count down counter. When the counter expires (i.e. reaches zero), the BRG clock output, BCLK, is toggled and the counter reloads.

17.2 BRG Baud Tuning

A baud tuning mechanism can be used to adjust the generated clock rate to the receive clock rate.

When baud tuning is enabled, the baud tuning mechanism monitors for a start bit, i.e. High-to-Low transition. When a start bit is found, the baud tuning machine measures the bit length by counting up until the next Low-to-High transition. The count-up value of the BRG is then loaded into the Count Up Value (CUV) register and a maskable interrupt is generated signaling the CPU that the bit length value is available. The CPU reads the value from the CUV and adjusts the CDV to the requested value.

The CUV can be used to adjust the CDV, in the BRG configuration register, to the requested value.

Section 18. IDMA Controller

The MV64360/1/2 has four independent IDMA engines. The IDMA engines optimize system performance by moving large amounts of data without significant CPU intervention.

A typical IDMA usage is to move data from an MV64360/1/2 interface (e.g. PCI-X) into the local DDR SDRAM. The data in the DRAM is processed by the CPU and driven back by the DMA to the external interface.

Each IDMA engine can move data between any source to any destination. It can transfer a single data buffer of up to 16 MB. It can also run in chain mode, in which each buffer has its own descriptor. The link list of descriptors can be placed in any of the MV64360/1/2 interfaces. For example, the DMA engine can fetch descriptors from the integrated SRAM, and transfer data from the PCI to the DRAM.

Control the IDMA transfer rate with external hardware using the DMAReq# pins. Controlling the transfer rate is useful when:

- The amount of data available for transfer is dynamically changing.
- The transfer destination is temporarily incapable to receive more data.

18.1 IDMA Operation

IDMA unit contains four 512 byte buffers, one buffer per DMA channel.

When a channel is activated, data is being read from the source into the buffer, and then written to the destination. Read and write are handled independently. The DMA engine transfers the buffer in chunks of 8 up to 128 bytes. It reads from the source as long as it has place in the buffer. It writes to the destination, as long as there is valid data in the buffer to be transferred. This independency, results in concurrent reads and writes, and maximum utilization of the DMA interface.

Since the four channels share the same resources, arbitration is required. The four channels use a configurable round-robin arbiter that allows different bandwidth allocation to each channel, see 18.5 “Arbitration” on page 311.

18.2 IDMA Descriptors

Each IDMA Channel Descriptor consists of four 32-bit registers. Each channel can be configured to work in a 64K descriptor mode, or with 16M descriptor mode, as shown in Figure 70.

Figure 70: IDMA Descriptors

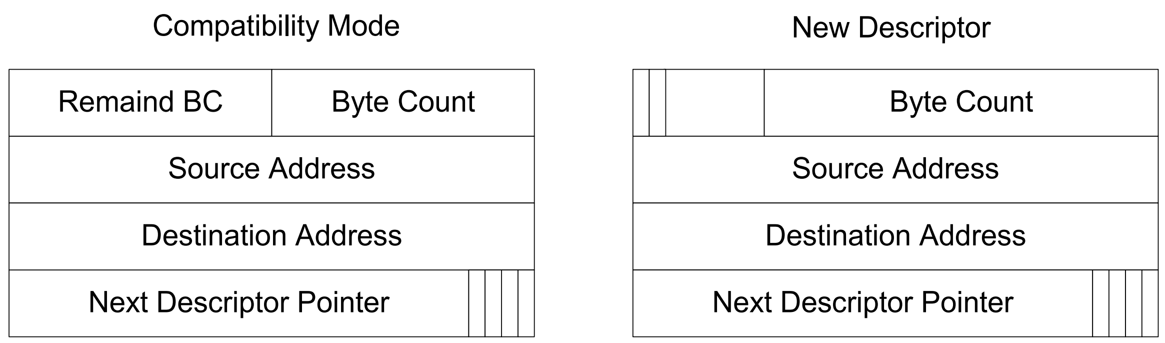


Table 108: DMA Descriptor Definitions

DMA Descriptor	Definition
Byte Count	Number of bytes of data to transfer. The maximum number of bytes which the IDMA controller can be configured to transfer is 64 KB-1 (16-bit register) in 64K descriptor mode or 16 MB-1 (24-bit register) in the 16M descriptor mode. This register decrements at the end of every burst of transmitted data from the source to the destination. When the byte count register is 0, or the End of Transfer pin is asserted, the IDMA transaction is finished or terminated.
Source Address	Bits[31:0] of the IDMA source address. According to the setting of the Channel Control register, this register either increments or holds the same value.
Destination Address	Bits[31:0] of the IDMA destination address. According to the setting of the Channel Control register, this register either increments or holds the same value.
Pointer to the Next Descriptor	Bits[31:0] of the IDMA Next Descriptor address for chained operation. The descriptor must be 16 sequential bytes located at 16-bytes aligned address (bits[3:0] are 0). **NOTE:** Only used when the channel is configured to Chained Mode.

18.3 IDMA Address Decoding

The four DMA channels share eight address windows. Each address window can be individually configured.

With each IDMA transaction, the IDMA engine first compares the address (source, destination, or descriptor) against its address decoding registers. Each window can be configured to different target interface. Address comparison is done to select the correct target interface (DRAM, PCI...).

Note

Unlike the GT-64240/60 devices, the MV64360/1/2 IDMA has its own address decoding map that is decoupled from the CPU interface address decoding windows.

If the address does not match any of the address windows, an interrupt is generated and the IDMA engine is stopped.

The PCI interface supports 64-bit addressing. Four of the eight address windows have an upper 32-bit address register. To access the PCI bus with 64-bit addressing cycles (DAC cycles), assign one (or more) of these four windows to target the PCI bus. The address generated on the PCI bus is composed of the window base address and the High Remap register.

For the DMA engine to avoid accessing forbidden address space (due to a programing bug), each channel uses access protection logic that prevents it from read/write access to specific address windows. In case of access violation, the IDMA halts and an interrupt is asserted.

Every access to the integrated SRAM or to the DRAM might require a snoop action on the CPU bus to maintain cache coherency. Any of the address decoding windows targeted to the DRAM can be marked as cache coherent region. A DMA access to these regions result in a snoop action on the CPU bus.

The IDMA also supports an address override feature. Each of the source, destination, or next descriptor address can be configured to use the override feature by using the Channel Control register's `SAddrOvr` bits [22:21], `DAddrOvr` bits [24:23], and `NAddrOvr` bits [26:25] (Table 603 on page 669). When the respective field is set to 0x1, the transaction target interface, attributes, and upper 32-bit address are taken from Base Address register (BAR) 1. When set to 0x2, these items are taken from BAR 2. And when set to 0x3, they are taken from BAR 3.

This address override feature, enables additional address decoupling. For example, it allows the use of the same source and destination addresses while the source is targeted to one interface and destination to a second interface.

Note

The MV64360/1/2 address override feature differs from the GT-64240/60 PCI override feature. The MV64360/1/2 allows for address override to any of the MV64360/1/2 interfaces.

For full details on IDMA address decoding, see 7.3 “IDMA Address Decoding” on page 91.

18.4 IDMA Channel Control

Each IDMA Channel has its own unique control register where certain IDMA modes are programmed. Following are the bit descriptions for each field in the control registers. For detailed registers description, see J.4 “IDMA Channel Control Registers” on page 669.

18.4.1 Address Increment/Hold

The IDMA engine supports both increment and hold modes, on both source and destination addresses.

If the Channel Control register's `SrcHold` bit [3] is set to ‘0’, the IDMA automatically increments the source address with each transfer. If set to ‘1’, the source address remains constant throughout the IDMA burst.

Similarly, If the same register's `DestHold` [5]is set to ‘0’, the IDMA automatically increments the destination address. If the set to ‘1’, the destination address remains constant throughout the IDMA burst.

Setting the `SrcHold` or `DestHold` bits is useful when the source/destination device is accessible through a constant address. For example, if the source/destination device is a FIFO, it is accessed with a single address, while data is being popped/pushed with each IDMA burst.

Note

When using Hold mode, the address is restricted to be aligned to the Burst Limit setting, see the Channel Control (Low) register's `ScrBurstLimit` bits [8:6] and bits[2:0]

18.4.2 Burst Limit

The whole IDMA byte count is chopped into small bursts.

The burst limit can vary from 8 to 128 bytes in modulo-2 steps (i.e. 8, 16..., 128). It determines the burst length of IDMA transaction against the source and destination. There are separate BurstLimit parameters for source and destination. For example, setting the source burst limit to 32 bytes and the destination burst limit to 128 bytes, means that the IDMA reads 32 bytes from the source, and then writes the data to the destination after combining to 128 bytes. The IDMA continues this read/write loop until transfer of the whole byte count is complete.

The burst limit setting is affected by the source and destination characteristics, as well as system bandwidth allocation considerations.

Note

Regardless of the burst limit setting, the fetch of a new descriptor is always a 16 bytes burst. This implies that descriptors cannot be located in devices that don't support such bursts. Particularly, they can not be located in 8 or 16-bit devices on the MV64360/1/2 device bus (see 12.3 "Data Pack/Unpack and Burst Support" on page 150).

If an IDMA accesses a cache coherent DRAM regions, the burst limit must not exceed 32 bytes.

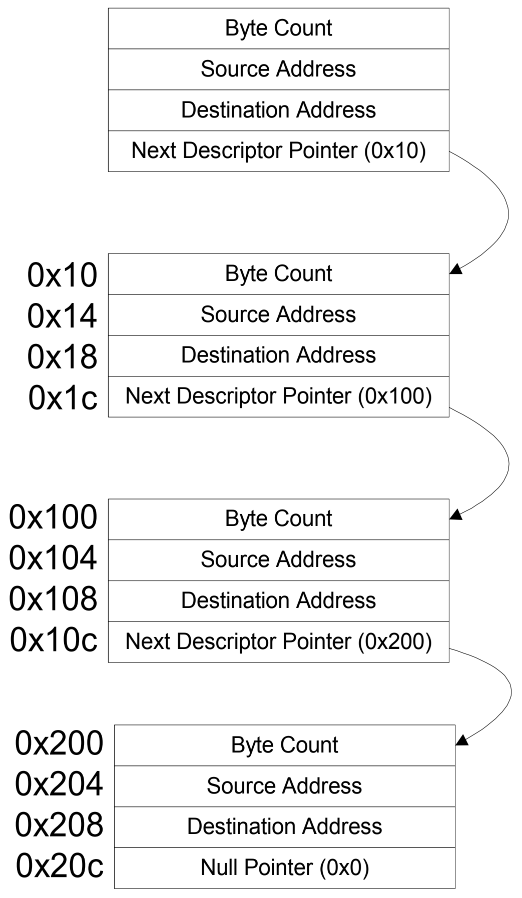
18.4.3 Chain Mode

When the Channel Control register's `ChainMode` bit [9] is set to '0', chained mode is enabled.

In chain mode, at the completion of one buffer transfer, the Pointer to Next Descriptor provides the address of a next IDMA descriptor. If it is a NULL pointer (value of '0'), it indicates that this is the last descriptor in the chain. If not, the DMA engine fetches the new descriptor, and starts transferring the new buffer.

Figure 71 shows an example of an IDMA descriptors chain.

Figure 71: Chained Mode IDMA



Fetch next descriptor can be forced by the Channel Control (Low) register's `FetchND` bit [13].

Setting this bit to '1' forces a fetch of the next descriptor based on the value in the Pointer to Next Descriptor register.

This bit can be set even if the current IDMA has not yet completed. In this case, the IDMA engine completes the current burst read and write and then fetches the next descriptor. This bit is reset back to '0' after the fetch of the new descriptor is complete. Setting `FetchND` is not allowed if the descriptor equals Null.

Note

If using the `FetchND` bit while the current DMA is in progress, the DMA Control (Low) register's `Abr` bit [20] must be set.

The first descriptor of a chain can be set directly by programing the channels registers, or can be fetched from memory, using the FetchND bit. If fetched from memory, the next descriptor address must be first written to the Next Descriptor Pointer register of the channel. The channel then must be enabled by setting the Channel Control (Low) register's `ChanEn` bit [12] to '1' (see 18.4.4 "Channel Activation" on page 306) and setting `FetchND` to '1'.

When the IDMA transfer is done, an IDMA completion interrupt is set. When running in chain mode, the Channel Control (Low) register's `IntMode`, bit [10] of the, controls whether to assert an interrupt on the completion of every byte count transfer or only with last descriptor byte count completion. If set to '0', the Interrupt Mask register's `Comp` bit [0] is set every time the IDMA byte count reaches '0'. If set to '1', the Comp Interrupt bit is asserted when both the Pointer to Next Descriptor Register has a NULL value and byte count is '0'.

If `ChainMod` is set to '1', chained mode is disabled and the Pointer to Next Descriptor register is not loaded at the completion of the IDMA transaction.

Notes

- In non-chained mode the Byte Count, Source, and Destination registers must be initialized prior to enabling the channel.
- If reading a new descriptor results in parity/ECC error indicated by the unit from which the descriptor is being read, the channel halts. This is done in order to prevent destructive reads/writes, due to bad source/destination pointers.
- If using FetchND to fetch the first descriptor and the `IntMode` bit is set to '0', a dummy DMA completion interrupt is asserted with this first fetch. Although, no data has been transferred.

18.4.4 Channel Activation

Software channel activation is done via the Channel Control (Low) register's `ChanEn` bit [12].

When set to '0', the channel is disabled. When set to '1', the IDMA is initiated based on the current setting loaded in the channel descriptor (i.e. byte count, source address, and destination address). An active channel can be temporarily stopped by clearing `ChanEn` bit and then continued later from the point it was stopped by setting `ChanEn` bit back to '1'.

Clearing the `ChanEn` bit during IDMA operation does not guarantee an immediate channel pause. The IDMA engine must complete transferring the last burst it was working on. Software can monitor the channel status by reading `ChanAct` bit [14].

To restart a suspended channel in non-chained mode, the `ChanEn` bit must be set to '1'. In Chained mode, the software must find out if the first fetch took place. If the fetch did take place, only `ChanEn` bit is set to '1'. If the fetch did not take place, the `FetchND` bit must also be set to '1'.

The `ChanAct` bit is read only. If set to '0', the channel is not active. If set to '1', the channel is active. In non-chain mode, this bit is de-asserted when the byte count reaches zero. In chain-mode, this bit is de-asserted when pointer to next descriptor is NULL and byte count reaches zero.

If `ChanEn` bit is set to '0' during IDMA transfer, `ChanAct` bit toggles to '0' as soon as the IDMA engine finishes the last burst it is working on.

To abort an IDMA transfer in the middle, the software must set `Abr` bit to '1'. Setting this bit has a similar affect to clearing `ChanEn` bit. However, it guarantees a smooth transfer of the IDMA engine to idle state. As soon as the IDMA is back in idle state, the `Abr` bit gets cleared, allowing the software to re-program the channel.

Note

If the byte count is smaller that the burst limit setting, the source and destination addresses must be 64-bit aligned.

If the close descriptor feature is used, only set the Abr bit after first clearing the `ChanEn` bit and then the `ChanAct` bit.

18.4.5 Source and Destination Addresses Alignment

The IDMA implementation maintains aligned accesses to both source and destination.

If source and destination addresses have different alignments, the IDMA performs multiple reads from the source to execute a write of full BurstLimit to the destination. For example, if the source address is 0x4, the destination address is 0x100, and BurstLimit of both source and destination is set to 8 bytes, the IDMA perform two reads from the source. First 4 bytes from address 0x4 then 8 bytes from address 0x8, and only then performs a write of 8 bytes to address 0x100.

This implementation guarantees that all reads from the source and all writes to the destination have all byte enables asserted (except for the IDMA block edges, in case they are not aligned). This is especially important when the source device does not tolerate read of extra data (destructive reads) or when the destination device does not support write byte enables.

Note

This implementation differs from the GT-64120 and GT-64130 devices. No SDA bit is required since the MV64360/1/2 implementation keeps accesses to both source and destination aligned.

18.4.6 Demand Mode

The IDMA channel can be triggered by software via `ChanEn` bit (block mode) or by external assertion of DMAReq# pin (demand mode). Setting the Channel Control (Low) register's `DemandMode` bit [11] to '0', sets the channel to operate in demand mode.

Each channel is coupled to the DMAReq# and DMAAck# pins when working in demand mode. DMAReq# is the external trigger to activate the channel. DMAAck# is the channel response, notifying the external device that its request is being served.

Both DMAReq# and DMAAck# are multiplexed on MPP pins. If setting a channel to demand mode, the DMAReq# pin is mandatory. Setting a channel to demand mode without configuring an MPP pin to act as the channels DMAReq# causes the channel to hang. See Section 26. "Pins Multiplexing" on page 343 section for more information.

Note

Program the number of SysClk cycles that DMAAck# is asserted through the Channel Control (Low) register's `DMAAck_Width` bit [4].

When running in demand mode, the IDMA moves a new BurstLimit of data upon demand, rather than continuos bursts from source to destination. This mode is required when the source device does not have the whole byte count in advance. It triggers a new burst limit transfer when it has a burst count available data to transfer. It can also be used in the compliment case, where the destination device cannot absorb the whole byte count, but only burst limit at a time.

The IDMA engine distinguishes between a DMAReq# generated by the source device and DMAReq# generated by the target device via the Channel Control (Low) register's `DMAReqDir` bit [15]. If DMAReq# is generated by the source (DMAReqDir is set to '0'), the IDMA reads a new BurstLimit of data from source with each new DMAReq assertion. However, it writes to the destination device whenever it can transfer a full BurstLim. In the alignment example in 18.4.5 "Source and Destination Addresses Alignment" on page 307, the first write to the destination occurs after two assertions of DMAReq# by the source. If DMAReq# is asserted by the destination (`DMAReqDir` is set to '1'), the DMA writes a new BurstLimit of data to the destination device with each new DMAReq assertion. In this case, a read from the source occurs regardless of DMAReq# assertion.

Note

This implementation is different than the one in the GT-64120 and GT-64130. In these devices, each DMAReq# assertion results in a single read from source and write to the destination.

DMAReq# can be treated as level or edge triggered input, depending on the setting of the Channel Control (Low) register's `DMAReqMode` bit [16]. When the device DMAReq# assertion is tightly coupled to the DMAAck# signal, an edge trigger DMAReq# might be needed, to prevent a redundant DMAReq# assertion due to late DMAReq# de-assertion.

Note

The edge triggered DMAReq# is a new feature not supported by the GT-64120 and GT-64130. In these devices, the problematic DMAReq# de-assertion timing is solved via the MDREQ bit. This bit is no longer supported.

The DMAAck# output pin indicates to the requesting device that the IDMA engine has finished transferring the current burst. DMAAck# can be configured to assert with the read from the source, with the write to destination, via the Channel Control (Low) register's `DMAAckDir` bits [30:29]. Setting DMAAck# to '1' results in DMAAck assertion with write access to the destination device.

Note

The setting for DMAAckDir must match DMAReqDir.

Since the Device interface unit has a queue of transactions, actual IDMA transaction to the device bus might take place many cycles after the IDMA access to the Device interface unit completed. There are devices that expect to see the DMAAck# signal asserted along with the actual transaction on the device bus, rather than with the IDMA access to the Device interface unit completion. When setting the Channel Control (Low) register's `DMAAckMode` bit [17] to '1', DMAAck is asserted with the actual transaction on the device bus. In this case, DMAAck# signal has the same timing characteristics as CSTiming# signal (see 12.2 "Device Timing Parameters" on page 148). When setting the DMAAckMode bit to '0', DMAAck is asserted, as soon as the IDMA engine issues the transaction to the target unit. DMAAck# is asserted for one or two cycles, depending on the Channel Control (Low) register's DMAAck_Width bit [4] setting.

Note

The `DMAAckMode` bit is only available for IDMA access to the device bus. Setting this bit to '1' while accessing any other interface than the device bus results in no DMAAck# assertion at all.

When using demand mode, the trigger of the channel can be configured to come from the timer rather than from DMAReq# pin. Each of the four IDMA channels is coupled to one of the eight MV64360/1/2 timers (channel0 to timer0, channel1 to timer1, and so on). When the channel is configured to demand mode, setting the Channel Control (Low) register's `TimerReq` bit [28] to '1' results in a timer trigger rather than DMAReq# trigger. In this case, when the timer/counter reaches the terminal count, an internal IDMA request is set and a new IDMA transfer is initiated.

This mode is useful to generate an IDMA transfer for every 'n' cycle. Set the timer to 'n' cycles, activate it, and then activate the IDMA channel in demand mode with `TimerReq` bit set. The IDMA engine generates a new burst every 'n' cycles.

Note

When running in demand mode and using chain IDMA, when reaching byte count ‘0’, the MV64360/1/2 fetches a new descriptor regardless of the DMAReq#. The DMAReq# affects only IDMA access to data, not to descriptors. This means that chain descriptors must always be ready for fetch.

When running in demand mode, the MV64360/1/2 does not issue a new burst read request from the source before completing the write transaction to the destination.

18.4.7 End Of Transfer

The MV64360/1/2 supports IDMA termination in the middle not only by software, but also by external hardware via EOT pins. Each channel has its own EOT input pin (EOT[0] for channel0, EOT[1] for channel1...). EOT[7:0] pins are multiplexed on MPP pins. To use this feature, the MPP lines must be programmed to act as EOT pins (see Section 26. “Pins Multiplexing” on page 343). EOT pins are edge trigger pins.

Setting the `EOTEn` bit [18] to ‘1’ enables this feature. The affect of EOT assertion can be configured via the `EOT-Mode` bit [19].

If the `EOTMode` bit is set to ‘0’, EOT assertion, when working in chain mode, causes fetching of a new descriptor from memory (if pointer to next descriptor is not equal to NULL) and executing the next IDMA. This is equivalent to executing fetch next descriptor in software.

If the `EOTMode` bit is set to ‘1’, EOT assertion causes the channel to halt. This is equivalent to setting the Abr bit to ‘1’ via the software.

If the IDMA channel is in non-chain mode, the `EOTMode` bit is not relevant. EOT assertion causes the current IDMA transfer to be stopped without further action.

A DMA completion interrupt is asserted (if not masked) upon IDMA termination with EOT.

Note

The IDMA engine stops only after finishing the current burst. For example, if it is programed to a burst limit of 64 bytes and EOT is sampled active in the middle of the 64 bytes read, the IDMA engine completes the read, performs the 64 byte write, and then halts. When using EOT, the source and destination must be 64-bit aligned.

18.4.8 Descriptor Ownership

A typical application of chain mode IDMA involves the CPU preparing a chain of descriptors in memory and then preparing buffers to move from source to destination. Buffers may be dynamically prepared, this means once a buffer was transferred the CPU can prepare a new buffer in the same location to be sent. This application requires some handshake between the IDMA engine and the CPU.

When working with the 16M descriptor mode, Channel DMA Byte Count register `Own` bit[31] acts as an ownership bit (Table 591 on page 662). If reset to ‘0’, the descriptor is owned by the MV64360/1/2 IDMA. If set to ‘1’, it is owned by the CPU. Once the CPU prepares a buffer to be transferred, it resets the ownership bit. This indicates that the buffer is owned by the IDMA. Once the IDMA completes transferring the buffer, it closes the descriptor by writing back the upper byte of Byte Count register (bits[31:24]) with MSB set to ‘1’. This indicates to the CPU that the buffer was transferred. When the CPU recognizes that it owns the buffer, it is allowed to place a new buffer to be transferred. An attempt by the IDMA to fetch a descriptor that is owned by CPU (which means CPU did not pre-pare a new buffer yet), results in an interrupt assertion and an IDMA channel halt.

Note

This feature is not supported in 64K descriptor mode.

The Descriptor is closed when the byte count reaches ‘0’ or when transfer is terminated in the middle via EOT or the fetch next descriptor command. In this case, the transfer may end with some data remaining in the buffer pointed by the current descriptor.

When working in 64K descriptor mode, when closing the descriptor, the IDMA engine writes the left byte count to the upper 16-bit of the byte count field of the descriptor. This is useful if an IDMA is terminated in the middle and a CPU might want to re-transmit the left byte count. In case the IDMA ended properly (all byte count was transferred), a ‘0’ value is written back to the descriptor.

When working with the 16M descriptor mode, there is an alternative way to signal to the CPU that the descriptor was not completely transferred. In this case, the IDMA engine rather than writing back the remaining byte count, it writes back to only bits[31:24] of the descriptor’s `ByteCount` field, with bit[30] indicating whether the whole byte count was transferred (0) or terminated before transfer completion (1). Bits[29:24] are meaningless.

Each IDMA channel has a Current Descriptor Pointer register (CDPTR) associated with it. This register is used for closing the current descriptor before fetching the next descriptor. The register is a read/write register but the CPU must not write to it. When the NPTR (Next pointer) is first programed, the CDPTR reloads itself with the same value written to NPTR. After processing a descriptor, the IDMA channel updates the current descriptor using CDPTR, saves NPTR into the CDPTR, and fetches a new descriptor.

18.4.9 Buffer Transfer Upon Demand

The MV64360/1/2 EOT implementation enables hardware control on buffers transfer. The concept is similar to the Demand mode (see 18.4.6 “Demand Mode” on page 307). Instead of transferring a BurstLimit amount of data upon demand, EOT enables a transfer of a full buffer upon demand.

To activate this mode:

1. Set byte count to ’0’.
2. Set Current Descriptor Pointer to ’0’.
3. Set the valid Source and Destination addresses. These addresses are not used but should be valid to prevent address mismatch error.
4. Set the Next Descriptor Pointer to point to the first descriptor.
5. Activate the channel in non-chain mode by setting the Channel Control (Low) register’s `ChainMode` bit [9] to ’1’, EOTEn bit [18] to ’1’, and EOTMode bit [19] set to ’0’ (see Table 603 on page 669).

With each EOT assertion, the DMA fetches a new descriptor, transfers the required byte count (as specified in the descriptor), and closes the descriptor (if close descriptor is enabled).

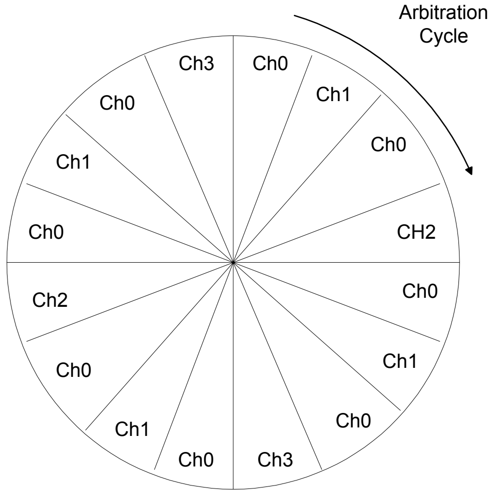
Note

A new EOT must not be asserted prior to the completion (close descriptor) of a previous buffer transfer. Use DMA completion interrupt as indication for transfer completion. To represent DMA completion, use one of the four MV64360/1/2 interrupt pins and set `InterruptMode` bit[10] to ’0’.

18.5 Arbitration

The IDMA controller has a programmable round-robin arbiter for the four DMA channels. Each channel can be configured to have different bandwidth allocation. Figure 72 shows an example arbitration cycle.

Figure 72: Configurable Weights Arbiter



The user can define each of the 16 slices of this “pizza arbiter”. In Figure 72, channel0 gets 50% of the bandwidth, channel1 25%, channel2 and channel3 12.5% each. At each clock cycle, the arbiter samples all channels requests and gives the bus to the next agent according to the “pizza”.

18.6 Big and Little Endian

The MV64360/1/2 supports Little and Big Endian conventions.

The DMA data endianess is determined via the Channel Control (High) register’s Endianess bit [0] (see Table 604 on page 671). The DMA engine organizes the read data in its buffer, based on this setting.

The internal registers of the MV64360/1/2 are always set in Little Endian mode. If the DMA endianess is configured to Big Endian, descriptors fetched from memory might need to be byte swapped before being placed in the device registers. Byte swapping is enabled by Channel Control (High) register’s `DescBS` bit [1].

In addition, the MV64360/1/2 supports access to Big and Little Endian devices on the PCI bus. PCI endianess is controlled via the PCI Command register’s `MByteSwap` bit [0] and `MWordSwap` bit [10] (see Table 378 on page 535). In addition, each address decoding windows targeted to the PCI can be configured to different endianess, in case the DMA channels target different endianess oriented PCI devices (see 13.10 “Data Endianess” on page 172).

18.7 DMA Interrupts

The IDMA interrupts are registered in the IDMA Interrupt Cause register. Upon an interrupt event, the corresponding cause bit is set to ‘1’. It is cleared upon a software write of ‘0’.

The IDMA Mask registers controls whether an interrupt event causes an interrupt assertion. The setting of the mask register only affects the interrupt assertion, it has no affect on the cause register bits setting.

The following interrupt events are supported per each channel:

- DMA completion
- DMA address out of range
- DMA access protect violation
- DMA write protect violation
- DMA descriptor ownership violation

In case of an error condition (address out of range, access protect violation, write protect violation, descriptor ownership violation), the IDMA transaction address is latched in the Address Error register. Once an address is latched, no new address (due to additional errors) can be latched, until the current address being read.

Section 19. Address and Data Integrity

The MV64360/1/2 supports address and data integrity on most of its interfaces.

- It supports parity checking and generation on the CPU, device, and PCI busses
- It supports ECC checking and generation on the SDRAM bus
- It supports parity on its integrated SRAM
- It supports CRC checking and generation on the Ethernet and Serial ports.

Note

Integrated SRAM only applies to the MV64360 and MV64361 devices.

19.1 CPU Parity Support

The CPU interface generates and checks data parity and address parity.

On CPU writes, the MV64360/1/2 samples data parity driven by the CPU with each data.

When a parity error occurs, the MV64360/1/2 generates an interrupt and latches the following:

- Bad address in the CPU Error Address register.
- Data in the CPU Error Data register.
- Parity in the CPU Error Parity register.

The same occurs on master read transactions on the 60x bus. If read data is received with bad data parity, address, data and parity are latched and an interrupt is generated.

On CPU reads, the MV64360/1/2 drives parity with each read data it drives on the CPU bus.

As a master on the 60x bus, the MV64360/1/2 also drives parity with each write data it drives on the bus.

The MV64360/1/2 also samples address parity driven by the CPU with the address. In case of bad address parity detection, it latches the bad address and parity in CPU Error Address register and generates an interrupt. The transaction is not propagated to the target.

As a bus master, the MV64360/1/2 drives address parity with the address it drives on the bus.

Note

In case of multiple errors are detected, the address, data, and parity are latched in the corresponding registers only for the first error. Latching of new data into these registers is only enabled when reading the CPU Error Address (Low) register. The interrupt handler must read this register last.

19.2 DDR SDRAM ECC

The MV64360/1/2 implements Error Checking and Correction (ECC) on accesses to the SDRAM. It supports detection and correction of one data bit errors, detection of two errors, and detection of three or four bit errors within the same nibble.

19.2.1 ECC Calculation

Each of the 64 data bits and eight check bits has a unique 8-bit ECC check code, as shown in Table 109. For example, data bit 12 has the check value of 01100001, and check bit 5 has the check value of 00100000.

Table 109: ECC Code Matrix

Check Bit	Data Bit	ECC Code Bits								Number of 1s in syndrome
		7	6	5	4	3	2	1	0	
	63	1	1	0	0	1	0	0	0	3
	62	1	1	0	0	0	1	0	0	3
	61	1	1	0	0	0	0	1	0	3
	60	1	1	0	0	0	0	0	1	3
	59	1	1	1	1	0	1	0	0	5
	58	1	0	0	0	1	1	1	1	5
4		0	0	0	1	0	0	0	0	1
3		0	0	0	0	1	0	0	0	1
	57	1	1	1	0	0	0	0	0	3
	56	1	0	1	1	0	0	0	0	3
	55	0	0	0	0	1	1	1	0	3
	54	0	0	0	0	1	0	1	1	3
	53	1	1	1	1	0	0	1	0	5
	52	0	0	0	1	1	1	1	1	5
5		0	0	1	0	0	0	0	0	1
2		0	0	0	0	0	1	0	0	1
	51	1	0	0	0	0	1	1	0	1
	50	0	1	0	0	0	1	1	0	3
	49	0	0	1	0	0	1	1	0	3
	48	0	0	0	1	0	1	1	0	3
	47	0	0	1	1	1	0	0	0	3
	46	0	0	1	1	0	1	0	0	3
	45	0	0	1	1	0	0	1	0	3
	44	0	0	1	1	0	0	0	1	3
	43	1	0	1	0	1	0	0	0	3

Table 109: ECC Code Matrix (Continued)

Check Bit	Data Bit	ECC Code Bits								Number of 1s in syndrome
		7	6	5	4	3	2	1	0	
	42	1	0	1	0	0	1	0	0	3
	41	1	0	1	0	0	0	1	0	3
	40	1	0	1	0	0	0	0	1	3
	39	1	0	0	1	1	0	0	0	3
	38	1	0	0	1	0	1	0	0	3
	37	1	0	0	1	0	0	1	0	3
	36	1	0	0	1	0	0	0	1	3
	35	0	1	0	1	1	0	0	0	3
	34	0	1	0	1	0	1	0	0	3
	33	0	1	0	1	0	0	1	0	3
	32	0	1	0	1	0	0	0	1	3
	31	1	0	0	0	1	0	1	0	3
	30	0	1	0	0	1	0	1	0	3
	29	0	0	1	0	1	0	1	0	3
	28	0	0	0	1	1	0	1	0	3
	27	1	0	0	0	1	0	0	1	3
	26	0	1	0	0	1	0	0	1	3
	25	0	0	1	0	1	0	0	1	3
	24	0	0	0	1	1	0	0	1	3
	23	1	0	0	0	0	1	0	1	3
	22	0	1	0	0	0	1	0	1	3
	21	0	0	1	0	0	1	0	1	3
	20	0	0	0	1	0	1	0	1	3
	19	1	0	0	0	1	1	0	0	3
	18	0	1	0	0	1	1	0	0	3
	17	0	0	1	0	1	1	0	0	3
	16	0	0	0	1	1	1	0	0	3
	15	0	1	1	0	1	0	0	0	3

Table 109: ECC Code Matrix (Continued)

Check Bit	Data Bit	ECC Code Bits								Number of 1s in syndrome
		7	6	5	4	3	2	1	0	
	14	0	1	1	0	0	1	0	0	3
	13	0	1	1	0	0	0	1	0	3
	12	0	1	1	0	0	0	0	1	3
	11	1	1	1	1	1	0	0	0	5
	10	0	1	0	0	1	1	1	1	5
7		1	0	0	0	0	0	0	0	1
0		0	0	0	0	0	0	0	1	1
	9	0	1	1	1	0	0	0	0	3
	8	1	1	0	1	0	0	0	0	3
	7	0	0	0	0	0	1	1	1	3
	6	0	0	0	0	1	1	0	1	3
	5	1	1	1	1	0	0	0	1	5
	4	0	0	1	0	1	1	1	1	5
6		0	1	0	0	0	0	0	0	1
1		0	0	0	0	0	0	1	0	1
	3	1	0	0	0	0	0	1	1	3
	2	0	1	0	0	0	0	1	1	3
	1	0	0	1	0	0	0	1	1	3
	0	0	0	0	1	0	0	1	1	3

The MV64360/1/2 calculates ECC by taking the EVEN parity of ECC check codes of all data bits that are logic one. For example, if the 64 bit data is 0x45. The binary equivalent is 01000101. From Table 109, the required check codes are 00001101 (bit[6]), 01000011 (bit[2]) and 00010011 (bit[0]). Bitwise XOR of this check codes (even parity) result in ECC value of 01011101.

For error checking, MV64360/1/2 reads 64-bits of data and 8-bits of ECC. It calculates ECC based on the 64-bit data and then compares it against the received ECC. The result of this comparison (bitwise XOR between received ECC and calculated ECC) is called the syndrome.

If the syndrome is 00000000, both the received data and ECC are correct.

If the syndrome is any other value, the MV64360/1/2 assumes either the received data or the received ECC are in error.

If the syndrome contains a single '1', there is a single bit error in the ECC byte. For example, if the received data is 0x45, the calculated ECC is 01011101, as explained before. If the received ECC is 01010101, the resulting syndrome is 00001000. Table 109 on page 314 shows that this syndrome corresponds to check bit 3. The MV64360/1/2 does not report or correct this type of error.

If the syndrome contains three or five ‘1’s, it indicates that there is at least one data bit error. For example, if the received data is 0x45, the calculated ECC is 01011101, as explained before. If the received ECC is 00011110, the resulting syndrome is 01000011. This syndrome includes three ‘1’s and it corresponds to data bit 2 as shown in Table 109. In this case, the MV64360/1/2 corrects the data by inverting data bit 2 (the corrected data is 0x41).

If the result syndrome contains two ‘1’s, it indicates that there is a double-bit error.

If the result syndrome contains four ‘1’s, it indicates a 4-bit error located in four consecutive bits of a nibble.

If the result syndrome contains five ‘1’s, and no four of the ‘1’s are contained in check bits [7:4] or check bits [3:0] (which means it does not correspond to any data bit of the table), it indicates a triple-bit error within a nibble.

Note

These types of errors cannot be corrected. The MV64360/1/2 reports an error but will not change the data.

19.2.2 DDR SDRAM Interface Operation

On SDRAM reads, the MV64360/1/2 reads the ECC byte with the data, calculates the ECC byte, and compares it against the read ECC byte. In case of a single bit error, it corrects the error and drives the correct data to the initiating interface. In case of two errors detection (or 3 or 4 errors that resides in the same nibble), it only reports an error, see section 19.2.3.

On a write transaction, the MV64360/1/2 calculates the new ECC and writes it to the ECC bank, with the data that is written to the data bank. Since the ECC calculation is based on a 64-bit data width, if the write transaction is smaller than 64 bits, the MV64360/1/2 runs a read modify write (RMW) sequence. It reads the full 64-bit data, merges the incoming data with the read data, and writes the new data back to SDRAM bank with new ECC byte.

Note

If identifying a non-correctable error during the read portion of the RMW sequence, the MV64360/1/2 writes the data back to DRAM with a non-correctable ECC byte (it calculates a new ECC byte and than flips two bits). This behavior guarantees that the error is still visible if there is a future read from this DRAM location.

In case of burst write to DRAM, the MV64360/1/2 executes a RMW access of the whole burst, even if only part of the data requires RMW. When interfacing with the DDR SDRAM, performing a RMW access only for the required data is not efficient, in most cases, due to the overhead of bus turnaround cycles.

The MV64360/1/2 also supports forcing bad ECC written to the ECC bank for debug purposes. If this mode is enabled, rather than calculating the ECC to be written to the ECC bank, it drives a fixed ECC byte configured in SDRAM ECC Control register, see Table 306 on page 500.

SDRAM interface also contains a 32-bit ECC error counter that counts the number of corrected, single bit errors that are detected. Use software to reset the ECC error counter.

19.2.3 ECC Error Report

In case of ECC error detection, the MV64360/1/2 asserts an interrupt (if not masked), and latches the:

- Address in the ECC Error Address register.
- 64-bit read data in the ECC Error Data register.
- Read ECC byte in the SDRAM ECC register.
- Calculated ECC byte in the Calculated ECC register.

Note

For more information about these registers, see C.3 “SDRAM Error Report Registers” on page 498.

The MV64360/1/2 reports an ECC error whenever it detects but cannot correct an error (2, 3, or 4 bits errors).

The MV64360/1/2 also reports on single bit errors (correctable errors), based on the setting of the ECC threshold, bits [23:16], in the ECC Control register.

- If the threshold is set to ‘0’, there is no report on single bit errors.
- If set to ‘1’, MV64360/1/2 reports each single bit error.
- If set to ‘n’, MV64360/1/2 reports each ‘n’ single bit error.

Note

In case of multiple errors detection, the address, data, and ECC are latched in the corresponding registers only for the first error. Latching of new data into these registers is enabled only after reading ECC Error Address register, and clearing the interrupt (writing 0x0 to the Error Address register). The interrupt handler must read this register last.

19.3 Parity Support for Devices

The MV64360/1/2 supports parity on the device bus. Each of the five device chip selects, can be configured to support, or not support, parity.

The device bus contains four parity bits per the 32-bits of address and data (parity bit per byte). It can be configured to Even or Odd parity. Whenever it drives the bus (address or write data), it drives parity on the parity lines. During a read access to device, it calculates parity on the incoming data, and compares to the received parity.

In case of parity error detection, the MV64360/1/2 asserts an interrupt (if not masked), and latches the:

- Address in the Device Error Address register.
- 32-bit read data in the Device Error Data register.
- 4-bit read parity in the Device Error Parity register.

19.4 PCI/PCI-X Parity Support

The MV64360/1/2 implements all parity features required by the PCI and PCI-X specifications. This includes PAR, PAR64#, PERR#, and SERR# generation and checking.

As an initiator (requester or split completer), the MV64360/1/2 generates even parity on PAR signals for write transaction’s address, attribute, and data phases. It samples PAR on data phase of read transactions.

Note

If the MV64360/1/2 detects bad parity and the PCI Status and Command Configuration register’s `PErrEn` bit [6] (see Table 419 on page 555) is set, it asserts PERR#.

As a target, the MV64360/1/2 generates even parity on PAR signals for a read transaction’s data phase. It samples PAR on the address phase and data phase of write or split completion transactions.

In all of the parity errors conditions, the MV64360/1/2 generates an interrupt (if not masked) and latches the:

- Address in the PCI Error Address registers (Table 414 on page 554 and Table 415 on page 554)
- Attributes in the PCI Error Attribute register (Table 416 on page 554)
- Command, byte-enable, and parity in the PCI Error Command register (Table 417 on page 554)

If the PCI Status and Command configuration register's `SErrEn` bit [8] is set to '1' and enabled via the PCI SERR Mask register (see Table 411 on page 550), the MV64360/1/2 may also assert SERR#. If any of the parity errors conditions occurs, SERR# is asserted.

Note

In case of multiple errors detection, address, data and parity are latched in the corresponding registers only for the first error. Latching of new data into these registers is enabled only when reading PCI Error Address (Low) register. The interrupt handler must read this register last.

19.5 Integrated SRAM Parity Support

Note

Integrated SRAM parity support only applies to the MV64360 and MV64361 devices. There is no integrated SRAM in the MV64362

The MV64360/1/2 supports parity on its integrated 2 Mb SRAM.

The SRAM is 144-bit wide– 128-bit data + 16-bit parity (parity bit per byte). During a write access to SRAM, the MV64360/1/2 calculates even parity for the write data and drives it on the parity lines, along with the data. During a read access to the SRAM, the MV64360/1/2 calculates even parity on the incoming data and compares it to the received parity.

In case of parity error detection, the MV64360/1/2 asserts an interrupt (if not masked), and latches the:

- Address in the SRAM Error Address register.
- 64-bit read data in the SRAM Error Data Low and High registers.
- 8-bit read parity in the SRAM Error Parity register.

19.6 Parity/ECC Errors Propagation

Although each interface includes the required logic to detect and report parity/ECC errors, this is sometimes inadequate, due to the latency of interrupt routines.

For example, bad parity is detected on a PCI write to SDRAM. In the time required for the CPU interrupt handler to handle the interrupt, the bad data may be read by the CPU.

To guarantee this scenario does not occur, need to propagate the bad PCI parity to SDRAM as a non-correctable ECC error. This guarantees that once the CPU reads this data, it recognizes it as erroneous data.

In case of a write access to SDRAM with bad parity indication, the SDRAM interface can force two ECC errors to the ECC bank. If the SDRAM ECC Control register's `PErrProp` bit [9] is set to '1' (Table 306 on page 500), the MV64360/1/2 calculates the new ECC byte and flips two bits before writing it to the ECC bank.

In case of a write access to the device bus with bad parity indication, the Device controller can force bad parity. If the Device Interface Control register's `PErrProp` bit [19] is set to '1' (Table 320 on page 508), the MV64360/1/2 calculates the new parity and flips the bits before writing it to the device.

Similarly, if the SRAM Configuration register's `PErrProp` bit in the is set to '1'(Table 276 on page 484), the MV64360/1/2 forces bad SRAM parity in case of parity error indication, during a write access.

In case of a CPU read from SDRAM that results in ECC error detection (but no correction), or a CPU read from PCI that results in parity error, the MV64360/1/2 generates an interrupt. The CPU interface can be also configured to force bad parity in this case. If `PerrProp` bit in the CPU Configuration register is set to '1', the MV64360/1/2 calculates data parity and flips all the bits when driving it on the CPU bus.

In case of PCI reads from SDRAM that results in ECC error detection (but no correction), or in any case of CPU or IDMA write to PCI with bad ECC/parity indication, the PCI interface can force bad parity on the bus.
If the PCI Command register `PErrProp` bit is set to '1', the MV64360/1/2 calculates data parity and flips the value it drives on PAR.

Section 20. PowerPC Cache Coherency

The MV64360/1/2 supports cache coherency between the DDR DRAM and CPU caches and between the integrated SRAM and CPU caches.

Any access to the DRAM/integrated SRAM may result in a snoop transaction driven by the MV64360/1/2 on the CPU bus. In case of a HIT in a modified line in CPU cache, the DRAM/integrated SRAM access may be suspended until the line write-back to DRAM/integrated SRAM completes.

Notes

- The MV64360/1/2 also supports inter-CPUs cache coherency, when interfacing multiple CPUs, including MPX bus data intervention. For full details, look in 9.11 “Multi-CPU and Symmetric Multi Processing (SMP) Support” on page 115.
- Whenever this section use the term “main memory”, it refers to both the DDR SDRAM and the integrated SRAM.

20.1 Background

Cache coherency is required when cacheable regions in memory (SDRAM or integrated SRAM) are shared between the CPU and other interfaces, such as PCI or DMA.

For example, with a cacheable region in the SDRAM, shared between the CPU and some PCI agent, once the CPU caches data from the DRAM and modifies it, the data placed in the DRAM is no longer coherent with the data residing in the CPU cache. Now, if the PCI agent performs a read of the same data from DRAM, it is necessary to update the DRAM with the most updated data.

Cache coherency can be maintained through software means (semaphores). However, this requires delicate software design. Since the PowerPC CPU supports snoop mechanism, this can be used to force cache coherency by hardware.

Note

If an address parity error is detected during a snoop write back, cache coherency cannot be guaranteed. To enable cache coherency, the address windows defined by the address decode and remap registers in the PCI interface must exactly match the CPU interface address windows. For example, the size of the SCS_0 memory region must be the same in both the CPU and PCI interfaces. In addition, the remap registers in the PCI interface should map the SCS_0 region to exactly the same region defined by the SCS_0 address decode registers in the CPU interface.

The PowerPC snoops any transaction on the bus that is marked as global (GBL# signal is asserted). In case of a snoop hit in a modified cache line, it asserts ARTRY# indicating that the snoop transaction must be retried. In addition, it requests for bus ownership and pushes the line back to memory. It continues to respond with ARTRY# as long as the line has not yet been pushed out (even if it is in its write-back buffer). Once the modified line is written back to memory, the data in memory is again coherent, and can be used by other interfaces.

When running in MPX bus mode, and CPU data intervention is enabled, the CPU may respond with HIT# assertion rather than ARTRY# assertion, to a snoop hit in a modified line. In this case, the CPU only issues a data only transaction, pushing the cache line back to memory. This method is more efficient in terms of bus utilization. There is no need to waste bus cycles on transactions retry.

The PowerPC CPUs implement MEI, MESI, or MERSI cache protocol (CPU dependent). The MERSI states are:

Table 110: MERSI Cache States

State	Description
I	Invalid Cache line is not valid.
S	Shared Cache line is shared with other caches. Main memory data is coherent.
E	Exclusive Cache line is exclusive - only one cache has this valid copy of the line. Main memory data is coherent.
M	Modified Cache line has been modified. Only one cache has this valid copy of the line. Main memory data is NOT coherent.
R	Recent Cache line is shared with other caches and is the most recently read version of the data. Main memory data is coherent. **NOTE:** The cache line in the R state is the one used to supply shared data intervention.

If a cache line is in S, E, or R state, main memory data is coherent with cache data. There is no problem for any interface to access main memory. In case of a write, there is a need to notify the CPU that the cache data is not valid anymore.

If a cache line is in M state, need to confirm that the modified data is written back to memory before any other interface can access this data.

20.2 Snoop Regions

Since snoop action has a performance penalty (especially in the case of snoop hit in a modified cache line), snoops must be limited only to the address space which is cacheable, and shared between CPU and other interfaces. Each of the MV64360/1/2 units supports programmable DRAM address windows, in which cache coherency is maintained.

Note

There are no configurable cache coherent windows for the integrated SRAM. The SRAM as a one piece can be configured to be cache coherent or not.

Each window can be defined as WT or WB region. Mark a window as WT if it is guaranteed that any cache line within this window is never marked as M line in the cache. A write access to this region requires a snoop on the CPU bus, to invalidate CPU cache line. However, since it is guaranteed that main memory data is coherent with the cache data, there is no need to wait for a snoop response. Main memory access must be completed even before generating the snoop transaction on the CPU bus. If a window is marked as WB, the access to main memory is postponed until the snoop is resolved. In the worst case, this means until a snoop copy-back transaction is driven to main memory.

Notes

- Cache coherency cannot be guaranteed if an address parity error is detected during a snoop write back.
- To enable cache coherency, the address windows defined by the PCI's address decode and remap registers must exactly match those of the CPU interface. For example, the size of the SCS_0 memory region must be the same in the CPU and PCI interfaces. In addition, the remap registers in the PCI interface must map the SCS_0 region to the exact, same region defined by the SCS_0 address decode registers in the CPU interface.

20.3 Snoop Action

The MV64360/1/2 distinguishes between different accesses to cache coherency regions:

- Read from WT region
- Write to WT region
- Read from WB region
- Partial write to WB region
- Write of a full cache line to WB region

Note

Since snooping works on a cache line basis, any access to DDR SDRAM or integrated SRAM that requires snoop must not cross the cache line boundary. Each of the MV64360/1/2 interfaces that requires cache coherency support, must be configured to bursts that do not cross cache line boundary (32 bytes).

20.3.1 Read from a WT Region

A read from a WT region does not require a snoop action (cache data is guaranteed to be coherent with main memory data). It is treated as if there was no hit in any of the cache coherency windows.

20.3.2 Write to a WT Region

A write to a WT region only requires invalidation of the CPU cache line.

In this case, the write access to main memory is performed with a minimum performance penalty. In parallel, the MV64360/1/2 requests CPU bus ownership and generates an address only snoop transaction on the bus.

If the address hits a CPU cache line, the line is invalidated. No further action is required.

In case the CPU responds with ARTRY# to the snoop transaction, MV64360/1/2 retries the snoop transaction.

20.3.3 Read from a WB Region

A read from a WB region requires snooping the CPU bus and, in case of hit in a modified cache line, also a wait for the snoop copy-back to complete.

In this case, the read access to main memory is postponed. The MV64360/1/2 requests the CPU bus ownership and generates an addresses only snoop transaction on the bus. If the address hits an E, S or R state cache line, the line is invalidated or kept valid in the cache (CPU dependent). If it hits an M line, the CPU requests bus ownership and drives a copy-back of the line. In case the CPU responds with ARTRY# to the snoop transaction, the MV64360/1/2 retries the snoop transaction.

Once guaranteed that the cache line is no longer in the CPU cache, the MV64360/1/2 also snoops its own CPU interface write buffer. Only after confirming the line is not in the write buffer, the original read access to main memory can complete. This procedure guarantees that the data being read from main memory is the most updated data.

20.3.4 Partial Write to WB Region

A write that is not a full cache line to a WB region requires snooping the CPU bus and, in case of a hit in a modified cache line, also wait for the snoop copy-back to complete.

In this case, the write access to main memory is postponed. The MV64360/1/2 requests for CPU bus ownership and generates an address only snoop transaction on the bus. If the address hits an E, S, or R state cache line, the line is invalidated. If it hits an M line, the CPU requests bus ownership and drives a copy-back of the line. In case the CPU responds with ARTRY# to the snoop transaction, the MV64360/1/2 retries the snoop transaction.

Once it is guaranteed that the cache line is no longer in the CPU cache, the MV64360/1/2 also snoops its own CPU interface write buffer. Only after making sure the line is not in the write buffer, the original write access to main memory is completed. This procedure guarantees that the new incoming data is merged with the modified CPU cache line data. This means that the next time the data is being read from main memory, it is the most updated data.

20.3.5 Write of a Full Cache Line to WB region

A write of a full cache line to a WB region, requires snooping the CPU bus, to invalidate a modified CPU cache line in case of hit.

In this case, the write access to main memory is postponed. The MV64360/1/2 requests the CPU bus ownership, and generates an address only snoop transaction on the bus. If the address hits a CPU cache line, the line is invalidated. In case the CPU responds with ARTRY# to the snoop transaction, the MV64360/1/2 retries the snoop transaction.

Then, the MV64360/1/2 also snoops its own CPU interface write buffer (the line might be placed in the write buffer due to previous line copy-back, not as a result of the snoop transaction). Only after making sure the line is not in the write buffer, the original write access to main memory can complete. This procedure guarantees that the next time the data is being read from main memory it is the most updated one.

Note

The CPU might sometimes respond with ARTRY# to a snoop transaction, even if it does not have to push the cache line back to memory. This may happen due to some temporary resources conflicts within the CPU. The MV64360/1/2 keeps retrying the snoop transaction until there is no CPU ARTRY# response.

20.3.6 CPU Bus Snoop Transactions

Snoop transactions are always address only transactions. There are three address only transactions that can be used as snoop transactions - kill block, flush block and clean block. Different PowerPC CPUs respond to these snoop transactions differently in case of hit in a cache line, as shown in Table 111

Table 111: CPU Snoop Response

Snoop Cycle	CPU Type	Snoop Response
Kill Block	603e,604e,75x	• Hit M line: Line copy back and invalidated • Hit E or S line: Line is invalidated.
	MPC74xx	• Hit M line: Line copy back only if ARTRY# is asserted by other CPU (multi CPU configurations). Line is invalidated. • Hit E,S or R line: Line is invalidated.
Flush Block	604e,MPC74xx	• Hit M line: Line copy back and invalidated. • Hit E, S or R line: Line is invalidated.
	603e,75x	Not supported.
Clean Block	604e,MPC74xx	• Hit M line: Line copy back but kept in E state • Hit E, S or R line: None (line is kept in it's present state)
	603e,75x	Not supported.

All CPUs support Kill Block transactions. The MPC604e and MPC74xx also support Clean and Flush Block transactions. The supported address only transactions can be determined through CPU Master Configuration register. A summary of snoop transaction generated by MV64360/1/2 is shown in Table 112.

Table 112: MV64360/1/2 Snoop Transactions

SDRAM Access	Snoop Cycle
Write to WT region	Kill block
Read from WB region	Clean block if supported, else flush block if supported, else kill block
Partial Write to WB region	Flush block if supported, else kill block
Write a full cache line to WB region	Kill block

The MV64360/1/2 internal address path is 32-bit wide. However, it does support the MPC7450 extended address bus mode. In this mode, a snoop address driven by the MV64360/1/2 CPU master interface is 36-bit wide. The address is constructed of the original 32-bit address targeted to main memory, concatenated with 4-bit address derived from the appropriate CPU interface address decoding Base Address register. For example, an access to a cache coherent region in DRAM CS[0]#, results in 36-bit snoop address - address bits[31:0] (A[4-35] in Motorola's naming convention) are the original address bits, address bits[35:32] (A[0-3] in Motorola's naming convention) are taken from bits[15:12] of CPU interface SCS[0] Base Address register.

20.4 Implementation

The MV64360/1/2 cache coherency implementation involves multiple units.

- The requesting unit that tries to access a cache coherent region in memory
- The memory controller that generates a snoop request to the CPU interface
- The CPU interface that generates the address only snoop transaction on the CPU bus, and indicates to the memory controller when snoop is resolved.

In the case of a snoop miss, the latency of an access to a cache coherent region is higher than the access to a non-cache coherent region on the same device. Of course it is much higher in the case of snoop hit in a modified cache line. However, the MV64360/1/2 implementation is optimized to maintain full throughput, for the case of most snoops results in cache miss. Consecutive transaction to cache coherent region are pipelined, resulting in full throughput as in the case of access to non cache coherent region.

Notes

- MV64360/1/2 units that access cache coherent regions in the DRAM or integrated SRAM, must be configured to limit their transactions burst size to 4, thus preventing from a single transaction to cross cache line boundary. For example, if one of the PCI access windows is marked as a cache coherent region, it's MBurst parameter should be set to 0x0 (which represent maximum burst of 4).
- For best snoop performance, enable the snoop pipeline by setting the Dunit Control (High) register's `SnoopPipe` bit [24] to '1'.

20.4.1 DRAM Cache Coherency

The DRAM controller contains four transaction queues. A transaction targeted to a non-cache coherent region in DRAM is placed in one of the four queues and executed on the DRAM bus (see 11.1.2 "Cache Coherency" on page 132).

The DRAM controller also contains snoop queue. A transaction targeted to a cache coherent region is placed in the snoop queue, instead of a transaction queue. As soon as there are pending transactions in the snoop queue, the DRAM controller issues a snoop request to the CPU interface (a snoop request per each transaction).

A write transaction that is targeted to a WT region, is placed in both the snoop queue for snoop request (line invalidate) generation, and in one of the transaction queues for execution (there is no need to wait from snoop resolution).

A transaction that is targeted to a WB region must wait for snoop resolution before being moved to the transaction queues. However, the DRAM controller does not stall the snoop pipe until snoop resolution. It continues to issue snoop requests to the CPU interface as long as it has pending transaction in it's snoop queue. With each snoop resolution indication from the CPU interface, the DRAM controller moves a transaction from the snoop queue to the transaction queues.

As soon as it receives snoop request from the DRAM controller, the CPU master interface drives the appropriate address only snoop transaction on the bus. Its transaction queue enables absorbing new snoop requests, while driving previous requests on the bus. In the case of snoop miss, the CPU interface can issue a new address only snoop transaction every fourth cycle. In case a cache line copy back is required, the pipe is stalled until the write back is executed.

The CPU interface signals snoop resolution after the modified cache line has been copied back to memory. The CPU interface supports the conventional ARTRY# method of pushing a cache line and the MPX data intervention method. In the conventional method, the CPU master interface retries the snoop transaction until there is no longer CPU ARTRY# response. In the data intervention method, once the CPU responds with HIT# to a snoop transaction, the CPU master interface waits for the corresponding data only transaction to take place. For more details, look in 9.8 “PowerPC Cache Coherency” on page 107

20.4.2 Integrated SRAM Cache Coherency

The integrated SRAM cache coherency concept is similar to the DRAM controller logic.

If the SRAM is configured to be cache coherent, all transactions from crossbar requires snoop on the CPU bus, before being accessed to SRAM. These transactions are driven to the CPU interface master, to generate the snoop transaction. In parallel, they are placed in the transaction queue, waiting for snoop resolution. Once snoop is resolved, the transaction is pulled from the queue for execution.

Note

CPU transactions (which obviously do not require snoop action), bypass the postponed snoop transactions in the queue, and are driven to the SRAM.

Section 21. Timer/Counters

There are four 32-bit wide timer/counters on the MV64360/1/2. Each timer/counter can be selected to operate as a timer or as a counter.

Each timer/counter increments with every SysClk rising edge.

In Counter mode, the counter counts down to terminal count, stops, and issues an interrupt.

In Timer mode, the timer counts down, issues an interrupt on terminal count, reloads itself to the programmed value, and continues to count.

Reads from the counter or timer are done from the counter itself, while writes are to its register. This means that read results are in the counter's real time value.

Each timer/counter can be configured to have an external count enable input, through one of the MPP pins. In this configuration, the counter counts down as long as the count enable pin is active low.

Each timer/counter has a TCTcnt output pin. It is also muxed on the MPP pins. This pin is asserted when the counter reaches zero. The Timer/Counter Control register's `TCnt0_Width` bit [3] and `TCnt1_Width` bit [11] (see Table 616 on page 678) determine if TcTcnt is asserted for one or two SysClk cycles.

If a wider timer is required, cascade two timers to generate a 64-bit timer. Cascade the timers by connecting the first timer's TCTcnt output to the second timer's TCEn input. With this configuration, each time the first counter reaches terminal count the second counter decrements by one.

Note

If using an external count enable input, it is necessary to configure the appropriate MPP pin prior to counter activation.

TCTcnt is asserted one SysClk cycle after the counter reaches zero.

Section 22. Watchdog Timer

The MV64360/1/2 internal watchdog timer is a 32-bit count down counter that can be used to generate a non-maskable interrupt or reset the system in the event of unpredictable software behavior.

After the watchdog is enabled, it is a free running counter that needs to be serviced periodically in order to prevent its expiration.

Note

WDE# and WDNMI# are represented in the Main Interrupt Cause register (see Table 639 on page 689). They are also multiplexed on the MPP pins (see Table 116 on page 344), enabling dedicated WDNMI# and WDE# interrupts.

22.1 Watchdog Operation

After reset, the watchdog is disabled.

The watchdog must be serviced periodically in order to avoid NMI or reset (WDE#). Watchdog service is performed by writing ‘01’ to CTL2, followed by writing ‘10’ to CTL2. Upon watchdog service, the MV64360/1/2 clears the WDNMI and WDE bits (if set) and reloads the Preset_VAL into the watchdog counter.

A write sequence of ‘01’ followed by ‘10’ into CTL1 disables/enables the watchdog. The watchdog’s current status can be read in the Watchdog Configuration EN bit 31. When disabled, the MV64360/1/2 sets the WDNMI and WDE bits (if clear) and reloads the Preset_VAL into the watchdog counter.

Preset_VAL and NMI_VAL can be changed while the watchdog is enabled. However, Preset_VAL will affect the watchdog only after it is loaded into the watchdog counter (e.g. after watchdog service).

If the watchdog is not serviced before the counter reaches NMI_VAL, a non-maskable interrupt event occurs. The WDNMI bit is cleared, asserting low the WDNMI# pin.

To de-assert the WDNMI# and/or WDE# pins, the watchdog must be serviced, disabled, or the MV64360/1/2 device must be reset. The MV64360/1/2 holds WDE# asserted for the duration of 16 system cycles after reset assertion.

Section 23. Two-Wire Serial Interface (TWSI)

The MV64360/1/2 has full TWSI support. It can act as master generating read/write requests and as a slave responding to read/write requests. It fully supports multiple TWSI masters environment (clock synchronization, bus arbitration).

The TWSI interface can be used for various applications. It can be used to control other TWSI on board devices, to read DIMM SPD ROM and is also used for serial ROM initialization. For more details, see Section 27. “Reset Configuration” on page 347.

23.1 TWSI Bus Operation

The TWSI port consists of two open drain signals:

- SCL (Serial Clock)
- SDA (Serial address/data)

The TWSI master starts a transaction by driving a start condition followed by a 7- or 10-bit slave address and a read/write bit indication. The target TWSI slave responds with acknowledge.

In case of a write access (R/W bit is ‘0’, following the TWSI slave acknowledge, the master drives 8-bit data and the slave responds with acknowledge. This write access (8-bit data followed by acknowledge) continues until the TWSI master ends the transaction with stop condition.

In case of read access following the TWSI slave address acknowledge, the TWSI slave drives 8-bit data and the master responds with acknowledge. This read access (8-bit data followed by acknowledge) continues until the TWSI master ends the transaction by responding with no acknowledge to the last 8-bit data, followed by a stop condition.

A target slave that cannot drive valid read data right after it received the address, can insert “wait states” by forcing SCL low until it has valid data to drive on the SDA line.

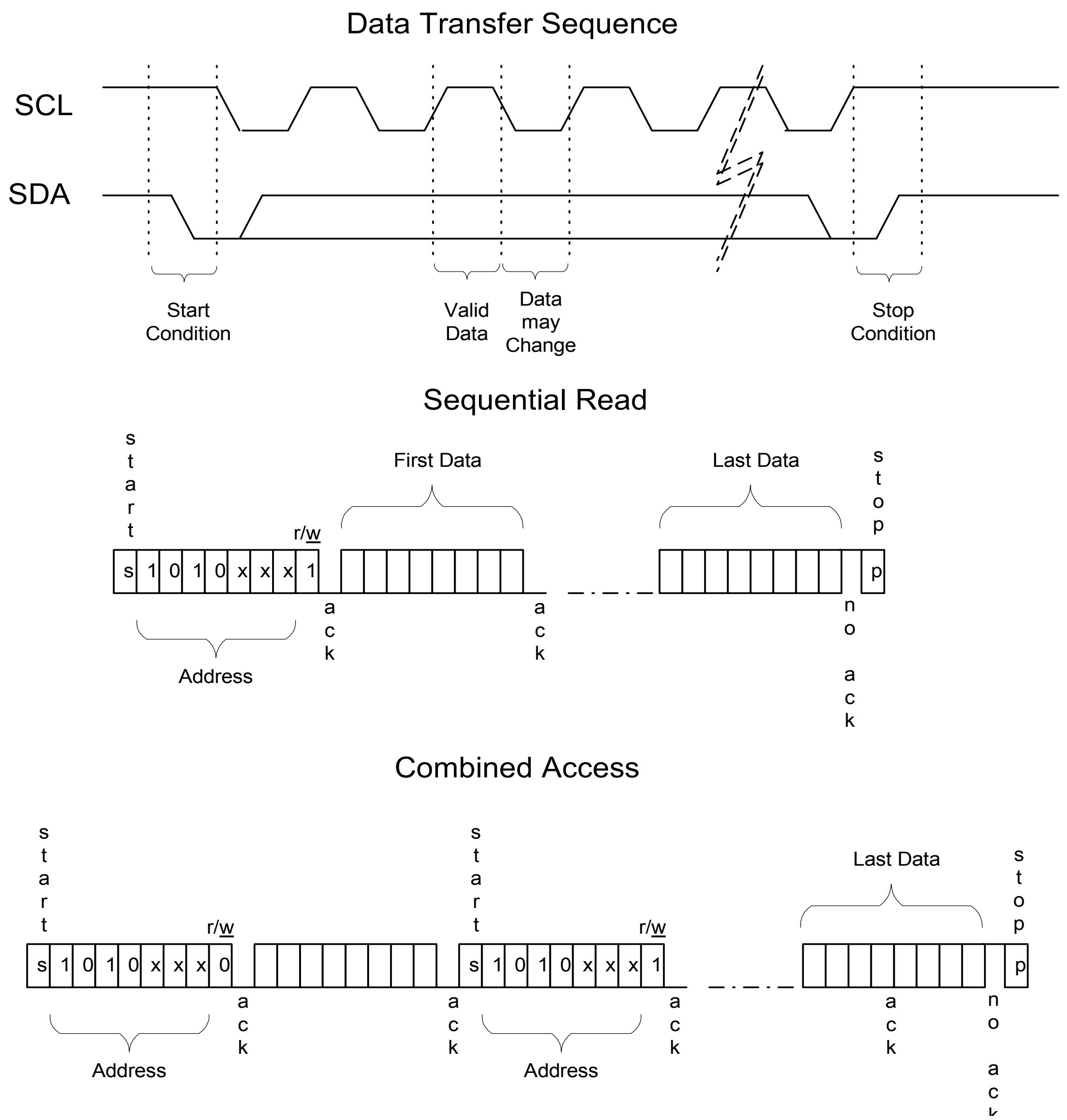
A master is allowed to combine two transactions. After the last data transfer, it can drive a new start condition followed by new slave address, rather than drive stop condition. Combining transactions guarantees that the master does not loose arbitration to some other TWSI master.

TWSI examples are shown in Figure 73.

Note

For full TWSI protocol description, see the TWSI specification.

Figure 73: TWSI Examples



23.2 TWSI Registers

The TWSI interface master and slave activities are handled by simple CPU (or PCI) access to internal registers, plus interrupt interface. The following sections describe each of these registers.

23.2.1 TWSI Slave Address registers

The TWSI slave interface supports both 7-bit and 10-bit addressing. The slave address is programmed by the Slave Address register (see Table 622 on page 683) and Extended Slave Address register (see Table 623 on page 683.

When the TWSI receives a 7-bit address after a start condition, it compares it against the value programed in the Slave Address register, and if it matches, it responds with acknowledge.

If the received 7 address bits are ‘11110xx’, meaning that it is an 10-bit slave address, the TWSI compares the received 10-bit address with the 10-bit value programed in the Slave Address and Extended Slave Address registers, and if it matches, it responds with acknowledge.

The TWSI interface also support slave response to general call transactions. If GCE bit in the Slave Address register is set to ‘1’, the TWSI also responds to general call address (0x0).

23.2.2 TWSI Data Register

The 8-bit Data register is used both in master and slave modes.

In master mode, the CPU must place the slave address or write data to be transmitted. In case of read access, it contains received data (need to be read by CPU).

In slave mode, the Data register contains data received from master on write access, or data to be transmitted (written by CPU) on read access.

Note

Data register MSB contains the first bit to be transmitted or being received.

23.2.3 TWSI Control Register

This 8-bit register contains the following bits:

Table 113: TWSI Control Register Bits

Bit	Function	Description
1:0	Reserved	Read only ‘0’.
2	Acknowledge Bit	When set to ‘1’, the TWSI drives an acknowledge bit on the bus in response to a received address (slave mode), or in response to a data received (read data in master mod, write data in slave mode). For a master to signal a TWSI target a read of last data, the CPU must clear this bit (generating no acknowledge bit on the bus). For the slave to respond, this bit must always be set back to 1.
3	Interrupt Flag	If any of the interrupt events occur, set to ‘1’ by TWSI hardware If set to ‘1’ and TWSI interrupts are enabled through bit[7], an interrupt is asserted.
4	Stop Bit	When set to ‘1’, the TWSI master initiates a stop condition on the bus. The bit is set only. It is cleared by TWSI hardware after a stop condition is driven on the bus.

Table 113: TWSI Control Register Bits (Continued)

Bit	Function	Description
5	Start Bit	When set to '1', the TWSI master initiates a start condition on the bus, when the bus is free, or a repeated start condition, if the master already drives the bus. The bit is set only. It is cleared by TWSI hardware after a start condition is driven on the bus.
6	TWSI Enable	If set to '1', the TWSI slave responds to calls to its slave address, and to general calls if enabled. If set to '0', SDA and SCL inputs are ignored. The TWSI slave does not respond to any address on the bus.
7	Interrupt Enable	If set to '1', TWSI interrupts are enabled. It is highly recommended to use TWSI interrupt to interface the TWSI module, rather than using register polling method.
31:8	Reserved	Reserved.

23.2.4 TWSI Status Register

This 8-bit register contains the current status of the TWSI interface. Bits[7:3] are the status code, bits[2:0] are Reserved (read only 0). Table 114 summarizes all possible status codes.

Table 114: TWSI Status Codes

Code	Status
0x00	Bus error.
0x08	Start condition transmitted.
0x10	Repeated start condition transmitted.
0x18	Address + write bit transmitted, acknowledge received.
0x20	Address + write bit transmitted, acknowledge not received.
0x28	Master transmitted data byte, acknowledge received.
0x30	Master transmitted data byte, acknowledge not received.
0x38	Master lost arbitration during address or data transfer.
0x40	Address + read bit transmitted, acknowledge received.
0x48	Address + read bit transmitted, acknowledge not received.
0x50	Master received read data, acknowledge transmitted.
0x58	Master received read data, acknowledge not transmitted.
0x60	Slave received slave address, acknowledge transmitted.
0x68	Master lost arbitration during address transmit, address is targeted to the slave (write access), acknowledge transmitted.
0x70	General call received, acknowledge transmitted.

Table 114: TWSI Status Codes (Continued)

Code	Status
0x78	Master lost arbitration during address transmit, general call address received, acknowledge transmitted.
0x80	Slave received write data after receiving slave address, acknowledge transmitted.
0x88	Slave received write data after receiving slave address, acknowledge not transmitted.
0x90	Slave received write data after receiving general call, acknowledge transmitted.
0x98	Slave received write data after receiving general call, acknowledge not transmitted.
0xA0	Slave received stop or repeated start condition.
0xA8	Slave received address + read bit, acknowledge transmitted.
0xB0	Master lost arbitration during address transmit, address is targeted to the slave (read access), acknowledge transmitted.
0xB8	Slave transmitted read data, acknowledge received.
0xC0	Slave transmitted read data, acknowledge not received.
0xC8	Slave transmitted last read byte, acknowledge received.
0xD0	Second address + write bit transmitted, acknowledge received.
0xD8	Second address + write bit transmitted, acknowledge not received.
0xE0	Second address + read bit transmitted, acknowledge received.
0xE8	Second address + read bit transmitted, acknowledge not received.
0xF8	No relevant status. Interrupt flag is kept 0.

23.2.5 Baud Rate Register

TWSI spec defines SCL frequency of 100 KHz (400 KHz in fast mode is not supported). The TWSI module contains a clock divider to generate the SCL clock.:

$$F_{SCL} = \frac{F_{SysClk}}{10 \cdot (M+1) \cdot 2^{(N+1)}}$$

Note

Where M is the value represented by bits[6:3] and N the value represented by bits[2:0]. If for example *M=N=4* (which are the default values), running *SysClk* at 100MHz results in *SCL* frequency of 62.5KHz.

As defined in the TWSI spec, the maximum supported SCL frequency is 100 KHz Fast mode (where SCL frequency is 400 KHz) is not supported.

23.3 TWSI Master Operation

The CPU can initiate TWSI master read and write transactions via TWSI registers, as described in the following sections.

23.3.1 Master Write Access

Master write access consists of the following steps:

1. The CPU sets the STA bit in the TWSI Control register to ‘1’. The TWSI master then generates a start condition as soon as the bus is free, sets an Interrupt flag, and sets the Status register to 0x8.
2. The CPU writes a 7-bit address, plus a write bit, to the Data register and the clears Interrupt flag for the TWSI master interface to drive the slave address on the bus. The target slave responds with acknowledge. This causes an Interrupt flag to be set and a status code of 0x18 is registered in the Status register.

   If the target TWSI device has an 10-bit address, the CPU needs to write the remainder 8-bit address bits to the Data register. The CPU then clears the Interrupt flag for the master to drive this address on the bus. The target device responds with acknowledge, causing an Interrupt flag to be set, and status code of 0xD0 be registered in the Status register.
3. The CPU writes data byte to the Data register, and then the clears Interrupt flag for the TWSI master interface to drive the data on the bus. The target slave responds with acknowledge, causing Interrupt flag to be set, and status code of 0x28 be registered in the Status register. The CPU continues this loop of writing new data to the Data register and clear Interrupt flag, as long as it needs to transmit write data to the target.
4. After the last data transmit, the CPU may terminate the transaction or restart a new transaction. To terminate the transaction, the CPU sets the Control register’s Stop bit and then clears the Interrupt flag, causing the TWSI master to generate a stop condition on the bus, and go back to idle state. To restart a new transaction, the CPU sets the TWSI Control register’s Start bit and clears the Interrupt flag, causing TWSI master to generate a new start condition.

Note

This sequence describes a normal operation. There are also abnormal cases, such as a slave not responding with acknowledge or arbitration loss. Each of these cases is reported in the Status register and needs to be handled by the CPU.

23.3.2 Master Read Access

1. Generating start condition, exactly the same as in the case of write access, see Section 23.3.1 Master Write Access.
2. Drive 7- or 10-bit slave address, exactly the same as in the case of write access, with the exception that the status code after the first address byte transmit is 0x40, and after 2nd address byte transmit (in case of 10-bit address) is 0xE0.
3. Read data being received from target device is placed in the data register and acknowledge is driven on the bus. Also interrupt flag is set, and status code of 0x50 is registered in the Status register. The CPU reads data from Data register and clears the Interrupt flag to continue receiving next read data byte. This look is continued as long as the CPU wishes to read data from the target device.
4. To terminate, the read access needs to respond with no acknowledge to the last data. It then generates a stop condition or generates a new start condition to restart a new transaction. With last data, the CPU clears the TWSI Control register’s Acknowledge bit (when clearing the Interrupt bit), causing the TWSI master inter-

face to respond with no acknowledge to last received read data. In this case, the Interrupt flag is set with status code of 0x58. Now, the CPU can issue a stop condition or a new start condition.

Note

The above sequence describes a normal operation. There are also abnormal cases, such as the slave not responding with acknowledge, or arbitration loss. Each of these cases is reported in the Status register and needs to be handled by CPU.

23.4 TWSI Slave Operation

The TWSI slave interface can respond to a read access, driving read data back to the master that initiated the transaction, or respond to write access, receiving write data from the master.

The two cases are described in the following sections.

23.4.1 Slave Read Access

Upon detecting a new address driven on the bus with read bit indication, the TWSI slave interface compares the address against the address programmed in the Slave Address register. If it matches, the slave responds with acknowledge. It also sets the Interrupt flag, and sets status code to 0xA8.

Note

If the TWSI slave address is 10-bit, the Interrupt flag is set and status code changes only after receiving and identify address match also on the 2nd address byte).

The CPU now must write new read data to the Data register and clears the Interrupt flag, causing TWSI slave interface to drive the data on the bus. The master responds with acknowledge causing an Interrupt flag to be set, and status code of 0xB8 to be registered in the Status register.

If the master does not respond with acknowledge, the Interrupt flag is set, status code 0f 0xC0 is registered, and TWSI slave interface returns back to idle state.

If the master generates a stop condition after driving an acknowledge bit, the TWSI slave interface returns back to idle state.

23.4.2 Slave Write Access

Upon detecting a new address driven on the bus with read bit indication, the TWSI slave interface compares the address against the address programed in the Slave Address register and, if it matches, responds with acknowledge. It also sets an Interrupt flag, and sets status code to 0x60 (0x70 in case of general call address, if general call is enabled).

Following each write byte received, the TWSI slave interface responds with acknowledge, sets an Interrupt flag, and sets status code to 0x80 (0x90 in case of general call access). The CPU then reads the received data from Data register and clears Interrupt flag, to allow transfer to continue.

If a stop condition or a start condition of a new access is detected after driving the acknowledge bit, an Interrupt flag is set and a status code of 0xA0 is registered.

Section 24. General Purpose Port

MV64360/1/2 contains a 32-bit General Purpose Port (GPP).

Each of the GPP pins can be assigned to act as a general purpose input or output pin and can be used to register external interrupts (when assigned as input pin). The GPP is multiplexed on the MV64360/1/2 MPP pins (see Section 26. “Pins Multiplexing” on page 343 section for more information).

24.1 GPP Control Registers

The MV64360/1/2 includes GPP I/O Control and GPP Level Control registers.

The I/O Control register determines the direction for each GPP pin. Setting a bit to ‘1’ configures the associated GPP pin to act as output pin. Setting a bit to ‘0’ configures the GPP pin as input pin.

The Level Control register determines the polarity for each GPP pin. Setting a bit to ‘1’ configures the associated GPP pin to be active low. Setting a bit to ‘0’ configures the GPP pin to be active high. The MV64360/1/2 negates an active low input pin before latching it inside. It inverts an active low output pin before driving it outside.

24.2 GPP Value Register

The MV64360/1/2 includes a 32-bit GPP Value register. Each GPP pin has an associated bit.

For pins configured as input pins, the associated bits are read only, and contains the value of the pins. When an input GPP pin is configured as asserted low, the value latched in GPP Value register is the negated value of the pin.

For pins configured as output pins, the associated bits are read/write. The value written to the GPP Value register bits is driven on the associated GPP output pins (inverted in case of active low pin).

24.3 GPP Interrupts

The GPP input pins can be used to register external interrupts. The MV64360/1/2 supports both edge sensitive and level sensitive interrupts.

If the Cunit Arbiter Control register’s `GPPInt` bit [10] is set to ’0’ (Table 564 on page 639), the external interrupts are treated as edge trigger interrupts. An assertion of a GPP input pin (toggle from ‘0’ to ‘1’ in case of active high pin, from ’1’ to ’0’ in case of active low pin), results in setting the corresponding bit in GPP Interrupt Cause register.

Note

The GPP pin must be kept active for at least one SysClk cycle to guarantee that the interrupt is registered.

If not masked by the GPP Interrupt Mask register, the GPP interrupt may cause a CPU or PCI interrupt. If a mask bit is set to ‘1’, interrupt is enabled. A mask register setting has no affect on registering GPP interrupts into the GPP Interrupt Cause register.

Interrupt is de-asserted as soon as software clears the corresponding bit in the GPP Interrupt cause register (write '0').

If the Cunit Arbiter Control register's `GPPInt` bit [10] is set to '1', the external interrupts are treated as level interrupts. In this mode, an interrupt is always generated when one of the GPP Value register bits is asserted and it is not masked by the GPP Interrupt Mask register (GPP Interrupt Cause register is not used for the interrupt generation).

Note

In this mode, interrupt handler clears the interrupt directly on originator device.

24.4 SMP Support

In SMP applications, there might be cases where each of the two CPUs requires to handle a different set of GPP interrupts. For these cases, the MV64360/1/2 implements two GPP Mask registers per the two CPUs. Each CPU can choose via it's mask register, which GPP interrupts to handle. More over, CPUs can choose to switch tasks during operation, and change their mask registers.

The GPPs as outputs, can be used by the CPUs to control external devices. In SMP application, there might be cases where each of the two CPUs requires to handle a different set of GPP outputs (interfacing different external devices). The GPP Value register is not good enough for this case. Lets take for example the case of CPU0 handles GPP[0] and CPU1 handles GPP[5]. Now if CPU0 sets bit[0] of GPP Value register, and now CPU1 wants to set bit[5], it must perform Read-Modify-Write in order not to clear bit[0], and even so, there is still a race condition, where CPU0 might want to change bit[0] at the same time.

For one CPU not to interfere with the other CPU, two additional registers are implemented - GPP Value Set and GPP Value Clear registers. Write a value of 1 to this register bit, sets the corresponding bit in the GPP Value register. Write a value of 0 has no affect. Similarly, write a value of 1 to a GPP Value Clear register bit, clears the corresponding bit in the GPP Value register. Writing a value of 0 has no affect. So, for the above example, in order for CPU1 to set bit[5], it writes a value of 0x20 to the GPP Value Set register. This guarantees that only bit[5] of the GPP Value register is set, and bit[0] is not affected.

Section 25. Interrupt Controller

The MV64360/1/2 includes an interrupt controller that routes internal interrupt requests (and optionally external interrupt requests) to both the CPU and the PCI bus.

The MV64360 and MV64361 can drive up to four interrupt pins; two open-drain interrupt pins (INT0# and INT1#) for the two PCI interfaces and two CPU interrupts (CPUInt[1:0]#).

The MV64362 can drive up to three interrupt pins; one open-drain interrupt pins (INT0#) for the two PCI interfaces and two CPU interrupts (CPUInt[1:0]#).

Notes

- The two open drain interrupt pins are not necessarily PCI interrupts. They can also be used as CPU interrupts. The only difference from CPUInt[1:0]# pins, is that a pullup is required.
- The MV64360/1/2 PCI interface(s) also supports MSI (Message Signaling Interrupt) as an alternative way to generate PCI interrupts. If MSI is enabled, rather than asserting INT0# or INT1# pins, the MV64360/1/2 PCI master generates a write transaction on the PCI. For more information, see 13.16.3 “Message Signaled Interrupt (MSI)” on page 182.

All interrupts driven by the MV64360/1/2 are level sensitive. The interrupt is kept active as long there is at least one non-masked cause bit set in the Interrupt Cause register.

Notes

- The MV64360/1/2 interrupt controller registers are implemented as part of the CPU interface unit, in order to have minimum read latency from CPU interrupt handler. This is not backward compatible with the GT-64240/60 implementation (meaning, the registers are placed in different offsets).
- The Main Interrupt Cause register is not backwards compatible with the GT-64240/60 implementation (some of the bits have different location within the registers).

The MV64360/1/2 can also be used as the interrupt controller for external devices generating interrupts to the CPU. It’s MPP pins can be configured as GPP interrupts, registering external level or edge sensitive interrupts.

25.1 Interrupt Cause and Mask Registers

The MV64360/1/2 handles interrupts in two stages. It includes a Main Cause register that summarizes the interrupts generated by each unit, and specific unit cause registers, that distinguish between each specific interrupt event.

25.1.1 Interrupts Cause Registers

Note

The following list of cause registers applies to all of the devices except where noted.

The MV64360/1/2 units cause registers are:

- CPU Cause register
- CPU Doorbell register
- DRAM Cause register
- Device Interface Cause register
- Integrated SRAM Cause register (MV64360 and MV64361)
- PCI0 Cause register
- PCI0 Inbound Cause register
- PCI0 Outbound Cause register
- PCI1 Cause register (MV64360 and MV64361)
- PCI1 Inbound Cause register (MV64360 and MV64361)
- PCI1 Outbound Cause register (MV64360 and MV64361)
- Gb Ethernet Unit Cause register
- Gb Ethernet0 Cause register
- Gb Ethernet1 Cause register (MV64360 and MV64361)
- Gb Ethernet2 Cause register (MV64360)
- IDMA Cause register
- Timers Cause register
- Comm Unit Cause register
- MPSC0 Cause register
- MPSC1 Cause register
- SDMA Cause register
- BRG Cause register
- GPP Cause register
- TWSI Cause register

Each unit has its own cause and mask register. Once an interrupt event occurs, its corresponding bit in the cause register is set to ‘1’. If the interrupt is not masked, it is also marked in the main interrupt cause register.

Note

The unit local mask register has no affect on the setting of interrupt bits in the Local Cause register. It only affects the setting of the interrupt bit in the Main Interrupt Cause register.

For example, if the CPU attempts to write to a write protected region, the CPU Error Cause register’s `WrErr` bit [4] (see Table 273 on page 481) is set to ‘1’. If the interrupt is not masked by CPU Mask register, the CPU bit in the Main Interrupt Cause register is also set. The interrupt handler first reads the Main Cause register and identifies that some CPU error event occurred. Then, it reads the CPU Error Cause register and identifies the exact cause for the interrupt.

Note

The Main Interrupt Cause register bits are Read Only. To clear an interrupt cause, the software needs to clear (write 0) the active bit(s) in the local cause register.

25.1.2 Interrupts Mask Registers

There is one mask register corresponding to each interrupt pins Setting these registers allows reporting different interrupt events on different interrupt pins. If a bit in the mask register is set to '1', the corresponding interrupt event is enabled. The setting of the mask bits has no affect on the value registered in the Interrupt Cause register, it only affects the assertion of the interrupt pin. An interrupt is asserted if at least one of the non masked bits in the cause register is set to 1.

The Main Interrupt Cause register is built of two 32-bit registers - Low and High. Each interrupts (INT0#, INT1#, CPUInt[1:0]#) also have two 32-bit mask registers, each.

Notes

- The INT1# interrupt only applies to the MV64360 and MV64361 devices.
- The Main Cause and Mask registers are physically placed in different units than the Local Cause and Mask registers. This means that one cannot guarantee write ordering between Main Mask registers and Local Cause registers. If such ordering is required (for example, clear cause bit in the local cause register, and then cancel mask in the main mask register), the first write must be followed with a read (that guarantees that the register programing is done) and only then programs the second register.

25.1.3 Selected Cause Registers

If any of the interrupt pins are asserted, for the interrupt handler to identify the exact interrupt, it must read both the Low and High Interrupt Cause registers. To minimize this procedure to a single read, the MV64360/1/2 contains Selected Cause registers. The interrupt handler can read these registers rather than the cause registers.

A Select Cause register is a shadow register of the Low or High Cause register, depending whether the active interrupt bit is in the Low or High Cause register. Bit[30] of the Select Cause register, indicates which of Low or High Cause registers are currently represented by the Select Cause register.

Lets take for example two interrupt events - CPU doorbell interrupt which is located in the High Cause register and DRAM ECC error which is located in the Low Cause register. If a CPU doorbell interrupt event occurs, a read of the Select Cause register will result with the value of the High Cause register, with Select bit[30] set to 1. If a DRAM ECC error occurs, a read of the Select Cause register will result with the value of the Low Cause register, with Select bit[30] set to 0. If both events happen simultaneously, a read of the Select Cause register will result with the value of the Low Cause register, Select bit[30] set to 0, and Stat bit[31] set to 1, indicating that there is also a pending interrupt in the High Cause register. In this case, the interrupt handler will need to read also the High Cause register.

25.1.4 Error Report Registers

The MV64360/1/2 also implements, on each of its interfaces, an Error Report registers that latches the address (and sometimes data, command, byte enables) upon an interrupt assertion caused by an error condition (such as parity error or address miss match). These registers can be helpful for the interrupt handler to locate the exact failure.

Note

For full details, see the registers section of each interface.

Section 26. Pins Multiplexing

26.1 Multiplexing for Ethernet Port2

Note

The following information on Ethernet Port2 only applies to the MV64360.

The MV64360 contains three GbE MACs. Ethernet Port0 and Ethernet Port1 have dedicated pins, see Section 3. "Pin Information" on page 41. To enable the Ethernet Port2, set DevAD[25] to '1', see Table 117 on page 348.

Ethernet Port2 pins are multiplexed on the upper pins of the PCI_1 interface. The port multiplexing for Ethernet Port2 is shown in Table 115.

Table 115: GbE Port2 Multiplexing

GE Port Pin	PCI Pin	GE Port Pin	PCI Pin
TxD2[7]	PAD1[63]	RxErr2	PAD1[50]
TxD2[6]	PAD1[62]	RxClk2	PAD1[49]
TxD2[5]	PAD1[61]	RxDv2	PAD1[48]
TxD2[4]	PAD1[60]	RxD2[0]	PAD1[47]
TxD2[3]	PAD1[59]	RxD2[1]	PAD1[46]
TxD2[2]	PAD1[58]	RxD2[2]	PAD1[45]
TxD2[1]	PAD1[57]	RxD2[3]	PAD1[44]
TxD2[0]	PAD1[56]	RxD2[4]	PAD1[43]
TxClkOut2	PAD1[55]	RxD2[5]	PAD1[42]
TxEn2	PAD1[54]	RxD2[6]	PAD1[41]
TxErr2	PAD1[53]	RxD2[7]	PAD1[40]
TxClk2	PAD1[52]	COL2	PAD1[39]
CRS2	PAD1[51]		

26.2 Multi Purpose Pins

The MV64360/1/2 contains 32 Multi Purpose Pins. Each pin can be assigned to a different functionality through the MPP Control registers, see Appendix P. "Pins Multiplexing Interface Registers" on page 696. The MPP pins can be used as hardware control signals to the MV64360/1/2 interfaces, or as a General Purpose In/Out Port (GPP). When the MPP pins are configured as GPPs, the GPP control and value registers must also be programmed, see Section 24. "General Purpose Port" on page 338.

Note

Unlike GT-64240/60, that has dedicated pins for its MPSCs, in the MV64360/1/2, the MPSCs are multiplexed on the MPP pins.

Table 116 shows each MPP pins' functionality as determined by the MPP Multiplex register.

Table 116: MPP Function Summary

MPP[0]	GPP[0]	MV64360/ MV64361: GNT1[0]# MV64362: Reserved	TxD0	MV64360/ MV64361: CD1 MV64362: Reserved	DBurst#	Debug[0]
MPP[1]	GPP[1]	MV64360/ MV64361: REQ1[0]# MV64362: Reserved	RxD0	MV64360/ MV64361: SCLK1 MV64362: Reserved	MV64360/ MV64361: PME1# MV64362: Reserved	Debug[1]
MPP[2]	GPP[2]	MV64360/ MV64361: GNT1[1]# MV64362: Reserved	RTS0	MV64360/ MV64361: TSCLK1 MV64362: Reserved	DBurst#	Debug[2]
MPP[3]	GPP[3]	MV64360/ MV64361: REQ1[1]# MV64362: Reserved	CTS0	MV64360/ MV64361: TxD1 MV64362: Reserved	InitAct	Debug[3]
MPP[4]	GPP[4]	MV64360/ MV64361: GNT1[2]# MV64362: Reserved	CD0	MV64360/ MV64361: TxD1 MV64362: Reserved	DBurst#	Debug[4]
MPP[5]	GPP[5]	MV64360/ MV64361: REQ1[2]# MV64362: Reserved	SCLK0	MV64360/ MV64361: RxD1 MV64362: Reserved	MV64360/ MV64361: PME1# MV64362: Reserved	Debug[5]
MPP[6]	GPP[6]	MV64360/ MV64361: GNT1[3]# MV64362: Reserved	TSCLK0	MV64360/ MV64361: RTS1 MV64362: Reserved	DBurst#	Debug[6]
MPP[7]	GPP[7]	MV64360/ MV64361: REQ1[3]# MV64362: Reserved	TxD0	MV64360/ MV64361: CTS1 MV64362: Reserved	InitAct	Debug[7]

Table 116: MPP Function Summary (Continued)

MPP[8]	GPP[8]	MV64360/ MV64361: GNT1[4]# MV64362: Reserved	TxD0	MV64360/ MV64361: CD1 MV64362: Reserved	DBurst#	Debug[8]
MPP[9]	GPP[9]	MV64360/ MV64361: REQ1[4]# MV64362: Reserved	RxD0	MV64360/ MV64361: SCLK1 MV64362: Reserved	MV64360/ MV64361: PME1# MV64362: Reserved	Debug[9]
MPP[10]	GPP[10]	MV64360/ MV64361: GNT1[5]# MV64362: Reserved	RTS0	MV64360/ MV64361: TSCLK1 MV64362: Reserved	DBurst#	Debug[10]
MPP[11]	GPP[11]	MV64360/ MV64361: REQ1[5]# MV64362: Reserved	CTS0	MV64360/ MV64361: RxD1 MV64362: Reserved	InitAct	Debug[11]
MPP[12]	GPP[12]	MV64360/ MV64361: GNT1[0]# MV64362: Reserved	CD0	MV64360/ MV64361: TxD1 MV64362: Reserved	DBurst#	Debug[12]
MPP[13]	GPP[13]	MV64360/ MV64361: REQ1[0]# MV64362: Reserved	SCLK0	MV64360/ MV64361: RTS1 MV64362: Reserved	MV64360/ MV64361: PME1# MV64362: Reserved	Debug[13]
MPP[14]	GPP[14]	MV64360/ MV64361: GNT1[1]# MV64362: Reserved	TSCLK0	MV64360/ MV64361: RxD1 MV64362: Reserved	DBurst#	Debug[14]
MPP[15]	GPP[15]	MV64360/ MV64361: GNT1[1]# MV64362: Reserved	RxD0	MV64360/ MV64361: CTS1 MV64362: Reserved	InitAct	Debug[15]
MPP[16]	GPP[16]	GNT0[0]#	DMAReq[0]#	InitAct	WDNMI#	Debug[16]
MPP[17]	GPP[17]	REQ0[0]#	DMAAck[0]#	EOT[3]#	WDE#	Debug[17]
MPP[18]	GPP[18]	GNT0[1]#	DMAReq[1]#	PME0#	WDNMI#	Debug[18]
MPP[19]	GPP[19]	REQ0[1]#	DMAAck[1]#	EOT[2]#	WDE#	Debug[19]

Table 116: MPP Function Summary (Continued)

MPP[20]	GPP[20]	GNT0[2]#	DMAReq[2]#	InitAct	BClkIn	Debug[20]
MPP[21]	GPP[21]	REQ0[2]#	DMAAck[2]#	EOT[1]#	BClkIn	Debug[21]
MPP[22]	GPP[22]	GNT0[3]#	DMAReq[3]#	PME0#	BClkOut	Debug[22]
MPP[23]	GPP[23]	REQ0[3]#	DMAAck[3]#	EOT[0]#	BClkOut	Debug[23]
MPP[24]	GPP[24]	GNT0[4]#	S[0]#	InitAct	WDNMI#	Debug[24]
MPP[25]	GPP[25]	REQ0[4]#	TCTcnt[0]#	EOT[3]#	WDE#	Debug[25]
MPP[26]	GPP[26]	GNT0[5]#	TCEn[1]#	PME0#	WDNMI#	Debug[26]
MPP[27]	GPP[27]	REQ0[5]#	TCTcnt[1]#	EOT[2]#	WDE#	Debug[27]
MPP[28]	GPP[28]	GNT0[0]#	TCEn[2]#	InitAct	BClkIn	Debug[28]
MPP[29]	GPP[29]	REQ0[0]#	TCTcnt[2]#	EOT[1]#	BClkIn	Debug[29]
MPP[30]	GPP[30]	GNT0[1]#	TCEn[3]#	PME0#	BClkOut	Debug[30]
MPP[31]	GPP[31]	REQ0[1]#	TCTcnt[3]#	EOT[0]#	BClkOut	Debug[31]

Note

Since each pin might act as output or input pin, depending on its configured functionality, all MPP pins wake up after reset as GPP input pins.

26.3 MPP I/O Pads

The 32 MPPs are 5V tolerant. This is useful, when interfacing 5V legacy devices. For example, if the MPPs are used as a PCI bus arbiter REQ#/GNT# signals, and interfacing 5V PCI master. Another example would be, when used as GPP input, registering external 5V interrupt inputs.

Note

The MPP pads have integrated clamping diodes as the PCI pads that clamp the input voltage to VREF. MPP[15:0] are clamped to VREF1 and MPP[31:16] to VREF0. If VREF is 3.3V, the corresponding MPPs must not be used with 5V signaling. Also, if not using PCI at all, the VREF pins can be left unconnected, in order to eliminate the MPPs clamping.

The MPPs pads have configurable output impedance. The MPPs are divided into two groups – MPP[15:0] and MPP[31:16]. Each group shares the same pad calibration logic. After reset:

- The MPP[15:0] pads output buffers are automatically tuned according to the impedance defined by an external resistor attached to pin PCI1_CAL (same tuning as the PCI1 pads).
- The MPP[31:16] pads drive is tuned according to the impedance defined by an external resistor attached to pin PCI0_CAL (same tuning as PCI0 pads).

Later, the MPPs drive strength can be changed via the PCI/MPP Pads Drive Control register (Table 377 on page 535).

Section 27. Reset Configuration

27.1 Reset Pins

The MV64360/1/2 supports three reset pins:

- SysRst#: The main reset pin.
- RST0# and RST1#: The PCI interface reset pins.

Separating SysRst# from the PCI reset pins is typically required in Hot Swap configurations, where you want the CPU to boot and start to initialize the board before the PCI slot reset signal is de-asserted. Separating the two PCI interface resets is required for PCI compliance reasons (since the two PCI busses are independent, each one must have its own reset).

SysRst# is the main MV64360/1/2 reset pin. When asserted, all MV64360/1/2 logics are in reset state and all outputs are floated, except for DRAM address and control outputs (see 11.8 “SDRAM Initialization” on page 140).

The PCI reset pins are independent. The PCI interface is kept in its reset state as long as its corresponding reset pin is asserted. On reset de-assertion, all PCI configuration registers are set to their initial values as specified in the PCI spec.

Notes

- The PCI reset pins may be de-asserted at or after SysRst# de-assertion.
- SysRst# MUST be asserted AT or AFTER PCI reset assertion.

Since the MV64360/1/2 supports SysRst# de-assertion prior to the PCI reset pins de-assertion, the CPU software might need a hook to recognize when the PCI bus is alive. Use the PCI Mode register’s `PRst` bit [31] of each PCI interface for this purpose, see Table 379 on page 538. Upon PCI reset de-assertion, the bit is set to ‘1’.

27.2 Pins Sample Configuration

The MV64360/1/2 must acquire some knowledge about the system before it is configured by the software. Special modes of operation are sampled on RESET to enable the MV64360/1/2 to function as required.

The following configuration pins are sampled during SysRst# assertion. These signals must be kept pulled up or down until SysRst# de-assertion (zero Hold time in respect to SysRst# de-assertion).

Note

If external logic is used instead of pull up and pull down resistors, the logic must drive the AD[31:0] signals to the desired values during Rst* assertion. The external logic must float the bus no later than the third cycle after Rst* de-assertion until normal operation occurs.

Table 117: Reset Configuration

Pin	Configuration Function
DevAD[0]	Serial ROM initialization
	0 = Disabled 1 = Enabled
DevAD[1]	DRAM Pads Calibration
	0 = Disable 1 = Enable
DevAD[3:2]	Serial ROM Address
	00 = ROM address is 1010000 01 = ROM address is 1010001 10 = ROM address is 1010010 11 = ROM address is 1010011 **NOTE:** If not using serial ROM initialization (DevAD[0] sampled LOW), DevAD[3:2] can remain with no pullup/pulldown.
DevAD[4]	Internal 60x bus Arbiter
	0 = Disable 1 = Enable
DevAD[5]	Default Internal Space Base
	0 = 0x1400.0000 1 = 0xF100.0000
DevAD[7:6]	CPU Bus Configuration
	00 = 60x bus 01 = MPX bus 10 = Reserved 11 = Reserved
DevAD[8]	CPU Pads Calibration
	0 = Disable 1 = Enable
DevAD[9]	Multiple MV64360/1/2 Support
	0 = Disable 1 = Enable

Table 117: Reset Configuration (Continued)

Pin	Configuration Function
DevAD[11:10]	Multi-MV64360/1/2 Address ID
	00 = MV64360/1/2 responds to A[9-10] ='00' 01 = MV64360/1/2 responds to A[9-10] ='01' 10 = MV64360/1/2 responds to A[9-10] ='10' 11 = MV64360/1/2 responds to A[9-10] ='11' **NOTE:** A[9-10] refers to the MV64360/1/2 signals. If not using multi-MV mode (DevAD[9] sampled LOW), DevAD[11:10] can remain with no pullup/pulldown.
DevAD[12]	PCI_0 Pads Calibration
	0 = Disable 1 = Enable
DevAD[13]	PCI_1 Pads Calibration **NOTE:** In the MV64362, pull-down to VSS
	0 = Disable 1 = Enable
DevAD[15:14]	BootCS# Device Width
	00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = Reserved
DevAD[16]	PCI Retry
	0 = Disable 1 = Enable
DevAD[17]	
	Must pull up.
DevAD[18]	DDR SDRAM Clock Select
	0 = DRAM is running at a higher frequency than the core clock 1 = DRAM is running with core clock **NOTE:** If set to '1', the DRAM PLL is not used.
DevAD[19]	DDR SDRAM Address/Control Delay
	0 = DRAM address/control signals toggle on falling edge of DRAM clock 1 = DRAM address/control signals toggle on rising edge of DRAM clock **NOTE:** The DevAD[19] setting depends on board topology and DRAM load. Detailed recommendations will follow silicon testing.

Table 117: Reset Configuration (Continued)

Pin	Configuration Function
DevAD[21:20]	DDR SDRAM Control Path Pipeline Select
	00 = Reserved 01 = Two pipe stages. (Up to 133 MHz) 10 = Reserved 11 = Three pipe stages. (Up to 183 MHz) **NOTE:** For DRAM frequency up to 133MHz, set to ’01’. This is the recommended setting when running the DRAM with the core clock. For DRAM frequency up to 183 MHz, set to ’11’.
DevAD[22]	DDR SDRAM Read Data Synchronization Select
	0 = Read data is synchronized to core clock 1 = Read data is synchronized to FBClkIn **NOTE:** The DevAD[24:22] setting depends on board topology and DRAM load. Detailed recommendations will follow silicon testing.
DevAD[23]	DDR SDRAM Read Control Logic Delay
	0 = Disabled 1 = Enabled The clock to the read control logic is delayed (using DFCDL). **NOTE:** The DevAD[24:22] setting depends on board topology and DRAM load. Detailed recommendations will follow silicon testing.
DevAD[24]	DDR SDRAM Read Data Delay
	0 = Disabled 1 = Enabled The read data sample clock is delayed (using DFCDL). **NOTE:** The DevAD[24:22] setting depends on board topology and DRAM load. Detailed recommendations will follow silicon testing.
DevAD[25]	GbE Port3 Enable **NOTE:** In the MV64361 and MV64362, must be pull-down to VSS
	0 = Disable 1 = Enable. **NOTE:** The third GbE port is multiplexed on the upper bits of PCI_1 bus (see Section 26. “Pins Multiplexing” on page 343).
DevAD[28:26]	PCI_1 DLL Control **NOTE:** In the MV64362, must be pull-down to ’000’.
	Same as PCI_0 DLL Control

Table 117: Reset Configuration (Continued)

Pin	Configuration Function
DevAD[31:29]	PCI_0 DLL control
	000 = DLL disable 001 = Conventional PCI mode at 66MHz 101 = PCI-X mode at 133MHz 110 = PCI-X mode at 66MHz **NOTE:** If interfacing conventional 33MHz PCI (M66EN sampled LOW), DLL is disabled, and DevAD[28:26] can be left with no pull up/down. DevAD[28:26] setting is preliminary and might be changed after silicon testing.
TxD0[0]	GbE port0 GMII/TBI Select
	0 = MII/GMII 1 = TBI
TxD1[0]	GbE port1 GMII/TBI Select **NOTE:** Only applies to the MV64360 and MV64361 devices.
	0 = MII/GMII 1 = TBI
TxD2[0]	GbE port2 GMII/TBI Select **NOTE:** Only applies to the MV64360 devices.
	0 = MII/GMII 1 = TBI **NOTE:** TxD2[0] is muxed on PCI_1 PAD1[56] pin. If not using Ethernet Port3, do not pull up/down this pin.
DevWE[3:0]	DRAM PLL N Divider [7:4]
	See 11.11 "DRAM Clocking" on page 144 on N[7:0] and M[5:0] settings. **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), DRAM PLL is not used. There is no need for pull up/down on DevWE[3:0], DevDP[3:0], BAdr[2:0] and TxD0[6:1].
DevDP[3:0]	DRAM PLL N Divider [3:0]
	See 11.11 "DRAM Clocking" on page 144 on N[7:0] and M[5:0] settings. **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), DRAM PLL is not used. There is no need for pull up/down on DevWr[3:0], DevDP[3:0], BAdr[2:0] and TxD0[6:1].
BAdr[0]	DRAM PLL NP
	Pull up NP **NOTE:** May be changed following silicon testing.

Table 117: Reset Configuration (Continued)

Pin	Configuration Function
BAdr[1]	DRAM PLL HIKVCO
	Pull down HIKVCO **NOTE:** May be changed following silicon testing.
BAdr[2]	DRAM PLL PU
	0 = PLL power down 1 = PLL power up (normal operation)
TxD0[6:1]	DRAM PLL M Divider
	See 11.11 “DRAM Clocking” on page 144 on N[7:0] and M[5:0] settings. **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), DRAM PLL is not used. There is no need for pull up/down on DevWr[3:0], DevDP[3:0], BAdr[2:0] and TxD0[6:1].
TxD0[7]	JTAG Pad Calb Bypass
	0 = Normal Operation 1 = Bypass pad calibration **NOTE:** When JTAG is working, set to ’1’.
MV64360 and MV64361: TxD1[1] **MV64362**: RstCfg[0]	Core PLL Bypass
	0 = Normal Operation 1 = Bypass the core’s PLL
MV64360 and MV64361: TxD1[4:2] **MV64362**: RstCfg[3:1]	Core PLL Control
	Tuning of the core PLL clock tree. Set to ’000’. **NOTE:** May be changed following silicon testing.

Note

Most of the reset sampled values are registered in Reset Sample Low and High registers. This is useful for board debug.

Part of the PCI interface signals are also sampled on PCI reset de-assertion, as specified in the PCI specification.

Table 118: PCI Reset Configuration

Pin	Configuration Function
64EN#	PCI_0 64-bit Enable **NOTE:** Only applies to the MV64360 and MV64362 devices.
	0 = Enabled. PCI interface is working in 64-bit mode. 1 = Disabled. PCI interface is working in 32-bit mode.
M66EN0	PCI_0 66MHz Enable
	0 = Disabled 1 = Enabled **NOTE:** Only relevant to conventional PCI mode.
DEVSEL0#, STOP0#, TRDY0#	PCI_0 Mode
	111 = Conventional PCI 110 = 66 MHz PCI-X 101 = 100 MHz PCI-X 100 = 133 MHz PCI-X All other combinations are Reserved.
REQ641#	PCI_1 64-bit Enable **NOTE:** Only applies to the MV64360 and MV64362 devices.
	Same as PCI_0 64EN#
M66EN1	PCI_1 66 MHz Enable **NOTE:** Only applies to the MV64360 and MV64362 devices.
	Same as PCI_0.
DEVSEL1#, STOP1#, TRDY1#	PCI_1 Mode **NOTE:** Only applies to the MV64360 and MV64362 devices.
	Same as PCI_0 Mode.

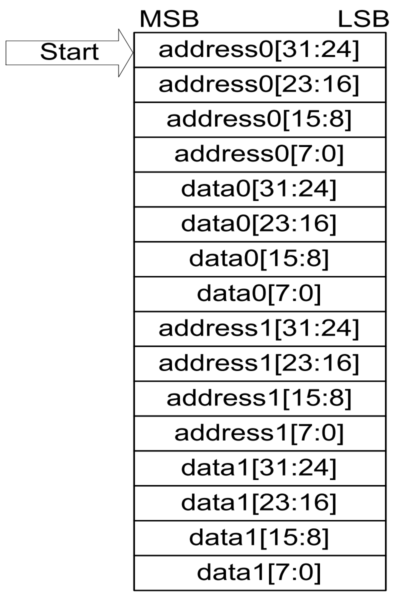
27.3 Serial ROM Initialization

The MV64360/1/2 supports initialization of ALL it's internal and configuration registers and other system components through the TWSI master interface. If serial ROM initialization is enabled, the MV64360/1/2 TWSI master starts reading initialization data from serial ROM and writes it to the appropriate registers (or to any of MV64360/1/2 interfaces, according to address decoding).

27.3.1 Serial ROM Data Structure

Serial ROM data structure consists of a sequence of 32-bit address and 32-bit data pairs, as shown in Figure 74.

Figure 74: Serial ROM Data Structure



The serial ROM initialization logic shares the address decoding registers of the MPSCs. It reads eight bytes at a time. It performs address decoding on the 32-bit address being read, and based on address decoding result, writes the next four bytes to the required target. This scheme enables not only to program the MV64360/1/2 internal registers, but also to initialize other system components. The only limitation is that it supports only single 32-bit writes (no byte enables nor bursts are supported). For example, it is possible to:

- Program the MV64360/1/2 internal registers by setting addresses that match the MV64360/1/2 internal registers space (default address is 0x14000XXX).
- Program the MV64360/1/2 PCI configuration registers using the PCI Configuration Address and PCI Configuration Data registers.
- Initialize other devices on the PCI bus by initiating PCI write transactions.

The Serial Init Last Data register (Table 658 on page 704)contains the expected value of last serial data item (default value is 0XFFFFFFFF). When the MV64360/1/2 reaches last data, it stops the initialization sequence.

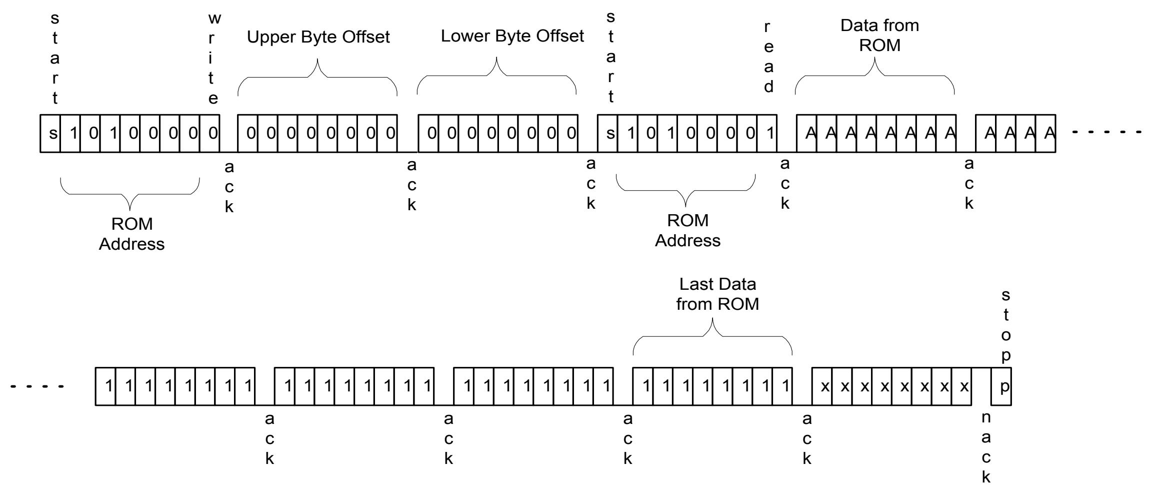
Note

Regardless of the CPU endianess setting, the serial ROM data must be set in Little Endian convention.

27.3.2 Serial ROM Initialization Operation

On SysRst# de-assertion, the MV64360/1/2 starts the initialization process. It first performs a dummy write access to the serial ROM, with data byte(s) of 0x0, in order to set the ROM byte offset to 0x0. Then, it performs the sequence of reads, until reaches last data item, as shown in Figure 75.

Figure 75: Serial ROM Read Example



For a detailed description of TWSI implementation, see Section 23. “Two-Wire Serial Interface (TWSI)” on page 331.

Notes

- Initialization data must be programmed in the serial ROM starting at offset 0x0
- The MV64360/1/2 assumes 7-bit serial ROM address of ‘b10100XX. The value of XX is sampled at reset (see 27.3.3 Serial ROM Initialization in Multi-MV Configuration).
- The MV64360/1/2 supports only serial ROM with a byte offset wider than eight bits. Setting the ROM byte offset to ‘0’ means the MV64360/1/2 performs a dummy write of two bytes. This is different than the GT-64240/60 device that also supports smaller byte offset).
- After receiving the last data identifier (default value is 0XFFFF.FFFF), the MV64360/1/2 receives an additional byte of dummy data. It responds with no-ack and then asserts the stop bit.

27.3.3 Serial ROM Initialization in Multi-MV Configuration

In multi-MV configuration, each MV64360/1/2 device must have its own serial ROM initialization code.

The Serial ROM address bits[1:0] are sampled at reset. Each MV64360/1/2 device must be strapped to a different value, thus having different serial ROM slave addresses.

Each serial ROMs treats slave address bits[1:0] differently. Some serial ROMs use these bits as device chip select. In this case, each slave address corresponds to a different serial ROM. This means that every MV64360/1/2 device has its own ROM on the TWSI bus. Other serial ROMs use these bits as an internal page select. In this case, one serial ROM is shared between all MV64360/1/2 devices.

On SysRst# de-assertion, all devices attempt to read from the serial ROM(s). However, since each one of them has a different initialization start address (address bits[1:0] differ), only one master device gains bus ownership. The rest looses arbitration and needs to wait until the first one finish its initialization. This way, each device eventually gains bus mastership and is able to read its ROM and perform initialization.

27.3.4 Other Interfaces During Initialization

During initialization, any PCI attempt to access the MV64360/1/2 results in retry termination. This allows the initialization sequence to program all PCI related registers, prior to an OS access to the MV64360/1/2.

Also, the DRAM initialization sequence is postponed until serial initialization completes. This guarantees that the SDRAM Mode registers are updated to the right values prior to DRAM initialization.

Note

Do not use serial ROM initialization to initialize the DRAM.

The CPU access might also need to be postponed until initialization is done. This is achieved by using external hardware to keep the CPU under reset for the entire initialization period. To identify when initialization is done, one of the MPP pins can be configured via the initialization code to act as initialization active output.

Section 28. Clocking

The MV64360/1/2 have multiple clock domains:

- SysClk. This is the system clock, which is also the MV64360/1/2 core clock. Up to 133MHz.

Note

Unlike the GT-64240/60 devices, the MV64360/1/2 does not support a separate CPU interface clock domain. The CPU interface is driven with the core clock.

- DRAM clock. Enables running the DRAM at frequency higher than the core clock (up to 183 MHz). This clock domain is optional.
- PCI_0 clock. Up to 66MHz in conventional PCI mode, and up to 133MHz in PCI-X mode.
- PCI_1 clock. Up to 66MHz in conventional PCI mode, and up to 133MHz in PCI-X mode.
- GbE clock. Up to 125MHz in GMII or TBI modes.
- MPSCs clock. Up to 50MHz.

Notes

- The PCI_1 clock only applies to the MV64360 and MV64361 devices.
- The PCI clock frequency must not exceed SysClk frequency.

28.1 PLL and DLL

The MV64360/1/2 has the following on-chip PLL s and DLLs to improve its AC timing:

- SysClk PLL. (PLL0) drives the internal core clock tree.
- DRAM clock PLL. (PLL1) drives the DDR DRAM interface clock tree. This is only applicable when running the DRAM at frequency higher than the core clock (see 11.11 “DRAM Clocking” on page 144)
- PCI0/1 DLL is used in 66MHz conventional PCI mode and in PCI-X mode for driving the PClk clock tree.

28.2 PLL Power Supply

The MV64360/1/2 PLLs and DLLs use the regular digital VDD and VSS power supplies, as well as the following analog power supplies:

- AVDD0 (D06) - SysClock PLL analog power supply (3.3V)
- AVDD1 (A05) - DRAM PLL analog power supply (3.3V)
- AVSS0 (E06), AVSS0_A (F06), and AVSS1 (B05) are quiet ground.
- PAVDD (AM14) - PCI DLLs analog power supply (1.8V)
- PAVSS (AN14) is quiet ground.

In addition PLL0 and PLL1 has a common RES pin, that should be connected to AVSS0 via 12.1Kohm resistor.

Section 29. Electrical Specifications

Note

The following parameters are preliminary and subject to change.

The HSTL, SSTL, and PCI electrical specifications meet the following standards.

- HSTL: EIA/JEDEC standard EIA/JESD8-6 (High speed transceiver logic (HSTL) A 1.5V output buffer supply voltage based interface standard for digital integrated circuits)
- SSTL: EIA/JEDEC standard EIA/JESD8-9 (Stub series terminated logic for 2.5 volts, SSTL_2)
- PCI spec 2.2 and PCI-X spec 1.0

29.1 Powering Up and Powering Down

Before powering up or powering down, refer to *AN-67: Powering Up and Powering Down Marvell Technology Integrated Circuits* available from Marvell.

Note

In Hot Insertion applications, never allow a state to exist in which PCI signals are active while Vref is not powered. In such a case, Vref must be floated (disconnected) or powered and never allowed to be 0V (tied to VSS). Otherwise, any input to the PCI-X pads may result in a damage to the chip.

29.2 Absolute and Recommended Operating Conditions

Table 119: Absolute Maximum Ratings

Symbol	Parameter	Min.	Max.	Unit
V_{DD} I/O	I/O Supply Voltage (all interfaces, excluding SDRAM)	-0.5	5.0	V
V_{DD} Core	Core Supply Voltage	-0.5	5.0	V
V_i	Pin Input Voltage (all interfaces, excluding 5V tolerant)	-0.5	V_{DD} +0.7V	
	Pin Input Voltage (5V tolerant)	-0.5	5.5	V
T_c	Operating Case Temperature	0	120	C
T_{stg}	Storage Temperature	-40	125	C

Notes

- It is recommended to read *AN-63: Thermal Management for Galileo Technology Products*, available from Marvell, before designing a system. This application note describes basic understanding of thermal management of integrated circuits (ICs) and guidelines to ensure optimal operating conditions.

- Operation at or beyond the maximum ratings is not recommended or guaranteed. Extended exposure at the maximum rating for extended periods of time may adversely affect device reliability.

Table 120: Recommended Operating Conditions

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{DD} Core	Core Supply Voltage	1.70	1.80	1.90	V
V_{DD} CPU	CPU interface I/O Supply Voltage @ 1.8V	1.70	1.80	1.90	V
	CPU interface I/O Supply Voltage @ 2.5V	2.38	2.50	2.62	V
	CPU interface I/O Supply Voltage @ 3.3V	3.15	3.3	3.45	V
V_{DD} DRAM	DRAM interface I/O Supply Voltage (SSTL)	2.30	2.50	2.70	V
AVDD0/1	PLL Supply Voltage	3.15	3.3	3.45	V
PAVDD	PCI DLL Supply Voltage	1.70	1.80	1.90	V
V_{DD} 3.3V	I/O Supply Voltage (all other interfaces)	3.15	3.3	3.45	V
V_i CPU 1.8V	CPU interface Input Voltage @ 1.8V	0		1.9	V
V_i CPU 2.5V	CPU interface Input Voltage @ 2.5V	0		2.62	V
V_i CPU 3.3V	CPU interface Input Voltage @ 3.3V	0		3.45	V
V_i 3.3V	Input Voltage (all other interfaces)	0		3.45	V
T_c	Operating Case Temperature	0		110	C

Table 121: Pin Capacitance

Symbol	Parameter	Min.	Typ.	Max.	Unit
C	Pin Capacitance	2.5		5.7	pF

29.3 DC Electrical Characteristics Over Operating Range

Table 122: DC Electrical Characteristics Over Operating Range

NOTE: To determine the input and output signals for the various interfaces, see Table 123 on page 361.

Symbol	Parameter	Test Conditions	Min.	Max.	Unit
V_{ih} CPU 1.8V	CPU interface Input HIGH level @ 1.8V		1.20		V
V_{il} CPU 1.8V	CPU interface Input LOW level @ 1.8V			0.60	V
V_{OH} CPU 1.8V	CPU interface output HIGH level @ 1.8V		1.35		V
V_{OL} CPU 1.8V	CPU interface output LOW level @ 1.8V			0.45	V
V_{ih} CPU 2.5V	CPU interface Input HIGH level @ 2.5V		1.70		V
V_{il} CPU 2.5V	CPU interface Input LOW level @ 2.5V			0.70	V
V_{OH} CPU 2.5V	CPU interface output HIGH level @ 2.5V		1.8		V
V_{OL} CPU 2.5V	CPU interface output LOW level @ 2.5V			0.60	V
V_{ih} CPU 3.3V	CPU interface Input HIGH level @ 3.3V		2.00		V
V_{il} CPU 3.3V	CPU interface Input LOW level @ 3.3V			0.8	V
V_{OH} CPU 3.3V	CPU interface output HIGH level @ 3.3V		2.40		V
V_{OL} CPU 3.3V	CPU interface output LOW level @ 3.3V			0.40	V
V_{ih}	Input HIGH level (all other interfaces, excluding HSTL and SSTL)		2.00		V
V_{il}	Input LOW level (all other interfaces, excluding HSTL and SSTL)			0.80	V
V_{oh}	Output HIGH Voltage (all other interfaces, excluding HSTL and SSTL)	I_{OH} = 8 mA	2.40		V
V_{ol}	Output LOW Voltage (all other interfaces, excluding HSTL and SSTL)	I_{OH} = 8 mA		0.40	V
V_{oh}	Output HIGH Voltage (all other interfaces, excluding HSTL and SSTL)	I_{OH} = 12 mA	2.40		V
V_{ol}	Output LOW Voltage (all other interfaces, excluding HSTL and SSTL)	I_{OH} = 12 mA		0.40	V
I_{ozh}	High Impedance Output Current			TBD	uA
I_{ozl}	High Impedance Output Current			TBD	uA

Table 122: DC Electrical Characteristics Over Operating Range (Continued)

NOTE: To determine the input and output signals for the various interfaces, see Table 123 on page 361.

Symbol	Parameter	Test Conditions	Min.	Max.	Unit
Icc	Operating Current **NOTE:** The MV64360/1/2 power consumption depends on the frequency, the voltage, and on which of the interfaces are used.	I/O Vcc I/O = 3.45 V Vcc CPU = 3.45 V f = 133MHz SysClk / Pclk0/1		370	mA
		Vcc DDR = 2.62 V		460	mA
		Core Vcc CORE = 1.9 V f = 133MHz SysClk / Pclk0/1		1340	mA

Table 123: Interface Input and Output Signal Guide

Interface Inputs/Outputs	Signals
CPU interface inputs	A[0-35], AP[0-3], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0/1#,HIT0/1#, ARTRY#, TS#, BR0/1#, AACK#, TA#, GT_BG#, GT_DBG#
DRAM interface inputs	DQ[63:0], CB[7:0], DM[8:0], DQS[8:0], FDBClkIn, StartBurstIn
PCI interface inputs	**MV64360:** PCLK0/1, Rst0/1#, FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, GNT0/1# **MV64361:** PCLK0/1, Rst0/1#, FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, PERR0/1#, PAD0/1[31:0], CBE0/1[3:0]#, PAR0/1, GNT0/1# **MV64362:** PCLK, Rst#, FRAME#, IRDY#, TRDY#, STOP#, IDSEL, DEVSEL#, REQ64#, ACK64#, PAR64, PERR#, PAD[63:0], CBE[7:0]#, PAR, GNT#
All other inputs	SysRst#, SysClk, JTRST#, JTMS, JTCLK, JTDI, Col1, Col0, Ready#, DevAD[31:0], DevDP[3:0], TxClk0/1, RxD0/1[7:0], RxDV0/1, RxClk0/1, RxErr0/1, CRS0/1, CLK125, MDIO, SDA **NOTE:** The TxClk1, RxD1[7:0], RxDV1, RxClk1, RxErr1, and CRS1 signals **only apply** to the MV64360 and MV64361 devices.

Table 123: Interface Input and Output Signal Guide (Continued)

Interface Inputs/Outputs	Signals
CPU interface outputs	TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31],DP[0-7], DTI[0-2], TA#, DBB#, BG0#, BG1#, DBG0#, DBG1#, GT_BR#
DRAM interface outputs	ClkOut, ClkOut# FdbClkOut, DQ[63:0], CB[7:0], DM[8:0], DQS[8:0], A[13:0], RAS#, CAS#, WE#, BA[1:0], CS[3:0]#, StartBurst# *
PCI interface outputs	**MV64360:** FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, SERR0/1#, REQ0/1# Int0/1 **MV64361:** FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, PERR0/1#, PAD0/1[31:0], CBE0/1[3:0]#, PAR0/1, SERR0/1#, REQ0/1# Int0/1 **MV64362:** FRAME#, IRDY#, TRDY#, STOP#, IDSEL, DEVSEL#, REQ64#, ACK64#, PAR64, PERR#, PAD[63:0], CBE[7:0]#, PAR, SERR#, REQ# Int
All other 8mA outputs	JTDO, SDA, SCK
All other 12mA outputs	DevWE[3:0]#, BAdr[2:0], CSTiming#, ALE, DevAD[31:0], DevDP[3:0], Txd0/1[7:0], TxErr0/1, TxEn0/1, TxClkOut0/1, MDIO, MDC **NOTE:** The Txd1[7:0], TxErr1, TxEn1, TxClkOut1 signals **only apply** to the MV64360 and MV64361 devices.

Note

All MPP[31:0] signals can be input and outputs because they use the PCI pads. They also share the same DC characteristics as the PCI pins.

29.4 Thermal Data

Table 124 shows the package thermal data for the MV64360/1/2.

Note

It is recommended to read AN-63: Thermal Management for Galileo Technology Products, available from Marvell, before designing a system. This application note describes basic understanding of thermal management of integrated circuits (ICs) and guidelines to ensure optimal operating conditions for Marvell Technology products.

Table 124: Thermal Data for The MV64360/1/2 in BGA 665

Symbol	Definition	Value per different air flows (m/s)			Unit
		0	1	2	
Θja	Thermal resistance: junction to ambient.	11.4	10.5	9.2	C/W
Ψjt	Thermal characterization parameter: junction to case center.	0.25	0.27	0.33	C/W
Θjc	Thermal resistance: junction to case (not air-flow dependent)	4			C/W

Section 30. Commercial AC Timing Specifications

Note

The following AC timing parameters are preliminary and subject to change.

30.1 Clock Timing

Table 125: Clock AC Timing

Signals	Description	Min.	Max.	Units	Load
SysClk	Frequency	66	133	MHz	
SysClk	Cycle Time	7.5	15	ns	
SysClk	Duty Cycle	40	60	%	
SysClk	Slew Rate	1		V/ns	
SysClk	Jitter		100	ps	
SysClk	PLL Lock Time		1	ms	

30.2 PowerPC CPU Interface

Notes

- All CPU interface Output Delays, Setup, and Hold times are referred to SysClk rising edge.

Table 126: PowerPC CPU AC Timing

Signals	Description	Min.	Max.	Units	Load
CPU Interface - 3.3V TTL					
A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0#, DRDY1#, HIT0#, HIT1#	Setup	1.9		ns	
ARTRY#, TS#, BR0#, BR1#	Setup	2.4		ns	
AACK#, TA#, ARTRY# **NOTE:** The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.	Setup	5.7		ns	

Table 126: PowerPC CPU AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
GT_BG#, GT_DBG#, TS#	Setup	2.7		ns	
NOTE: GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also, TS* setup time is a bit higher when using external bus arbiter.					
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], BR0#, BR1#, DRDY0#, DRDY1#, HIT0#, HIT1#, ARTRY#, AACK#, TA#, GT_BG#, GT_DBG#	Hold	0		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31], DP[0-7], DTI[0-2], TA#, DBB#	Output Delay	1.0	3.5	ns	See Figure 76.
BG0#, BG1#, DBG0#, DBG1#	Output Delay	1.0	3.4	ns	
GT_BR#	Output Delay	1.0	3.4	ns	
CPU Interface - 2.5V TTL					
A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0#, DRDY1#, HIT0#, HIT1#	Setup	2.0		ns	
ARTRY#, TS#, BR0#, BR1#	Setup	2.5		ns	
AACK#, TA#, ARTRY#	Setup	5.8		ns	
NOTE: The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.					
GT_BG#, GT_DBG#, TS#	Setup	2.8		ns	
NOTE: GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also, TS* setup time is a bit higher when using external bus arbiter.					
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], BR0#, BR1#, DRDY0#, DRDY1#, HIT0#, HIT1#, ARTRY#, AACK#, TA#, GT_BG#, GT_DBG#	Hold	0		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31], DP[0-7], DTI[0-2], TA#, DBB#	Output Delay	1.0	3.9	ns	See Figure 76.
BG0#, BG1#, DBG0#, DBG1#	Output Delay	1.0	3.7	ns	
GT_BR#	Output Delay	1.0	3.7	ns	

Table 126: PowerPC CPU AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
CPU Interface - 1.8V TTL					
A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0#, DRDY1#, HIT0#, HIT1#	Setup	2.2		ns	
ARTRY#, TS#, BR0#, BR1#	Setup	2.7		ns	
AACK#, TA#, ARTRY# **NOTE:** The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.	Setup	6.0		ns	
GT_BG#, GT_DBG#, TS# **NOTE:** GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also, TS* setup time is a bit higher when using external bus arbiter.	Setup	3.0		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], BR0#, BR1#, DRDY0#, DRDY1#, HIT0#, HIT1#, ARTRY#, AACK#, TA#, GT_BG#, GT_DBG#	Hold	0		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31], DP[0-7], DTI[0-2], TA#, DBB#	Output Delay	1.0	4.5	ns	See Figure 76.
BG0#, BG1#, DBG0#, DBG1#	Output Delay	1.0	4.3	ns	
GT_BR#	Output Delay	1.0	4.3	ns	

30.3 DDR SDRAM Interface

Notes

- To run the SDRAM in sync mode (which is the recommended mode for minimum DRAM access latency), the SDRAM ClkOut trace and DQS trace lengths must be minimal. At this time, preliminary board simulations with 5.7" trace length, and up to two double sided registered DIMMs, shows proper operation in sync mode. This 5.7" trace includes the portion of the trace length within the DIMM to the farthest DRAM device.

- If using un-buffered DIMMs (or DRAM devices on board), ClkOut/ClkOut3 pair must be distributed to the DIMMs via a zero delay (PLL) clock buffer.
- The DRAM interface output delays are measured from ClkOut rising edge (ClkOut/ClkOut# crossing point) to V_{TT}.

Table 127: DDR SDRAM AC Timing 133 Mhz Frequency

Signals	Description	Min.	Max.	Units	Load
ClkOut/ClkOut#	Frequency	100	133	MHz	
ClkOut/ClkOut#	Cycle Time	7.5	10	ns	
ClkOut/ClkOut#	Duty Cycle	48	52	%	
ClkOut/ClkOut#	Slew Rate	1		V/ns	
DQ[63:0], CB[7:0]	Skew (Data input skew relative to DQS input.)	-1.2	1.5	ns	
A[13:0], BA[1:0]	Address output delay (relative to ClkOut)	0.9	2.2	ns	See Figure 77.
RAS#, CAS#, WE#, CS[3:0]#, StartBurst#	Control output delay (relative to ClkOut)	0.9	2.1	ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DS} (Data output setup relative to DQS.)	0.6		ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DH} (Data output hold relative to DQS.)	1.7		ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DIPW} (Data output pulse width.)	1.25		ns	

Table 128: DDR SDRAM AC Timing 183 Mhz Frequency

Signals	Description	Min.	Max.	Units	Load
ClkOut/ClkOut#	Frequency	133	183	MHz	
ClkOut/ClkOut#	Cycle Time	5.5	7.5	ns	
ClkOut/ClkOut#	Duty Cycle	48	52	%	
ClkOut/ClkOut#	Slew Rate	1		V/ns	
DQ[63:0], CB[7:0]	Skew (Data input skew relative to DQS input.)	-0.78	1.09	ns	

Table 128: DDR SDRAM AC Timing 183 Mhz Frequency (Continued)

Signals	Description	Min.	Max.	Units	Load
A[13:0], BA[1:0]	Address output delay (relative to ClkOut)	0.9	2.2	ns	See Figure 77.
RAS#, CAS#, WE#, CS[3:0]#, StartBurst#	Control output delay (relative to ClkOut)	0.9	2.1	ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DS} (Data output setup relative to DQS.)	0.65		ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DH} (Data output hold relative to DQS.)	0.92		ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DIPW} (Data output pulse width.)	1.25		ns	

30.4 Device Interface

Note

All Device interface setup, hold, and output delay times are referred to the SysClk's rising edge.

Table 129: Device Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
DevAD[31:0], DevDP[3:0]	Setup	1.9		ns	
Ready#	Setup	2.0		ns	
DevAD[31:0], DevDP[3:0]	Hold	0		ns	
Ready#	Hold	0		ns	
CSTiming#, DevWE[3:0]#, ALE, BADR[2:0]	Control output delay	1.0	4.4	ns	30pF
DevAD[31:0], DevDP[3:0]	Data output delay	1.0	4.4	ns	

30.5 PCI Interface in PCI-X Mode

Notes

- In this mode, the PCI interface DLL is enabled.
- All PCI interface setup, hold, and output delay times are referred to the PCLK's rising edge.
- REQ0/1# and GNT0/1# are only relevant when using an external PCI bus arbiter.

Table 130: PCI Interface in PCI-X Mode AC Timing

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Frequency	60	133	MHz	
PCLK0/1	Cycle Time	7.5	15	ns	
PCLK0/1	Duty Cycle	40	60	%	
PCLK0/1	Slew Rate	1.5	4	V/ns	
Rst0/1#	Active Time	1.0		ms	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Setup (Bused Signals)	1.2		ns	
GNT0/1#	Setup (Point-to-Point Signals)	1.2		ns	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, GNT0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Hold	0.5		ns	

Table 130: PCI Interface in PCI-X Mode AC Timing (Continued)
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, SERR0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Output Delay (Bused Signals)	0.7	4.0	ns	See Figure 78.
REQ0/1#	Output Delay (Point-to-Point Signals)	0.7	4.2	ns	

30.6 PCI Interface in Conventional PCI Mode at 66 MHz

Notes

- In this mode, 66En is sampled active and the DLL is enabled.
- All PCI interface setup, hold, and output delay times are referred to the PCLK's rising edge.
- REQ0/1# and GNT0/1# are only relevant when using an external PCI bus arbiter.

Table 131: PCI Interface in Conventional PCI Mode at 66 MHz AC Timing
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Frequency	33	66	MHz	
PCLK0/1	Cycle Time	15	30	ns	
PCLK0/1	Duty Cycle	40	60	%	
PCLK0/1	Slew Rate	1.5	4	V/ns	
Rst0/1#	Active Time	1.0		ms	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Setup (Bused Signals)	3.2		ns	

Table 131: PCI Interface in Conventional PCI Mode at 66 MHz AC Timing (Continued)
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
GNT0/1#	Setup (Point-to-Point Signals)	3.9		ns	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, GNT0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Hold	0.1		ns	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, SERR0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Output Delay (Bused Signals)	2.0	6.0	ns	See Figure 78.
REQ0/1#	Output Delay (Point-to-Point Signals)	2.0	6.0	ns	

30.7 PCI Interface in Conventional PCI Mode at 33 MHz

Notes

- In this mode, 66En is sampled inactive and the DLL is disabled.
- All PCI interface setup, hold, and output delay times are referred to the PCLK's rising edge.
- REQ0/1# and GNT0/1# are only relevant when using an external PCI bus arbiter.

Table 132: PCI Interface in Conventional PCI Mode at 33 MHz AC Timing
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Frequency	0	33	MHz	

Table 132: PCI Interface in Conventional PCI Mode at 33 MHz AC Timing (Continued)
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Cycle Time	30	∞	ns	
PCLK0/1	Duty Cycle	40	60	%	
PCLK0/1	Slew Rate	1	4	V/ns	
Rst0#, Rst1#	Active Time	1		ms	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Setup (Bused Signals)	7.0		ns	
GNT0/1#	Setup (Point-to-Point Signals)	10.0		ns	
FRAME0/1#, IRDY0/1#, PAD0/1[63:0], TRDY0/1#, STOP0/1#, IDSEL0/1, PAR640/1, DEVSEL0/1# GNT0/1#, REQ640/1#, ACK640/1#,PAR0/1, PERR0/1#, CBE0/1[7:0]# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Hold	0.2		ns	
FRAME0/1#, TRDY0/1#, IRDY0/1#, DEVSEL0/1#, PAD0/1[63:0], STOP0/1#, CBE0/1[7:0]#, REQ640/1#, ACK640/1#, REQ0/1#, PAR0/1, PERR0/1#, SERR0/1#, PAR640/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Output Delay (Bused Signals)	2.0	11.0	ns	See Figure 78.
REQ0/1#	Output Delay (Point-to-Point Signals)	2.0	12.0	ns	

30.8 MPP Interface

Notes

- All MPP pins setup, hold, and output delay times are referred to the SysClk rising edge, unless stated otherwise.
- PME0#, PME1#, WDE#, WDNMI# are asynchronous outputs.

Table 133: MPP Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
GPP[31:0]	Setup	2.2		ns	
GPP[31:0]	Hold	1.0		ns	
GPP[15:0]	Output Delay	1.0	5.5	ns	30pF
GPP[31:16]	Output Delay	1.0	4.7	ns	30pF
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter setup time relative to PCLK, 33MHz, DLL disabled.	Setup	10.0		ns	
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter hold time relative to PCLK, 33MHz, DLL disabled.	Hold	0.2		ns	
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter setup time relative to PCLK, 66MHz, DLL enabled.	Setup	5.0		ns	
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter hold time relative to PCLK, 66MHz, DLL enabled.	Hold	0		ns	
REQ0[5:0]# **NOTE:** PCI-x arbiter setup time relative to PCLK, 133MHz, DLL enabled.	Setup	1.25		ns	
REQ1[5:0]# **NOTE:** PCI-x arbiter setup time relative to PCLK, 133MHz, DLL enabled.	Setup	1.7		ns	
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-x arbiter hold time relative to PCLK, 133MHz, DLL enabled.	Hold	0.5		ns	

Table 133: MPP Interface AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
GNT0[5:0]#, GNT1[5:0]#	Output Delay	2.0	12.0	ns	See Figure 78.
NOTE: PCI-2.2 arbiter output delay relative to PCLK, 33MHz, DLL disabled.					
GNT0[5:0]#, GNT1[5:0]#	Output Delay	2.0	6.0	ns	
NOTE: PCI-2.2 arbiter output delay relative to PCLK, 66MHz, DLL enabled.					
GNT0[5:0]#	Output Delay	0.7	3.9	ns	
NOTE: PCI-x arbiter output delay relative to PCLK, 133MHz, DLL enabled.					
GNT1[5:0]#	Output Delay	0.7	4.2	ns	
NOTE: PCI-x arbiter output delay relative to PCLK, 133MHz, DLL enabled.					
DMAReq[3:0]#	Setup	2.2		ns	
DMAReq[3:0]#	Hold	1		ns	
DMAAck[3:0]	Output Delay Sampled	1	5.5	ns	30pF
DMAAck[3:0]	Output Delay Not sampled	1	7.7	ns	30pF
EOT[3:0]	Setup	2.2		ns	
EOT[3:0]	Hold	1		ns	
TCEn[3:0]#	Setup	2.2		ns	
TCEn[3:0]#	Hold	1		ns	
TCTcnt[3:0]#	Output Delay	1	5.5	ns	30pF
DBurst#	Output Delay Not sampled	1	7.5	ns	30pF
InitAct output delay	Output Delay	1	5.5	ns	30pF
SCLK	Frequency		50	MHz	
SCLK	Cycle Time	20		ns	
TSCLK	Frequency		50	MHz	
TSCLK	Cycle Time	20		ns	
TxD0/1	Output Delay	5	12.9	ns	30pF
RxD0/1	Setup	5.5		ns	

Table 133: MPP Interface AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
RxD0/1	Hold	2		ns	
RTS0/1	Output Delay	5	12.5	ns	30pF
NOTE: TxD0/1, RxD0/1, and RTS0/1 are relative to TSCLK					
CTS0/1	Setup	2.9		ns	
CTS0/1	Hold	1.5		ns	
CD0/1	Setup	1		ns	
CD0/1	Hold	2		ns	
NOTE: CTS0/1, CD0/1 are relative to TSCLK					

30.9 Ethernet Interface AC Timing

Table 134: Ethernet Interface AC Timing

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device has only one Ethernet port.

Signals	Description	Min.	Max.	Units	Load
Ethernet PCS Interface **NOTE:** Rx[9:0] hold and setup times are referred to the RxClk[1:0] rising and falling edges. Tx[9:0] Output Delays, are referred to the TxClkOut rising edge.					
RxClk0/1[1:0] frequency	Frequency		62.5	MHz	
TxClkOut0/1 frequency	Frequency		125	MHz	
Rx0/1[9:0]	Setup	2		ns	
Rx0/1[9:0]	Hold	1		ns	
Tx0/1[9:0]	Output Delay	1.5	5.5	ns	See Figure 79
Ethernet GMII Interface **NOTE:** All input pins Setup, and Hold times are referred to the RxClk rising edge. All output pins output delays, are referred to the TxClkOut rising edge.					
RxClk0/1	Frequency		125	MHz	
TxClkOut0/1	Frequency		125	MHz	
RxD0/1[7:0], RxDv0/1, RxErr0/1	Setup	2		ns	
RxD0/1[7:0], RxDv0/1, RxErr0/1	Hold	0.2		ns	

Table 134: Ethernet Interface AC Timing (Continued)

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device has only one Ethernet port.

Signals	Description	Min.	Max.	Units	Load
TxD0/1[7:0], TxEn0/1, TxErr0/1	Output Delay	0.5	5.5	ns	See Figure 79
Ethernet MII Interface **NOTE:** All input pins Setup, and Hold times are referred to the RxClk rising edge. All output pins output delays, are referred to the TxClk rising edge.					
RxClk0/1	Frequency		25	MHz	
TxClk0/1	Frequency		25	MHz	
RxD0/1[3:0], RxDv0/1, RxErr0/1	Setup	2		ns	
RxD0/1[3:0]	Hold	1		ns	
RxDv0/1, RxErr0/1	Hold	1		ns	
TxD0/1[3:0], TxEn0/1, TxErr0/1	Output Delay	2	11	ns	See Figure 79

30.10TWSI Interface AC Timing

Note

SDA setup, hold, and output delay times are referred to SCK rising edge.

Table 135: TWSI Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
SCK	Frequency		400	KHz	
SCK	Clock Cycle	2.5		us	
SDA	Setup	4		ns	
SDA	Hold	1		ns	
SDA	Output Delay	1	6	ns	20pf

30.11JTAG Interface AC Timing

Note

JTDI and JTMS setup and hold times, and JTDO output delay, are referred to JTCLK rising edge.

Table 136: JTAG Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
JTCK	Frequency		1	MHz	
JTCK	Clock Cycle	1000		ns	
JTRST#	Active Time	1		ms	
JTDI, JTMS	Boundary scan data setup	75		ns	
JTDI, JTMS	Boundary scan data h old	10		ns	
JTDO	Boundary scan data output delay	1	75	ns	

Figure 76: CPU Interface Output Delay Test Load

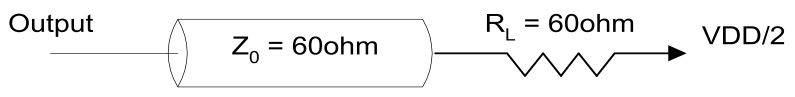
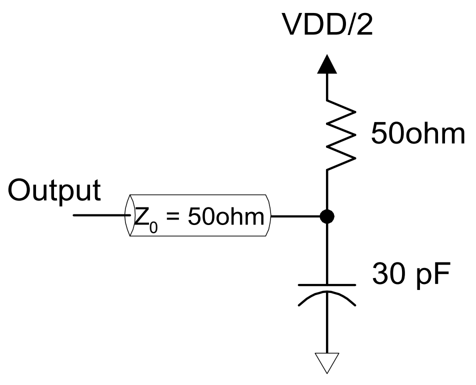


Figure 77: DRAM SSTL I/O Pads Output Delay Test Load



Note

The DRAM load applies to both CLK/CLK# and to DRAM address and control outputs.

Figure 78: PCI I/O Pads Output Delay Test Load

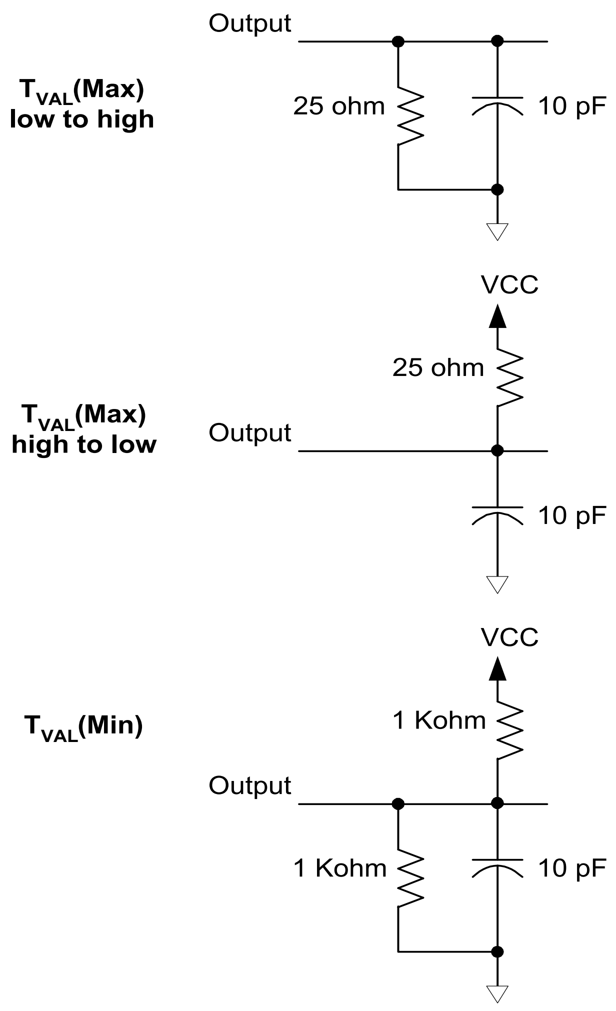
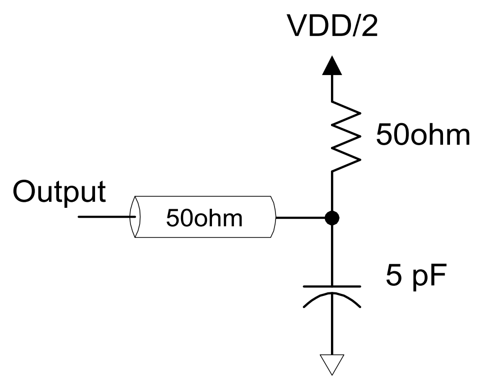


Figure 79: GMII Output Delay Test Load



Note

GMII test load applies to both Tx clock and to Tx data.

Section 31. Preliminary Industrial AC Timing Specifications (100 MHz)

Note
The following AC timing parameters are preliminary and subject to change.

31.1 Clock Timing

Table 137: Clock AC Timing

Signals	Description	Min.	Max.	Units	Load
SysClk	Frequency	66	100	MHz	
SysClk	Cycle Time	10	15	ns	
SysClk	Duty Cycle	40	60	%	
SysClk	Slew Rate	1		V/ns	
SysClk	Jitter		100	ps	
SysClk	PLL Lock Time		1	ms	

31.2 PowerPC CPU Interface

Notes
- All CPU interface Output Delays, Setup, and Hold times are referred to SysClk rising edge.

Table 138: PowerPC CPU AC Timing

Signals	Description	Min.	Max.	Units	Load
CPU Interface - 3.3V TTL					
A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0#, DRDY1#, HIT0#, HIT1#	Setup	2.9		ns	
ARTRY#, TS#, BR0#, BR1#	Setup	3.4		ns	
AACK#, TA#, ARTRY# **NOTE:** The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.	Setup	6.7		ns	

Table 138: PowerPC CPU AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
GT_BG#, GT_DBG#, TS#	Setup	3.7		ns	
NOTE: GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also TS* setup time is a bit higher when using external bus arbiter.					
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], BR0#, BR1#, DRDY0#, DRDY1#, HIT0#, HIT1#, ARTRY#, AACK#, TA#, GT_BG#, GT_DBG#	Hold	0.5		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31], DP[0-7], DTI[0-2], TA#, DBB#	Output Delay	0.7	4.7	ns	See Figure 80.
BG0#, BG1#, DBG0#, DBG1#	Output Delay	0.7	4.7	ns	
GT_BR#	Output Delay	0.7	4.7	ns	
CPU Interface - 2.5V TTL					
A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0#, DRDY1#, HIT0#, HIT1#	Setup	3.0		ns	
ARTRY#, TS#, BR0#, BR1#	Setup	3.5		ns	
AACK#, TA#, ARTRY#	Setup	6.8		ns	
NOTE: The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.					
GT_BG#, GT_DBG#, TS#	Setup	3.8		ns	
NOTE: GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also TS* setup time is a bit higher when using external bus arbiter.					
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], BR0#, BR1#, DRDY0#, DRDY1#, HIT0#, HIT1#, ARTRY#, AACK#, TA#, GT_BG#, GT_DBG#	Hold	0.5		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31], DP[0-7], DTI[0-2], TA#, DBB#	Output Delay	0.7	5.0	ns	See Figure 80.
BG0#, BG1#, DBG0#, DBG1#	Output Delay	0.7	5.0	ns	
GT_BR#	Output Delay	0.7	5.0	ns	
CPU Interface - 1.8V TTL					

Table 138: PowerPC CPU AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0#, DRDY1#, HIT0#, HIT1#	Setup	3.2		ns	
ARTRY#, TS#, BR0#, BR1#	Setup	3.7		ns	
AACK#, TA#, ARTRY# **NOTE:** The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.	Setup	7.0		ns	
GT_BG#, GT_DBG#, TS# **NOTE:** GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also TS* setup time is a bit higher when using external bus arbiter.	Setup	4.0		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], BR0#, BR1#, DRDY0#, DRDY1#, HIT0#, HIT1#, ARTRY#, AACK#, TA#, GT_BG#, GT_DBG#	Hold	0.5		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31], DP[0-7], DTI[0-2], TA#, DBB#	Output Delay	0.7	5.6	ns	See Figure 80.
BG0#, BG1#, DBG0#, DBG1#	Output Delay	0.7	5.6	ns	
GT_BR#	Output Delay	0.7	5.6	ns	

31.3 DDR SDRAM Interface

Notes

- To run the SDRAM in sync mode (which is the recommended mode for minimum DRAM access latency), the SDRAM ClkOut trace and DQS trace lengths must be minimal. At this time, preliminary board simulations with 5.7" trace length, and up to two double sided registered DIMMs, shows proper operation in sync mode. This 5.7" trace includes the portion of the trace length within the DIMM to the farthest DRAM device.
- If using un-buffered DIMMs (or DRAM devices on board), ClkOut/ClkOut3 pair must be distributed to the DIMMs via a zero delay (PLL) clock buffer.
- The DRAM interface outputs are measured from ClkOut rising edge (ClkOut/ClkOut# crossing point) to V_{TT}.

Table 139: DDR SDRAM AC Timing 133 Mhz Frequency

Signals	Description	Min.	Max.	Units	Load
ClkOut/ClkOut#	Frequency		100	MHz	

Table 139: DDR SDRAM AC Timing 133 Mhz Frequency (Continued)

Signals	Description	Min.	Max.	Units	Load
ClkOut/ClkOut#	Cycle Time	10		ns	
ClkOut/ClkOut#	Duty Cycle	47	53	%	
ClkOut/ClkOut#	Slew Rate	1		V/ns	
DQ[63:0], CB[7:0]	Skew (Data input skew relative to DQS input.)	-1.5	1.5	ns	
A[13:0], BA[1:0]	Address output delay (relative to ClkOut)	0.3	3.2	ns	See Figure 81.
RAS#, CAS#, WE#, CS[3:0]#, StartBurst#	Control output delay (relative to ClkOut)	0.3	3.2	ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DS} (Data output setup relative to DQS.)	0.4		ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DH} (Data output hold relative to DQS.)	1.4		ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DIPW} (Data output pulse width.)	TBD		ns	

31.4 Device Interface

Note

All Device interface setup, hold, and output delay times are referred to the SysClk's rising edge.

Table 140: Device Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
DevAD[31:0], DevDP[3:0]	Setup	2.9		ns	
Ready#	Setup	3.0		ns	
DevAD[31:0], DevDP[3:0]	Hold	0.5		ns	
Ready#	Hold	0.5		ns	
CSTiming#, DevWE[3:0]#, ALE, BADR[2:0]	Control output delay	0.5	5.4	ns	30pF
DevAD[31:0], DevDP[3:0]	Data output delay	0.5	5.4	ns	

31.5 PCI Interface in PCI-X Mode

Notes

- In this mode, the PCI interface DLL is enabled.
- All PCI interface setup, hold, and output delay times are referred to the PCLK's rising edge.
- REQ0/1# and GNT0/1# are only relevant when using an external PCI bus arbiter.

Table 141: PCI Interface in PCI-X Mode AC Timing

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Frequency	60	100	MHz	
PCLK0/1	Cycle Time	10	15	ns	
PCLK0/1	Duty Cycle	40	60	%	
PCLK0/1	Slew Rate	1.5	4	V/ns	
Rst0/1#	Active Time	1.0		ms	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Setup (Bused Signals)	2.0		ns	
GNT0/1#	Setup (Point-to-Point Signals)	2.0		ns	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, GNT0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Hold	0.5		ns	

Table 141: PCI Interface in PCI-X Mode AC Timing (Continued)
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, SERR0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Output Delay (Bused Signals)	0.7	4.7	ns	See Figure 82.
REQ0/1# *	Output Delay (Point-to-Point Signals)	0.7	4.9	ns	

31.6 PCI Interface in Conventional PCI Mode at 66 MHz

Notes

- In this mode, 66En is sampled active and the DLL is enabled.
- All PCI interface setup, hold, and output delay times are referred to the PCLK's rising edge.
- REQ0/1# and GNT0/1# are only relevant when using an external PCI bus arbiter.

Table 142: PCI Interface in Conventional PCI Mode at 66 MHz AC Timing
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Frequency	33	66	MHz	
PCLK0/1	Cycle Time	15	30	ns	
PCLK0/1	Duty Cycle	40	60	%	
PCLK0/1	Slew Rate	1.5	4	V/ns	
Rst0/1#	Active Time	1.0		ms	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Setup (Bused Signals)	3.2		ns	

Table 142: PCI Interface in Conventional PCI Mode at 66 MHz AC Timing (Continued)
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
GNT0/1#	Setup (Point-to-Point Signals)	3.9		ns	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, GNT0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Hold	0.4		ns	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, SERR0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Output Delay (Bused Signals)	2.0	6.0	ns	See Figure 82.
REQ0#, REQ1#	Output Delay (Point-to-Point Signals)	2.0	6.0	ns	

31.7 PCI Interface in Conventional PCI Mode at 33 MHz

Notes

- In this mode, 66En is sampled inactive and the DLL is disabled.
- All PCI interface setup, hold, and output delay times are referred to the PCLK's rising edge.
- REQ0/1# and GNT0/1# are only relevant when using an external PCI bus arbiter.

Table 143: PCI Interface in Conventional PCI Mode at 33 MHz AC Timing
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Frequency	0	33	MHz	

Table 143: PCI Interface in Conventional PCI Mode at 33 MHz AC Timing (Continued)

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Cycle Time	30	∞	ns	
PCLK0/1	Duty Cycle	40	60	%	
PCLK0/1	Slew Rate	1	4	V/ns	
Rst0#, Rst1#	Active Time	1		ms	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[6331:0], CBE0/1[74:0]#, PAR0/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Setup (Bused Signals)	7.0		ns	
GNT0/1#	Setup (Point-to-Point Signals)	10.0		ns	
FRAME0/1#, IRDY0/1#, PAD0/1[6331:0], TRDY0/1#, STOP0/1#, IDSEL0/1, PAR640/1, DEVSEL0/1# GNT0/1#, REQ640/1#, ACK640/1#,PAR0/1, PERR0/1#, CBE0/1[74:0]# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Hold	0.5		ns	

Table 143: PCI Interface in Conventional PCI Mode at 33 MHz AC Timing (Continued)

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
FRAME0/1#, TRDY0/1#, IRDY0/1#, DEVSEL0/1#, PAD0/1[6331:0], STOP0/1#, CBE0/1[74:0]#, REQ640/1#, ACK640/1#, REQ0/1#, PAR0/1, PERR0/1#, SERR0/1#, PAR640/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Output Delay (Bused Signals)	2.0	11.0	ns	See Figure 82.
REQ0/1#	Output Delay (Point-to-Point Signals)	2.0	12.0	ns	

31.8 MPP Interface

Notes

- All MPP pins setup, hold, and output delay times are referred to the SysClk rising edge, unless stated otherwise.
- PME0#, PME1#, WDE#, WDNMI# are asynchronous outputs.

Table 144: MPP Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
GPP[31:0]	Setup	3.2		ns	
GPP[31:0]	Hold	1.5		ns	
GPP[15:0]	Output Delay	0.5	6.5	ns	30pF
GPP[31:16]	Output Delay	0.5	5.7	ns	30pF
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter setup time relative to PCLK, 33MHz, DLL disabled.	Setup	10.0		ns	
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter hold time relative to PCLK, 33MHz, DLL disabled.	Hold	0.5		ns	
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter setup time relative to PCLK, 66MHz, DLL enabled.	Setup	5.0		ns	

Table 144: MPP Interface AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter hold time relative to PCLK, 66MHz, DLL enabled.	Hold	0.3		ns	
REQ0[5:0]# **NOTE:** PCI-x arbiter setup time relative to PCLK, 100MHz, DLL enabled.	Setup	2.2		ns	
REQ1[5:0]# **NOTE:** PCI-x arbiter setup time relative to PCLK, 100MHz, DLL enabled.	Setup	2.7		ns	
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-x arbiter hold time relative to PCLK, 133MHz, DLL enabled.	Hold	0.5		ns	
GNT0[5:0]#, GNT1[5:0]# **NOTE:** PCI-2.2 arbiter output delay relative to PCLK, 33MHz, DLL disabled.	Output Delay	2.0	12.0	ns	See Figure 82.
GNT0[5:0]#, GNT1[5:0]# **NOTE:** PCI-2.2 arbiter output delay relative to PCLK, 66MHz, DLL enabled.	Output Delay	1.9	6.0	ns	
GNT0[5:0]# **NOTE:** PCI-x arbiter output delay relative to PCLK, 100MHz, DLL enabled.	Output Delay	0.7	4.7	ns	
GNT1[5:0]# **NOTE:** PCI-x arbiter output delay relative to PCLK, 100MHz, DLL enabled.	Output Delay	0.7	5.0	ns	
DMAReq[3:0]#	Setup	2.9		ns	
DMAReq[3:0]#	Hold	1		ns	
DMAAck[3:0]	Output Delay Sampled	1	6.0	ns	30pF
DMAAck[3:0]	Output Delay Not sampled	1	8.2	ns	30pF
EOT[3:0]	Setup	2.9		ns	
EOT[3:0]	Hold	1		ns	
TCEn[3:0]#	Setup	2.9		ns	
TCEn[3:0]#	Hold	1		ns	

Table 144: MPP Interface AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
TCTcnt[3:0]#	Output Delay	1	6.2	ns	30pF
DBurst#	Output Delay Not sampled	1	8.0	ns	30pF
InitAct output delay	Output Delay	1	6.2	ns	30pF
SCLK	Frequency		50	MHz	
SCLK	Cycle Time	20		ns	
TSCLK	Frequency		50	MHz	
TSCLK	Cycle Time	20		ns	
TxD0/1	Output Delay	5	12.9	ns	30pF
RxD0/1	Setup	5.5		ns	
RxD0/1	Hold	2		ns	
RTS0/1 **NOTE:** TxD0/1, RxD0/1, and RTS0/1 are relative to TSCLK	Output Delay	5	12.5	ns	30pF
CTS0/1	Setup	2.9		ns	
CTS0/1	Hold	1.5		ns	
CD0/1	Setup	1		ns	
CD0/1 **NOTE:** CTS0/1, CD0/1 are relative to TSCLK	Hold	2		ns	

31.9 Ethernet Interface AC Timing

Table 145: Ethernet Interface AC Timing

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device has only one Ethernet port.

Signals	Description	Min.	Max.	Units	Load
Ethernet PCS Interface **NOTE:** Rx[9:0] hold and setup times are referred to the RxClk[1:0] rising and falling edges. Tx[9:0] Output Delays, are referred to the TxClkOut rising edge.					
RxClk0/1[1:0] frequency	Frequency		50	MHz	
TxClkOut0/1 frequency	Frequency		100	MHz	
Rx0/1[9:0]	Setup	2.5		ns	
Rx0/1[9:0]	Hold	1		ns	

Table 145: Ethernet Interface AC Timing (Continued)

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device has only one Ethernet port.

Signals	Description	Min.	Max.	Units	Load
Tx0/1[9:0]	Output Delay	1.4	6.0	ns	See Figure 83
Ethernet GMII Interface **NOTE:** All input pins Setup, and Hold times are referred to the RxClk rising edge. All output pins output delays, are referred to the TxClkOut rising edge.					
RxClk0/1	Frequency		100	MHz	
TxClkOut0/1	Frequency		100	MHz	
RxD0/1[7:0], RxDv0/1, RxErr0/1	Setup	2.5		ns	
RxD0/1[7:0], RxDv0/1, RxErr0/1	Hold	0.4		ns	
TxD0/1[7:0], TxEn0/1, TxErr0/1	Output Delay	0.5	6.0	ns	See Figure 83
Ethernet MII Interface **NOTE:** All input pins Setup, and Hold times are referred to the RxClk rising edge. All output pins output delays, are referred to the TxClk rising edge.					
RxClk0/1	Frequency		25	MHz	
TxClk0/1	Frequency		25	MHz	
RxD0/1[3:0], RxDv0/1, RxErr0/1	Setup	2		ns	
RxD0/1[3:0]	Hold	1		ns	
RxDv0/1, RxErr0/1	Hold	1		ns	
TxD0/1[3:0], TxEn0/1, TxErr0/1	Output Delay	2	11	ns	See Figure 83

31.10TWSI Interface AC Timing

Note

SDA setup, hold, and output delay times are referred to SCK rising edge.

Table 146: TWSI Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
SCK	Frequency		400	KHz	

Table 146: TWSI Interface AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
SCK	Clock Cycle	2.5		us	
SDA	Setup	4		ns	
SDA	Hold	1		ns	
SDA	Output Delay	1	6	ns	20pf

31.11JTAG Interface AC Timing

Note

JTDI and JTMS setup and hold times, and JTDO output delay, are referred to JTCLK rising edge.

Table 147: JTAG Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
JTCK	Frequency		1	MHz	
JTCK	Clock Cycle	1000		ns	
JTRST#	Active Time	1		ms	
JTDI, JTMS	Boundary scan data setup	75		ns	
JTDI, JTMS	Boundary scan data h old	10		ns	
JTDO	Boundary scan data output delay	1	75	ns	

Figure 80: CPU Interface Output Delay Test Load

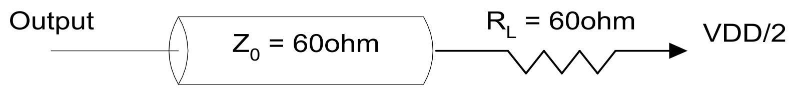
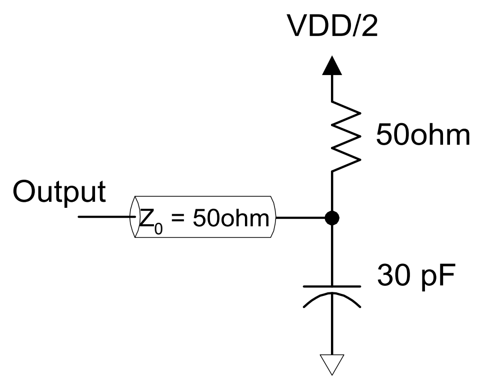


Figure 81: DRAM SSTL I/O Pads Output Delay Test Load



Note
The DRAM load applies to both CLK/CLK# and to DRAM address and control outputs.

Figure 82: PCI I/O Pads Output Delay Test Load

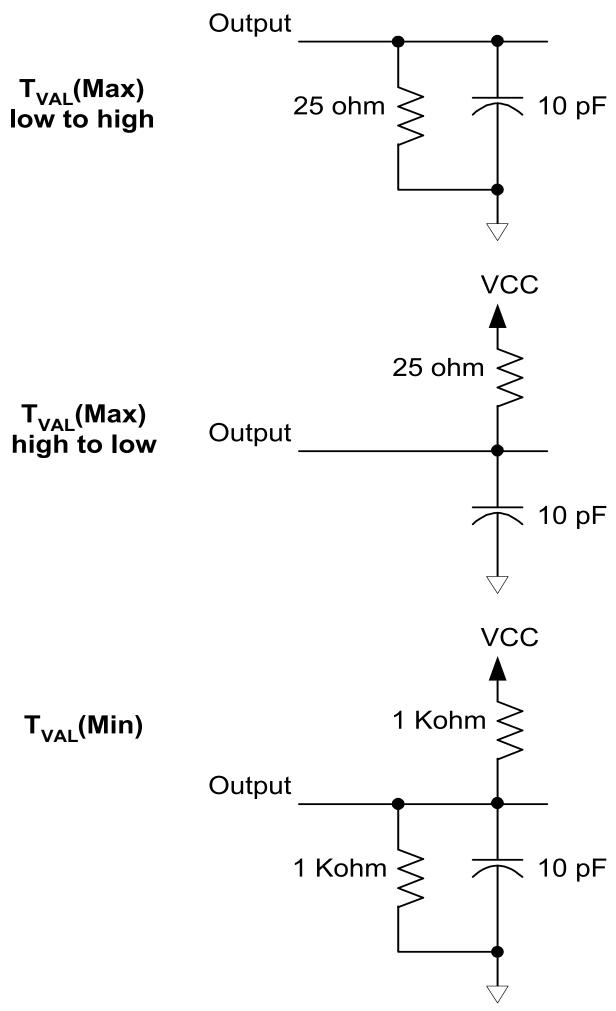
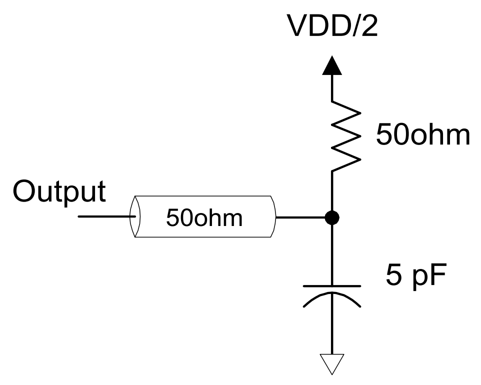


Figure 83: GMII Output Delay Test Load



Note

GMII test load applies to both Tx clock and to Tx data.

Section 32. Preliminary Industrial AC Timing Specifications (125 MHz)

Note
The following AC timing parameters are preliminary and subject to change.

32.1 Clock Timing

Table 148: Clock AC Timing

Signals	Description	Min.	Max.	Units	Load
SysClk	Frequency	66	125	MHz	
SysClk	Cycle Time	8	15	ns	
SysClk	Duty Cycle	40	60	%	
SysClk	Slew Rate	1		V/ns	
SysClk	Jitter		100	ps	
SysClk	PLL Lock Time		1	ms	

32.2 PowerPC CPU Interface

Notes
- All CPU interface Output Delays, Setup, and Hold times are referred to SysClk rising edge.
- The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.
- GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also TS* setup time is a bit higher when using external bus arbiter.

Table 149: PowerPC CPU AC Timing

Signals	Description	Min.	Max.	Units	Load
CPU Interface - 3.3V TTL					
A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0#, DRDY1#, HIT0#, HIT1#	Setup	2.4		ns	

Table 149: PowerPC CPU AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
ARTRY#, TS#, BR0#, BR1#	Setup	2.9		ns	
AACK#, TA#, ARTRY# **NOTE:** The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.	Setup	6.2		ns	
GT_BG#, GT_DBG#, TS# **NOTE:** GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also TS* setup time is a bit higher when using external bus arbiter.	Setup	3.2		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], BR0#, BR1#, DRDY0#, DRDY1#, HIT0#, HIT1#, ARTRY#, AACK#, TA#, GT_BG#, GT_DBG#	Hold	0.5		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31], DP[0-7], DTI[0-2], TA#, DBB#	Output Delay	0.7	4.0	ns	See Figure 84.
BG0#, BG1#, DBG0#, DBG1#	Output Delay	0.7	3.9	ns	
GT_BR#	Output Delay	0.7	3.9	ns	
CPU Interface - 2.5V TTL					
A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0#, DRDY1#, HIT0#, HIT1#	Setup	2.5		ns	
ARTRY#, TS#, BR0#, BR1#	Setup	3.0		ns	
AACK#, TA#, ARTRY# **NOTE:** The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.	Setup	6.3		ns	
GT_BG#, GT_DBG#, TS# **NOTE:** GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also TS* setup time is a bit higher when using external bus arbiter.	Setup	3.3		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], BR0#, BR1#, DRDY0#, DRDY1#, HIT0#, HIT1#, ARTRY#, AACK#, TA#, GT_BG#, GT_DBG#	Hold	0.5		ns	

Table 149: PowerPC CPU AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31], DP[0-7], DTI[0-2], TA#, DBB#	Output Delay	0.7	4.4	ns	See Figure 84.
BG0#, BG1#, DBG0#, DBG1#	Output Delay	0.7	4.2	ns	
GT_BR#	Output Delay	0.7	4.2	ns	
CPU Interface - 1.8V TTL					
A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], DRDY0#, DRDY1#, HIT0#, HIT1#	Setup	2.7		ns	
ARTRY#, TS#, BR0#, BR1#	Setup	3.2		ns	
AACK#, TA#, ARTRY#	Setup	6.5		ns	
NOTE: The TA# and AACK# inputs are only relevant in multi-GT mode. Also ARTRY# setup time is a bit higher in multi-GT mode.					
GT_BG#, GT_DBG#, TS#	Setup	3.5		ns	
NOTE: GT_BR#, GT_BG#, GT_DBG# are only relevant when using an external 60x bus arbiter. Also TS* setup time is a bit higher when using external bus arbiter.					
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], DH[0-31], DL[0-31], DP[0-7], BR0#, BR1#, DRDY0#, DRDY1#, HIT0#, HIT1#, ARTRY#, AACK#, TA#, GT_BG#, GT_DBG#	Hold	0.5		ns	
TS#, A[0-35], AP[3:0], TBST#, TSIZ[0-2], TT[0-4], GBL#, AACK#, ABB#, DH[0-31], DL[0-31], DP[0-7], DTI[0-2], TA#, DBB#	Output Delay	0.7	5.0	ns	See Figure 84.
BG0#, BG1#, DBG0#, DBG1#	Output Delay	0.7	4.8	ns	
GT_BR# **	Output Delay	0.7	4.8	ns	

32.3 DDR SDRAM Interface

Notes

- To run the SDRAM in sync mode (which is the recommended mode for minimum DRAM access latency), the SDRAM ClkOut trace and DQS trace lengths must be minimal. At this time, preliminary board simulations with 5.7" trace length, and up to two double sided registered DIMMs, shows proper operation in sync mode. This 5.7" trace includes the portion of the trace length within the DIMM to the farthest DRAM device.

- If using un-buffered DIMMs (or DRAM devices on board), ClkOut/ClkOut3 pair must be distributed to the DIMMs via a zero delay (PLL) clock buffer.
- The DRAM interface output delays are measured from ClkOut rising edge (ClkOut/ClkOut# crossing point) to V_{TT}.

Table 150: DDR SDRAM AC Timing 125 Mhz Frequency

Signals	Description	Min.	Max.	Units	Load
ClkOut/ClkOut#	Frequency	100	125	MHz	
ClkOut/ClkOut#	Cycle Time	8	10	ns	
ClkOut/ClkOut#	Duty Cycle	48	52	%	
ClkOut/ClkOut#	Slew Rate	1		V/ns	
DQ[63:0], CB[7:0]	Skew (Data input skew relative to DQS input.)	-1.5	1.5	ns	
A[13:0], BA[1:0]	Address output delay (relative to ClkOut)	0.5	2.7	ns	See Figure 85.
RAS#, CAS#, WE#, CS[3:0]#, StartBurst#	Control output delay (relative to ClkOut)	0.5	2.7	ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DS} (Data output setup relative to DQS.)	0.4		ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DH} (Data output hold relative to DQS.)	1.4		ns	
DQ[63:0], CB[7:0], DM[8:0]	t_{DIPW} (Data output pulse width.)	TBD		ns	

32.4 Device Interface

Note

All Device interface setup, hold, and output delay times are referred to the SysClk's rising edge.

Table 151: Device Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
DevAD[31:0], DevDP[3:0]	Setup	2.4		ns	

Table 151: Device Interface AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
Ready#	Setup	2.5		ns	
DevAD[31:0], DevDP[3:0]	Hold	0.5		ns	
Ready#	Hold	0.5		ns	
CSTiming#, DevWE[3:0]#, ALE, BADR[2:0]	Control output delay	0.5	4.9	ns	30pF
DevAD[31:0], DevDP[3:0]	Data output delay	0.5	4.9	ns	

32.5 PCI Interface in PCI-X Mode

Notes

- In this mode, the PCI interface DLL is enabled.
- All PCI interface setup, hold, and output delay times are referred to the PCLK's rising edge.
- REQ0/1# and GNT0/1# are only relevant when using an external PCI bus arbiter.

Table 152: PCI Interface in PCI-X Mode AC Timing

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Frequency	60	125	MHz	
PCLK0/1	Cycle Time	8	15	ns	
PCLK0/1	Duty Cycle	40	60	%	
PCLK0/1	Slew Rate	1.5	4	V/ns	
Rst0/1#	Active Time	1.0		ms	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Setup (Bused Signals)	1.6		ns	
GNT0/1#	Setup (Point-to-Point Signals)	1.6		ns	

Table 152: PCI Interface in PCI-X Mode AC Timing (Continued)
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, GNT0/1#	Hold	0.5		ns	
NOTE: The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.					
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, SERR0/1#	Output Delay (Bused Signals)	0.7	4.2	ns	See Figure 86.
NOTE: The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.					
REQ0/1#	Output Delay (Point-to-Point Signals)	0.7	4.4	ns	

32.6 PCI Interface in Conventional PCI Mode at 66 MHz

Notes

- In this mode, 66En is sampled active and the DLL is enabled.
- All PCI interface setup, hold, and output delay times are referred to the PCLK’s rising edge.
- REQ0/1# and GNT0/1# are only relevant when using an external PCI bus arbiter

Table 153: PCI Interface in Conventional PCI Mode at 66 MHz AC Timing
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Frequency	33	66	MHz	
PCLK0/1	Cycle Time	15	30	ns	
PCLK0/1	Duty Cycle	40	60	%	
PCLK0/1	Slew Rate	1.5	4	V/ns	
Rst0/1#	Active Time	1.0		ms	

Table 153: PCI Interface in Conventional PCI Mode at 66 MHz AC Timing (Continued)

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Setup (Bused Signals)	3.2		ns	
GNT0/1#	Setup (Point-to-Point Signals)	3.9		ns	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, GNT0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Hold	0.4		ns	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[63:0], CBE0/1[7:0]#, PAR0/1, SERR0/1# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Output Delay (Bused Signals)	2.0	6.0	ns	See Figure 86.
REQ0/1#	Output Delay (Point-to-Point Signals)	2.0	6.0	ns	

32.7 PCI Interface in Conventional PCI Mode at 33 MHz

Notes

- In this mode, 66En is sampled inactive and the DLL is disabled.
- All PCI interface setup, hold, and output delay times are referred to the PCLK's rising edge.
- REQ0/1# and GNT0/1# are only relevant when using an external PCI bus arbiter

Table 154: PCI Interface in Conventional PCI Mode at 33 MHz AC Timing
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
PCLK0/1	Frequency	0	33	MHz	
PCLK0/1	Cycle Time	30	∞	ns	
PCLK0/1	Duty Cycle	40	60	%	
PCLK0/1	Slew Rate	1	4	V/ns	
Rst0#, Rst1#	Active Time	1		ms	
FRAME0/1#, IRDY0/1#, TRDY0/1#, STOP0/1#, IDSEL0/1, DEVSEL0/1#, REQ640/1#, ACK640/1#, PAR640/1, PERR0/1#, PAD0/1[6331:0], CBE0/1[74:0]#, PAR0/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Setup (Bused Signals)	7.0		ns	
GNT0/1#	Setup (Point-to-Point Signals)	10.0		ns	
FRAME0/1#, IRDY0/1#, PAD0/1[6331:0], TRDY0/1#, STOP0/1#, IDSEL0/1, PAR640/1, DEVSEL0/1# GNT0/1#, REQ640/1#, ACK640/1#,PAR0/1, PERR0/1#, CBE0/1[74:0]# **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Hold	0.5		ns	

Table 154: PCI Interface in Conventional PCI Mode at 33 MHz AC Timing (Continued)
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device only uses one PCI bus.

Signals	Description	Min.	Max.	Units	Load
FRAME0/1#, TRDY0/1#, IRDY0/1#, DEVSEL0/1#, PAD0/1[6331:0], STOP0/1#, CBE0/1[74:0]#, REQ640/1#, ACK640/1#, REQ0/1#, PAR0/1, PERR0/1#, SERR0/1#, PAR640/1 **NOTE:** The following signals **do not apply** to the MV64361 device: REQ640/1#, ACK640/1#, PAR640/1, PAD0/1[63:32], and CBE0/1[7:4]#.	Output Delay (Bused Signals)	2.0	11.0	ns	See Figure 86.
REQ0/1# *	Output Delay (Point-to-Point Signals)	2.0	12.0	ns	

32.8 MPP Interface

Notes

- All MPP pins setup, hold, and output delay times are referred to the SysClk rising edge, unless stated otherwise.
- PME0#, PME1#, WDE#, WDNMI# are asynchronous outputs.

Table 155: MPP Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
GPP[31:0]	Setup	2.7		ns	
GPP[31:0]	Hold	1.5		ns	
GPP[15:0]	Output Delay	0.5	6.0	ns	30pF
GPP[31:16]	Output Delay	0.5	5.2	ns	30pF
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter setup time relative to PCLK, 33MHz, DLL disabled.	Setup	10.0		ns	
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter hold time relative to PCLK, 33MHz, DLL disabled.	Hold	0.5		ns	
REQ0[5:0]#, REQ1[5:0]# **NOTE:** PCI-2.2 arbiter setup time relative to PCLK, 66MHz, DLL enabled.	Setup	5.0		ns	

Table 155: MPP Interface AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
REQ0[5:0]#, REQ1[5:0]#	Hold	0.3		ns	
NOTE: PCI-2.2 arbiter hold time relative to PCLK, 66MHz, DLL enabled.					
REQ0[5:0]#	Setup	1.7		ns	
NOTE: PCI-x arbiter setup time relative to PCLK, 133MHz, DLL enabled.					
REQ1[5:0]#	Setup	2.2		ns	
NOTE: PCI-x arbiter setup time relative to PCLK, 133MHz, DLL enabled.					
REQ0[5:0]#, REQ1[5:0]#	Hold	0.5		ns	
NOTE: PCI-x arbiter hold time relative to PCLK, 133MHz, DLL enabled.					
GNT0[5:0]#, GNT1[5:0]#	Output Delay	2.0	12.0	ns	See Figure 86.
NOTE: PCI-2.2 arbiter output delay relative to PCLK, 33MHz, DLL disabled.					
GNT0[5:0]#, GNT1[5:0]# *	Output Delay	1.9	6.0	ns	
NOTE: PCI-2.2 arbiter output delay relative to PCLK, 66MHz, DLL enabled.					
GNT0[5:0]#	Output Delay	0.7	4.2	ns	
NOTE: PCI-x arbiter output delay relative to PCLK, 125MHz, DLL enabled.					
GNT1[5:0]#	Output Delay	0.7	4.5	ns	
NOTE: PCI-x arbiter output delay relative to PCLK, 125MHz, DLL enabled.					
DMAReq[3:0]#	Setup	2.4		ns	
DMAReq[3:0]#	Hold	1		ns	
DMAAck[3:0]	Output Delay Sampled	1	5.5	ns	30pF
DMAAck[3:0]	Output Delay Not sampled	1	7.7	ns	30pF
EOT[3:0]	Setup	2.4		ns	
EOT[3:0]	Hold	1		ns	
TCEn[3:0]#	Setup	2.4		ns	
TCEn[3:0]#	Hold	1		ns	

Table 155: MPP Interface AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
TCTcnt[3:0]#	Output Delay	1	5.7	ns	30pF
DBurst#	Output Delay Not sampled	1	7.5	ns	30pF
InitAct output delay	Output Delay	1	5.7	ns	30pF
SCLK	Frequency		50	MHz	
SCLK	Cycle Time	20		ns	
TSCLK	Frequency		50	MHz	
TSCLK	Cycle Time	20		ns	
TxD0/1	Output Delay	5	12.9	ns	30pF
RxD0/1	Setup	5.5		ns	
RxD0/1	Hold	2		ns	
RTS0/1 **NOTE:** TxD0/1, RxD0/1, and RTS0/1 are relative to TSCLK	Output Delay	5	12.5	ns	30pF
CTS0/1	Setup	2.9		ns	
CTS0/1	Hold	1.5		ns	
CD0/1	Setup	1		ns	
CD0/1 **NOTE:** CTS0/1, CD0/1 are relative to TSCLK	Hold	2		ns	

32.9 Ethernet Interface AC Timing

Table 156: Ethernet Interface AC Timing

NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device has only one Ethernet port.

Signals	Description	Min.	Max.	Units	Load
Ethernet PCS Interface **NOTE:** Rx[9:0] hold and setup times are referred to the RxClk[1:0] rising and falling edges. Tx[9:0] Output Delays, are referred to the TxClkOut rising edge.					
RxClk0/1[1:0] frequency	Frequency		62.5	MHz	
TxClkOut0/1 frequency	Frequency		125	MHz	
Rx0/1[9:0]	Setup	2		ns	
Rx0/1[9:0]	Hold	1		ns	

Table 156: Ethernet Interface AC Timing (Continued)
NOTE: The "0/1" extension to the PCI label only applies to the MV64360 and MV64361 devices. The MV64362 device has only one Ethernet port.

Signals	Description	Min.	Max.	Units	Load
Tx0/1[9:0]	Output Delay	1.4	5.5	ns	See Figure 87
Ethernet GMII Interface **NOTE:** All input pins Setup, and Hold times are referred to the RxClk rising edge. All output pins output delays, are referred to the TxClkOut rising edge.					
RxClk0/1	Frequency		125	MHz	
TxClkOut0/1	Frequency		125	MHz	
RxD0/1[7:0], RxDv0/1, RxErr0/1	Setup	2		ns	
RxD0/1[7:0], RxDv0/1, RxErr0/1	Hold	0.4		ns	
TxD0/1[7:0], TxEn0/1, TxErr0/1	Output Delay	0.5	5.5	ns	See Figure 87
Ethernet MII Interface **NOTE:** All input pins Setup, and Hold times are referred to the RxClk rising edge. All output pins output delays, are referred to the TxClk rising edge.					
RxClk0/1	Frequency		25	MHz	
TxClk0/1	Frequency		25	MHz	
RxD0/1[3:0], RxDv0/1, RxErr0/1	Setup	2		ns	
RxD0/1[3:0]	Hold	1		ns	
RxDv0/1, RxErr0/1	Hold	1		ns	
TxD0/1[3:0], TxEn0/1, TxErr0/1	Output Delay	2	11	ns	See Figure 87

32.10TWSI Interface AC Timing

Note

SDA setup, hold, and output delay times are referred to SCK rising edge.

Table 157: TWSI Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
SCK	Frequency		400	KHz	

Table 157: TWSI Interface AC Timing (Continued)

Signals	Description	Min.	Max.	Units	Load
SCK	Clock Cycle	2.5		us	
SDA	Setup	4		ns	
SDA	Hold	1		ns	
SDA	Output Delay	1	6	ns	20pf

32.11JTAG Interface AC Timing

Note

JTDI and JTMS setup and hold times, and JTDO output delay, are referred to JTCLK rising edge.

Table 158: JTAG Interface AC Timing

Signals	Description	Min.	Max.	Units	Load
JTCK	Frequency		1	MHz	
JTCK	Clock Cycle	1000		ns	
JTRST#	Active Time	1		ms	
JTDI, JTMS	Boundary scan data setup	75		ns	
JTDI, JTMS	Boundary scan data h old	10		ns	
JTDO	Boundary scan data output delay	1	75	ns	

Figure 84: CPU Interface Output Delay Test Load

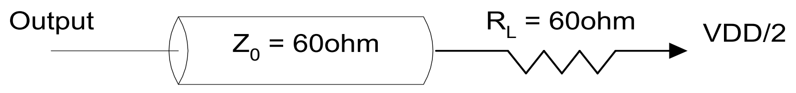
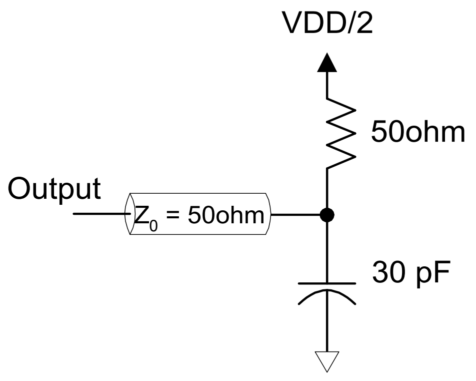


Figure 85: DRAM SSTL I/O Pads Output Delay Test Load



Note

The DRAM load applies to both CLK/CLK# and to DRAM address and control outputs.

Figure 86: PCI I/O Pads Output Delay Test Load

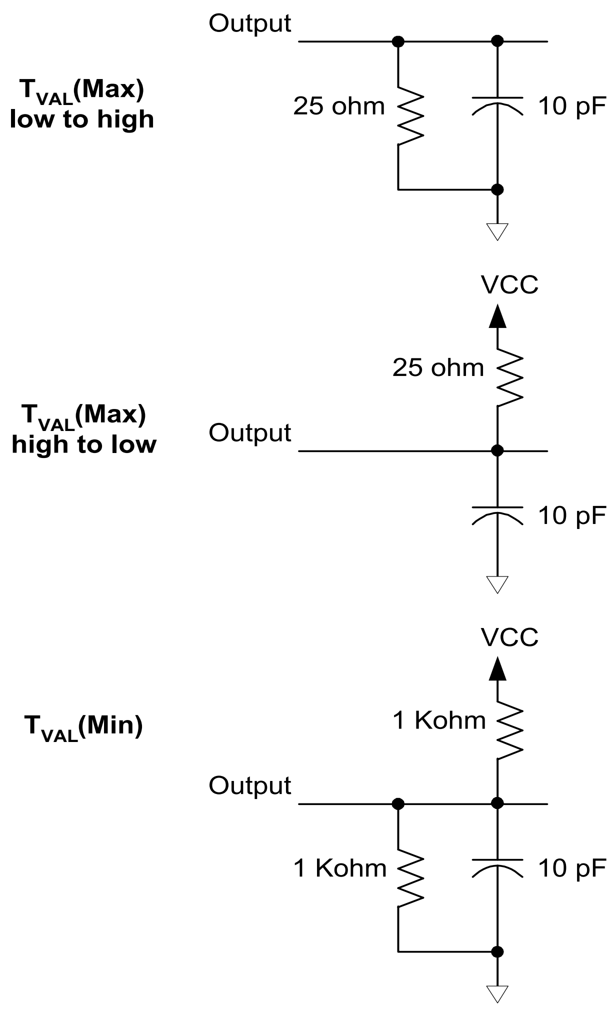
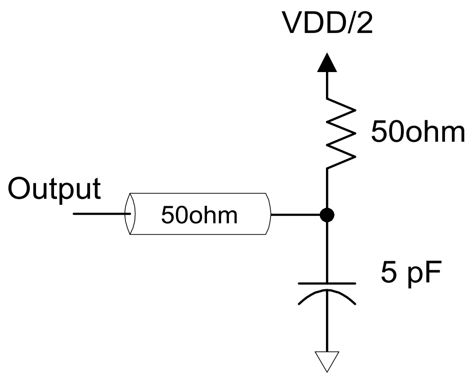


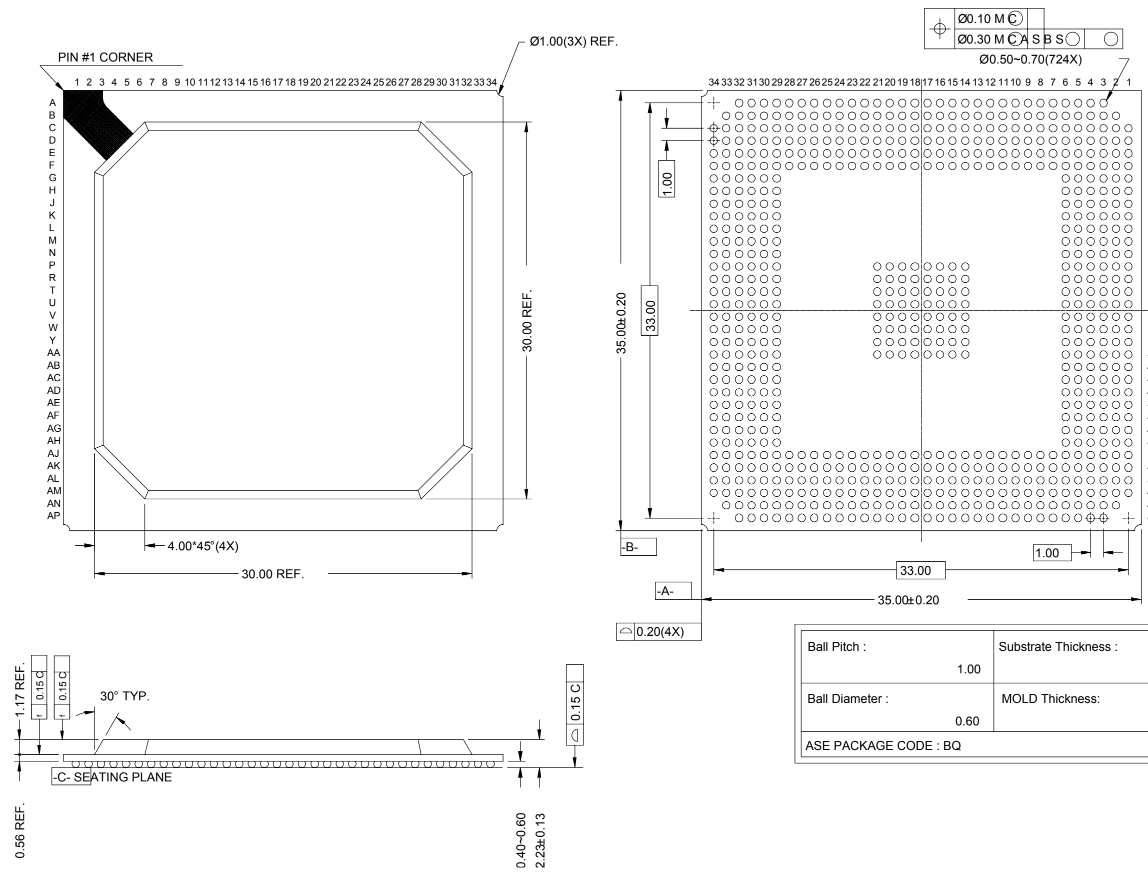
Figure 87: GMII Output Delay Test Load



Note

GMII test load applies to both Tx clock and to Tx data.

Section 33. 724 BGA Package Mechanical Information

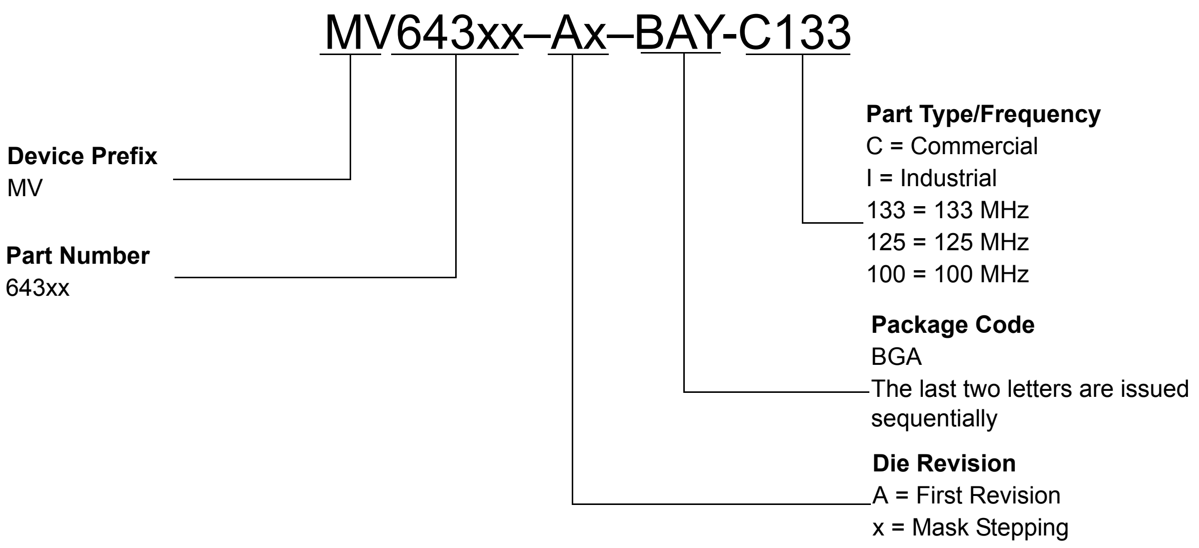


Ball Pitch : 1.00	Substrate Thickness : 0.56
Ball Diameter : 0.60	MOLD Thickness: 1.17
ASE PACKAGE CODE : BQ	

Section 34. MV64360/1/2 Part Numbering

Figure 88 shows the part order numbering scheme for the MV64360/1/2 parts. Refer to Marvell FAEs or representatives for complete information when you order parts and need to specify the correct die revision.

Figure 88: Sample Part Number



The part numbers for the MV64360/1/2 are as follows:

Commercial

- MV64360-A2-BAY-C133
- MV64361-A2-BAY-C133
- MV64362-A2-BAY-C133

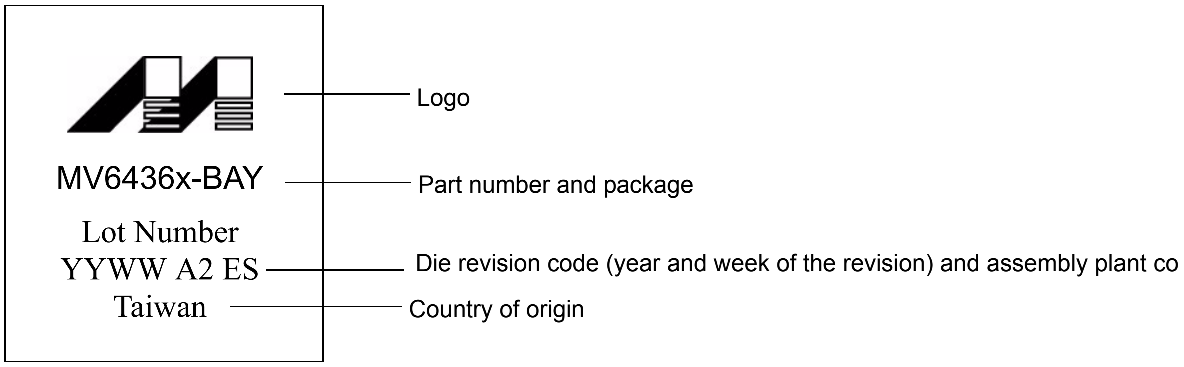
Industrial

- MV64360-A2-BAY-I100
- MV64361-A2-BAY-I100
- MV64362-A2-BAY-I100
- MV64360-A2-BAY-I125
- MV64361-A2-BAY-I125
- MV64362-A2-BAY-I125

These are the only valid part numbers that can be used when ordering the MV64360/1/2.

Figure 89 shows the part marking.

Figure 89: Part Marking



MV64360/1/2 Register Set

List of Registers

Appendix A. CPU Interface Registers (Continued)

Appendix A. CPU Interface Registers (Continued)

Appendix A. CPU Interface Registers (Continued)

Appendix A. CPU Interface Registers (Continued)

Appendix E. PCI Interface Registers (Continued)

Appendix E. PCI Interface Registers (Continued)

Appendix E. PCI Interface Registers (Continued)

Appendix E. PCI Interface Registers (Continued)

Appendix E. PCI Interface Registers (Continued)

Appendix E. PCI Interface Registers (Continued)

Appendix G. Gigabit Ethernet Controller Interface Registers (Continued)

Appendix G. Gigabit Ethernet Controller Interface Registers (Continued)

CONFIDENTIAL

Appendix A. CPU Interface Registers

A.1 Programing the CPU Configuration Register

The CPU setting of the CPU Configuration register requires special care, since it affects the MV64360/1/2 behavior on consecutive CPU accesses.

To change the register, the following steps are recommended:

1. Read the CPU Configuration register. This guarantees that all previous transactions in the CPU interface pipe are flushed.
2. Program the register to its new value.
3. Read polling of the register until the new data is being read.

Note

The CPU Configuration register wakes up with transactions pipeline disable. It is recommended to change this default in order gain the maximum CPU interface performance.

Setting the CPU Configuration register must be done once. For example, if the CPU interface is configured to support out of order read completion, changing the register to not support OOO read completion is fatal.

A.2 CPU Interface Register Maps

Table 159: CPU Address Decode Register Map
NOTE: PCI_1, CPU1, and integrated SRAM registers are only valid for the MV64360 and MV64361 devices.

Register Name	Offset	Table Number and Page Number
CS[0]# Base Address	0x008	Table 165, p.443
CS[0]# Size	0x010	Table 166, p.443
CS[1]# Base Address	0x208	Table 167, p.443
CS[1]# Size	0x210	Table 168, p.443
CS[2]# Base Address	0x018	Table 169, p.444
CS[2]# Size	0x020	Table 170, p.444
CS[3]# Base Address	0x218	Table 171, p.444
CS[3]# Size	0x220	Table 172, p.444
DevCS[0]# Base Address	0x028	Table 173, p.444
DevCS[0]# Size	0x030	Table 174, p.445

Table 159: CPU Address Decode Register Map (Continued)
NOTE: PCI_1, CPU1, and integrated SRAM registers are only valid for the MV64360 and MV64361 devices.

Register Name	**Offset**	**Table Number and Page Number**
DevCS[1]# Base Address	0x228	Table 175, p.445
DevCS[1]# Size	0x230	Table 176, p.445
DevCS[2]# Base Address	0x248	Table 177, p.445
DevCS[2]# Size	0x250	Table 178, p.446
DevCS[3]# Base Address	0x038	Table 179, p.446
DevCS[3]# Size	0x040	Table 180, p.446
BootCS# Base Address	0x238	Table 181, p.446
BootCS# Size	0x240	Table 182, p.446
PCI_0 I/O Base Address	0x048	Table 183, p.447
PCI_0 I/O Size	0x050	Table 184, p.447
PCI_0 Memory 0 Base Address	0x058	Table 185, p.447
PCI_0 Memory 0 Size	0x060	Table 186, p.448
PCI_0 Memory 1 Base Address	0x080	Table 187, p.448
PCI_0 Memory 1 Size	0x088	Table 188, p.449
PCI_0 Memory 2 Base Address	0x258	Table 189, p.449
PCI_0 Memory 2 Size	0x260	Table 190, p.450
PCI_0 Memory 3 Base Address	0x280	Table 191, p.450
PCI_0 Memory 3 Size	0x288	Table 192, p.450
PCI_1 I/O Base Address	0x090	Table 193, p.451
PCI_1 I/O Size	0x098	Table 194, p.451
PCI_1 Memory 0 Base Address	0x0A0	Table 195, p.451
PCI_1 Memory 0 Size	0x0A8	Table 196, p.452
PCI_1 Memory 1 Base Address	0x0B0	Table 197, p.452
PCI_1 Memory 1 Size	0x0B8	Table 198, p.453
PCI_1 Memory 2 Base Address	0x2A0	Table 199, p.453
PCI_1 Memory 2 Size	0x2A8	Table 200, p.453
PCI_1 Memory 3 Base Address	0x2B0	Table 201, p.454
PCI_1 Memory 3 Size	0x2B8	Table 202, p.454

Table 159: CPU Address Decode Register Map (Continued)
NOTE: PCI_1, CPU1, and integrated SRAM registers are only valid for the MV64360 and MV64361 devices.

Register Name	Offset	Table Number and Page Number
Integrated SRAM Base Address	0x268	Table 203, p.454
Internal Space Base Address	0x068	Table 204, p.455
Base Address Enable	0x278	Table 205, p.455
PCI_0 I/O Address Remap	0x0F0	Table 206, p.455
PCI_0 Memory 0 Address Remap (Low)	0x0F8	Table 207, p.456
PCI_0 Memory 0 Address Remap (High)	0x320	Table 208, p.456
PCI_0 Memory 1 Address Remap (Low)	0x100	Table 209, p.456
PCI_0 Memory 1 Address Remap (High)	0x328	Table 210, p.456
PCI_0 Memory 2 Address Remap (Low)	0x2f8	Table 211, p.457
PCI_0 Memory 2 Address Remap (High)	0x330	Table 212, p.457
PCI_0 Memory 3 Address Remap (Low)	0x300	Table 213, p.457
PCI_0 Memory 3 Address Remap (High)	0x338	Table 214, p.457
PCI_1 I/O Address Remap	0x108	Table 215, p.457
PCI_1 Memory 0 Address Remap (Low)	0x110	Table 216, p.458
PCI_1 Memory 0 Address Remap (High)	0x340	Table 217, p.458
PCI_1 Memory 1 Address Remap (Low)	0x118	Table 218, p.458
PCI_1 Memory 1 Address Remap (High)	0x348	Table 219, p.458
PCI_1 Memory 2 Address Remap (Low)	0x310	Table 220, p.459
PCI_1 Memory 2 Address Remap (High)	0x350	Table 221, p.459
PCI_1 Memory 3 Address Remap (Low)	0x318	Table 222, p.459
PCI_1 Memory 3 Address Remap (High)	0x358	Table 223, p.459
CPU/PCI_0 Headers Retarget Control	0x3B0	Table 224, p.460
CPU/PCI_0 Header Retarget Base	0x3B8	Table 225, p.460
CPU/PCI_1 Headers Retarget Control	0x3C0	Table 226, p.461
CPU/PCI_1 Headers Retarget Base	0x3C8	Table 227, p.461
CPU/GE Headers Retarget Control	0x3D0	Table 228, p.461
CPU/GE Headers Retarget Base	0x3D8	Table 229, p.461
CPU/IDMA Headers Retarget Control	0x3E0	Table 230, p.461

Table 159: CPU Address Decode Register Map (Continued)
NOTE: PCI_1, CPU1, and integrated SRAM registers are only valid for the MV64360 and MV64361 devices.

Register Name	Offset	Table Number and Page Number
CPU/IDMA Headers Retarget Base	0x3E8	Table 231, p.462

Table 160: CPU Control Register Map

Register	Offset	Page
CPU Configuration	0x000	Table 232, p.462
CPU Mode	0x120	Table 233, p.464
CPU Pads Calibration	0x3B4	Table 234, p.464
Reset Sample (Low)	0x3C4	Table 235, p.465
Reset Sample (High)	0x3D4	Table 236, p.468
CPU Master Control	0x160	Table 237, p.470
CPU Interface Crossbar Control (Low)	0x150	Table 238, p.471
CPU Interface Crossbar Control (High)	0x158	Table 239, p.472
CPU Interface Crossbar Time Out	0x168	Table 240, p.472

Table 161: SMP Register Map
NOTE: CPU1 registers are only valid for the MV64360 and MV64361 devices.

Register	Offset	Page
Who Am I	0x200	Table 241, p.473
CPU0 Doorbell	0x214	Table 242, p.473
CPU0 Doorbell Clear	0x21C	Table 243, p.473
CPU1 Doorbell	0x224	Table 244, p.474
CPU1 Doorbell Clear	0x22c	Table 245, p.474
CPU0 Doorbell Mask	0x234	Table 246, p.474
CPU1 Doorbell Mask	0x23C	Table 247, p.474
Semaphore0	0x244	Table 248, p.475
Semaphore1	0x24C	Table 249, p.475
Semaphoer2	0x254	Table 250, p.475
Semaphore3	0x25C	Table 251, p.475
Semaphore4	0x264	Table 252, p.476
Semaphore5	0x26C	Table 253, p.476

Table 161: SMP Register Map (Continued)
NOTE: CPU1 registers are only valid for the MV64360 and MV64361 devices.

Register	Offset	Page
Semaphore6	0x274	Table 254, p.476
Semaphore7	0x27C	Table 255, p.476

Table 162: CPU Sync Barrier Register Map
NOTE: CPU1 registers are only valid for the MV64360 and MV64361 devices.

Register	Offset	Page
CPU0 Sync Barrier Trigger	0x0C0	Table 256, p.477
CPU0 Sync Barrier Virtual	0x0C8	Table 257, p.477
CPU1 Sync Barrier Trigger	0x0D0	Table 258, p.477
CPU1 Sync Barrier Virtual	0x0D8	Table 259, p.477

Table 163: CPU Access Protection Register Map

Register	Offset	Page
CPU Protect Window 0 Base Address	0x180	Table 260, p.478
CPU Protect Window 0 Size	0x188	Table 261, p.478
CPU Protect Window 1 Base Address	0x190	Table 262, p.478
CPU Protect Window 1 Size	0x198	Table 263, p.479
CPU Protect Window 2 Base Address	0x1A0	Table 264, p.479
CPU Protect Window 2 Size	0x1A8	Table 265, p.479
CPU Protect Window 3 Base Address	0x1B0	Table 266, p.479
CPU Protect Window 3 Size	0x1B8	Table 267, p.479

Table 164: CPU Error Report Register Map

Register	Offset	Page
CPU Error Address (Low)	0x070	Table 268, p.480
CPU Error Address (High)	0x078	Table 269, p.480
CPU Error Data (Low)	0x128	Table 270, p.481
CPU Error Data (High)	0x130	Table 271, p.481
CPU Error Parity	0x138	Table 272, p.481
CPU Error Cause	0x140	Table 273, p.481
CPU0 Error Mask	0x148	Table 274, p.482

A.3 CPU Address Decode Registers

Table 165: CS[0]# Base Address
Offset: 0x008

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x0	CS[0] Base Address. Corresponds to CPU address bits[35:16][1].
31:20	Reserved	RES 0x0	Reserved.

1. Address bits[35:32] take part in the address decoding process when using the MPC7450 in extended address mode. Otherwise, Base address bits[19:16] (that corresponds to CPU address bits[35:32]) must be 0.

Table 166: CS[0]# Size
Offset: 0x010

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x7F	CS[0]# Bank Size. Corresponds to Base Address bits[15:0]. Must be programed from LSB to MSB as sequence of ‘1’s followed by sequence of ‘0’s.
31:16	Reserved	RES 0x0	Reserved.

Table 167: CS[1]# Base Address
Offset: 0x208

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x00080	CS[1] Base Address
31:20	Reserved	RES 0x0	Reserved.

Table 168: CS[1]# Size
Offset: 0x210

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x7F	CS[1] Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 169: CS[2]# Base Address
Offset: 0x018

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x00100	CS[2] Base Address
31:20	Reserved	RES 0x0	Reserved.

Table 170: CS[2]# Size
Offset: 0x020

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x7F	CS[2] Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 171: CS[3]# Base Address
Offset: 0x218

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x00180	CS[3] Base Address
31:20	Reserved	RES 0x0	Reserved.

Table 172: CS[3]# Size
Offset: 0x220

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x7F	CS[3] Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 173: DevCS[0]# Base Address
Offset: 0x028

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x01C00	CS[0] Base Address

Table 173: DevCS[0]# Base Address
Offset: 0x028

Bits	Field	Type/ Init Val	Description
31:20	Reserved	RES 0x0	Reserved.

Table 174: DevCS[0]# Size
Offset: 0x030

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x7F	CS[0] Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 175: DevCS[1]# Base Address
Offset: 0x228

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x01C80	CS[1] Base Address
31:20	Reserved	RES 0x0	Reserved.

Table 176: DevCS[1]# Size
Offset: 0x230

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x7F	CS[1] Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 177: DevCS[2]# Base Address
Offset: 0x248

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x01D00	CS[2] Base Address
31:20	Reserved	RES 0x0	Reserved.

**Table 178: DevCS[2]# Size
Offset: 0x250**

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0xFF	CS[2] Bank Size
31:16	Reserved	RES 0x0	Reserved.

**Table 179: DevCS[3]# Base Address
Offset: 0x038**

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x0FF00	CS[3] Base Address
31:20	Reserved	RES 0x0	Reserved.

**Table 180: DevCS[3]# Size
Offset: 0x040**

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x7F	CS[3] Bank Size
31:16	Reserved	RES 0x0	Reserved.

**Table 181: BootCS# Base Address
Offset: 0x238**

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x0FF80	CS[3] Base Address
31:20	Reserved	RES 0x0	Reserved.

**Table 182: BootCS# Size
Offset: 0x240**

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x7F	BootCS Bank Size

Table 182: BootCS# Size
Offset: 0x240

Bits	Field	Type/ Init Val	Description
31:16	Reserved	RES 0x0	Reserved.

Table 183: PCI_0 I/O Base Address
Offset: 0x048

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x01000	PCI_0 I/O Space Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RW 0x1	PCI Master Data Swap Control 00 = Byte Swap 01 = No swapping 10 = Both byte and word swap 11 = Word swap
31:26	Reserved	RES 0x0	Reserved.

Table 184: PCI_0 I/O Size
Offset: 0x050

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_0 I/O Space Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 185: PCI_0 Memory 0 Base Address
Offset: 0x058

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x01200	PCI_0 Memory 0 Base Address
23:20	Reserved	RES 0x0	Reserved.

Table 185: PCI_0 Memory 0 Base Address (Continued)
Offset: 0x058

Bits	Field	Type/ Init Val	Description
25:24	PCISwap	RW 0x1	PCI master data swap control 00 = Byte Swap 01 = No swapping 10 = Both byte and word swap 11 = Word swap
26	NS	RW 0x0	PCI Master No Snoop Attribute Control **NOTE:** Only relevant in PCI-X mode. 0 - PCI master does not assert NS attribute 1 - PCI master asserts NS attribute
27	PCIReq64[1]	RW 0x0	**NOTE:** Only valid for the MV64360 and MV64362. This bit is reserved in the MV64361 device. PCI master REQ64# policy 0 = Asserts REQ64# only when transaction is longer than 64-bits. 1 = Always assert REQ64#. **NOTE:** When working in PCI-X mode, this bit must be reset.
31:28	Reserved	RES 0x0	Reserved.

1. Relevant only when configured to 64-bit PCI bus.

Table 186: PCI_0 Memory 0 Size
Offset: 0x060

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_0 Memory 0 Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 187: PCI_0 Memory 1 Base Address
Offset: 0x080

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x0F200	PCI_0 Memory 1 Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RW 0x1	Same as PCI_0 Memory 0 Base Address.

**Table 187: PCI_0 Memory 1 Base Address
Offset: 0x080 (Continued)**

Bits	Field	Type/ Init Val	Description
26	NS	RW 0x0	Same as PCI_0 Memory 0 Base Address.
27	PCIReq64	RW 0x0	Same as PCI_0 Memory 0 Base Address. **NOTE:** Only valid for the MV64360 and MV64362. This bit is reserved in the MV64361 device.
31:28	Reserved	RES 0x0	Reserved.

**Table 188: PCI_0 Memory 1 Size
Offset: 0x088**

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_0 Memory 1 Bank Size
31:16	Reserved	RES 0x0	Reserved.

**Table 189: PCI_0 Memory 2 Base Address
Offset: 0x258**

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x0F400	PCI_0 Memory 2 Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RW 0x1	Same as PCI_0 Memory 0 Base Address.
16	NS	RW 0x0	Same as PCI_0 Memory 0 Base Address.
27	PCIReq64	RW 0x0	Same as PCI_0 Memory 0 Base Address. **NOTE:** Only valid for the MV64360 and MV64362. This bit is reserved in the MV64361 device.
31:28	Reserved	RES 0x0	Reserved.

Table 190: PCI_0 Memory 2 Size
Offset: 0x260

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_0 Memory 2 Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 191: PCI_0 Memory 3 Base Address
Offset: 0x280

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x0F600	PCI_0 Memory 3 Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RW 0x1	Same as PCI_0 Memory 0 Base Address.
26	NS	RW 0x0	Same as PCI_0 Memory 0 Base Address.
27	PCIReq64	RW 0x0	Same as PCI_0 Memory 0 Base Address. **NOTE:** Only valid for the MV64360 and MV64362. This bit is reserved in the MV64361 device.
31:28	Reserved	RES 0x0	Reserved.

Table 192: PCI_0 Memory 3 Size
Offset: 0x288

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_0 Memory 3 Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 193: PCI_1 I/O Base Address
Offset: 0x090

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x02000	PCI_1 I/O Space Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RW 0x1	Same as PCI_0 Memory 0 Base Address.
31:26	Reserved	RES 0x0	Reserved.

Table 194: PCI_1 I/O Size
Offset: 0x098

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_1 I/O Space Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 195: PCI_1 Memory 0 Base Address
Offset: 0x0A0

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x02200	PCI_1 Memory 0 Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RW 0x1	Same as PCI_0 Memory 0 Base Address.
26	NS	RW 0x0	Same as PCI_0 Memory 0 Base Address.
27	PCIReq64	RW 0x0	Same as PCI_0 Memory 0 Base Address. **NOTE:** Only valid for the MV64360. This bit is reserved in the MV64361 device.

**Table 195: PCI_1 Memory 0 Base Address (Continued)
Offset: 0x0A0**

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
31:28	Reserved	RES 0x0	Reserved.

**Table 196: PCI_1 Memory 0 Size
Offset: 0x0A8**

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_1 Memory 0 Bank Size
31:16	Reserved	RES 0x0	Reserved.

**Table 197: PCI_1 Memory 1 Base Address
Offset: 0x0B0**

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x02400	PCI_1 Memory 1 Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RW 0x1	Same as PCI_0 Memory 0 Base Address.
26	NS	RW 0x0	Same as PCI_0 Memory 0 Base Address.
27	PCIReq64	RW 0x0	Same as PCI_0 Memory 0 Base Address. **NOTE:** Only valid for the MV64360. This bit is reserved in the MV64361 device.
31:28	Reserved	RES 0x0	Reserved.

Table 198: PCI_1 Memory 1 Size
Offset: 0x0B8

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_1 Memory 1 Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 199: PCI_1 Memory 2 Base Address
Offset: 0x2A0

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x02600	PCI_1 Memory 2 Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RW 0x1	Same as PCI_0 Memory 0 Base Address.
26	NS	RW 0x0	Same as PCI_0 Memory 0 Base Address.
27	PCIReq64	RW 0x0	Same as PCI_0 Memory 0 Base Address. **NOTE:** Only valid for the MV64360. This bit is reserved in the MV64361 device.
31:28	Reserved	RES 0x0	Reserved.

Table 200: PCI_1 Memory 2 Size
Offset: 0x2A8

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_1 Memory 2 Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 201: PCI_1 Memory 3 Base Address
Offset: 0x2B0

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x02800	PCI_1 Memory 3 Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RW 0x1	Same as PCI_0 Memory 0 Base Address.
26	NS	RW 0x0	Same as PCI_0 Memory 0 Base Address.
27	PCIReq64	RW 0x0	Same as PCI_0 Memory 0 Base Address. **NOTE:** Only valid for the MV64360. This bit is reserved in the MV64361 device.
31:28	Reserved	RES 0x0	Reserved.

Table 202: PCI_1 Memory 3 Size
Offset: 0x2B8

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x1FF	PCI_1 Memory 3 Bank Size
31:16	Reserved	RES 0x0	Reserved.

Table 203: Integrated SRAM Base Address
Offset: 0x268

NOTE: Only applies to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
19:0	Base[1]	RW 0x04200	Integrated SRAM Base Address
31:20	Reserved	RES 0x0	Reserved.

1. Bits[1:0] are read only ‘0’. The integrated SRAM Base must be 256 KB aligned

Table 204: Internal Space Base Address
Offset: 0x068

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x01400	MV64360/1/2 Internal Space Base Address
23:20	Reserved	RES 0x0	Reserved.
25:24	PCISwap	RES 0x1	Same as PCI_0 Memory 0 Base Address. Relevant only for PCI master configuration transactions on the PCI bus. **NOTE:** Reserved for Marvell Technology usage.
31:26	Reserved	RES 0x0	Reserved.

Table 205: Base Address Enable
Offset: 0x278

Bits	Field	Type/ Init Val	Description

20:0	En	RW
		0x0
			Base address enable.
			Bit per each address window.
			If set to '0', the corresponding window is enabled.
			En[0] = CS[0]
			En[1] = CS[1]
			En[2] = CS[2]
			En[3] = CS[3]
			En[4] = DevCS[0]
			En[5] = DevCS[1]
			En[6] = DevCS[2]
			En[7] = DevCS[3]
			En[8] = BootCS
			En[9] = PCI_0 I/O
			En[10] = PCI_0 Mem0
			En[11] = PCI_0 Mem1
			En[12] = PCI_0 Mem2
			En[13] = PCI_0 Mem3
			En[14] = PCI_1 I/O
			En[15] = PCI_1 Mem0
			En[16] = PCI_1 Mem1
			En[17] = PCI_1 Mem2
			En[18] = PCI_1 Mem3
			En[19] = Integrated SRAM
			En[20] = Internal Space
			NOTE: En[19:14] only apply to the MV64360 and MV64361. These settings are reserved in the MV64362.

31:21	Reserved	RES 0x0	Reserved.

Table 206: PCI_0 I/O Address Remap
Offset: 0x0F0

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0x1000	PCI_0 I/O Space Address Remap

Table 206: PCI_0 I/O Address Remap
Offset: 0x0F0

Bits	Field	Type/ Init Val	Description
31:16	Reserved	RES 0x0	Reserved.

Table 207: PCI_0 Memory 0 Address Remap (Low)
Offset: 0x0F8

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0x1200	PCI_0 Memory 0 Address Remap (low 32 bits)
31:16	Reserved	RES 0x0	Reserved.

Table 208: PCI_0 Memory 0 Address Remap (High)
Offset: 0x320

Bits	Field	Type/ Init Val	Description
31:0	Remap	RW 0x0	PCI_0 Memory 0 Address Remap (high 32 bits)

Table 209: PCI_0 Memory 1 Address Remap (Low)
Offset: 0x100

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0xF200	PCI_0 Memory 1 Address Remap (low 32 bits)
31:16	Reserved	RES 0x0	Reserved.

Table 210: PCI_0 Memory 1 Address Remap (High)
Offset: 0x328

Bits	Field	Type/ Init Val	Description
31:0	Remap	RES 0x0	PCI_0 Memory 1 Address Remap (high 32 bits)

Table 211: PCI_0 Memory 2 Address Remap (Low)
Offset: 0x2f8

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0xF400	PCI_0 Memory 0 Address Remap (low 32 bits)
31:16	Reserved	RES 0x0	Reserved.

Table 212: PCI_0 Memory 2 Address Remap (High)
Offset: 0x330

Bits	Field	Type/ Init Val	Description
31:0	Remap	RW 0x0	PCI_0 Memory 2 Address Remap (high 32 bits)

Table 213: PCI_0 Memory 3 Address Remap (Low)
Offset: 0x300

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0xF600	PCI_0 Memory 1 Address Remap (low 32 bits)
31:16	Reserved	RES 0x0	Reserved.

Table 214: PCI_0 Memory 3 Address Remap (High)
Offset: 0x338

Bits	Field	Type/ Init Val	Description
31:0	Remap	RW 0x0	PCI_0 Memory 3 Address Remap (high 32 bits)

Table 215: PCI_1 I/O Address Remap
Offset: 0x108

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0x2000	PCI_1 I/O Space Address Remap
31:16	Reserved	RES 0x0	Reserved.

Table 216: PCI_1 Memory 0 Address Remap (Low)
Offset: 0x110

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0x2200	PCI_1 Memory 0 Address Remap (low 32 bits)
31:16	Reserved	RES 0x0	Reserved.

Table 217: PCI_1 Memory 0 Address Remap (High)
Offset: 0x340

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
31:0	Remap	RW 0x0	PCI_1 Memory 0 Address Remap (high 32 bits)

Table 218: PCI_1 Memory 1 Address Remap (Low)
Offset: 0x118

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0x2400	PCI_1 Memory 1 Address Remap (low 32 bits)
31:16	Reserved	RES 0x0	Reserved.

Table 219: PCI_1 Memory 1 Address Remap (High)
Offset: 0x348

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
31:0	Remap	RW 0x0	PCI_1 Memory 1 Address Remap (high 32 bits)

**Table 220: PCI_1 Memory 2 Address Remap (Low)
Offset: 0x310**

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0x2600	PCI_1 Memory 2 Address Remap (low 32 bits)
31:16	Reserved	RES 0x0	Reserved.

**Table 221: PCI_1 Memory 2 Address Remap (High)
Offset: 0x350**

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
31:0	Remap	RW 0x0	PCI_1 Memory 2 Address Remap (high 32 bits)

**Table 222: PCI_1 Memory 3 Address Remap (Low)
Offset: 0x318**

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
15:0	Remap	RW 0x2800	PCI_1 Memory 3 Address Remap (low 32 bits)
31:16	Reserved	RES 0x0	Reserved.

**Table 223: PCI_1 Memory 3 Address Remap (High)
Offset: 0x358**

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
31:0	Remap	RW 0x0	PCI_1 Memory 3 Address Remap (high 32 bits)

Table 224: CPU/PCI_0 Headers Retarget Control
Offset: 0x3B0

Bits	Field	Type/ Init Val	Description
0	En	RW 0x0	Headers retarget enable bit 0x0 = Disable 0x1 = Enable
3:1	BSize	RW 0x0	Buffer Size 0x0 = 256 bytes 0x1 = 512 bytes 0x2 = 1 KB 0x3 = 2 KB 0x4 = 4 KB 0x5 = 8 KB 0x6 - 0x7= Reserved
15:4	Reserved	RES 0x0	Reserved.
31:16	Mask1	RW 0x0	Defines the total space of the buffers to be manipulated, in 64 KB granularity. Size must be set from LSB to MSB as a sequence of 1's, followed by sequence of 0's. For example, in order to retarget the headers of 1K buffers of 1 KB size, which means 1MB of buffers space, Mask1 should be set to 0x000f **NOTE:** The total address space of retargeted headers must not exceed integrated SRAM size (256 KB). The minimum buffers space to be manipulated is 64 KB

Table 225: CPU/PCI_0 Header Retarget Base
Offset: 0x3B8

Bits	Field	Type/ Init Val	Description
3:0	Remap	RW 0x0	Remap address. Defines the upper bits of headers space in SRAM.
7:4	Mask2	RW 0x0	Defines how many of the integrated SRAM address upper bits should be remapped. If for example, the headers space is 64 KB (which is one quarter of the total SRAM space), Mask2 can be set to b'0011, which means that the two MSB bits of SRAM address (address bits[17:16]) will be replaced with the corresponding Remap bits).
15:12	BaseH	RW 0x0	High Base address. Corresponds to address bits[35:32]
31:16	Base	RW 0x0	Base address. Retarget is executed if address matches Base. Corresponds to CPU address bits[31:16]

Table 226: CPU/PCI_1 Headers Retarget Control
Offset: 0x3C0

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as CPU/PCI0 Headers Retarget Control

Table 227: CPU/PCI_1 Headers Retarget Base
Offset: 0x3C8

NOTE: All "PCI_1" registers only apply to the MV64360 and MV64361 devices.

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as CPU/PCI_0 Headers Retarget Base

Table 228: CPU/GE Headers Retarget Control
Offset: 0x3D0

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as CPU/PCI_0 Headers Retarget Control

Table 229: CPU/GE Headers Retarget Base
Offset: 0x3D8

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as CPU/PCI_0 Headers Retarget Base

Table 230: CPU/IDMA Headers Retarget Control
Offset: 0x3E0

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as CPU/PCI_0 Headers Retarget Control

Table 231: CPU/IDMA Headers Retarget Base
Offset: 0x3E8

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as CPU/PCI0 Headers Retarget Base

A.4 CPU Control Registers

Table 232: CPU Configuration
Offset: 0x000

Bits	Field	Type/ Init Val	Description
7:0	NoMatchCnt	RW 0xFF	CPU Address Miss Counter
8	NoMatchCntEn	RW 0x0	CPU Address Miss Counter Enable **NOTE:** Relevant only if multi-GT is enabled. 0 = Disabled 1 = Enabled.
9	NoMatchCntExt	RW 0x0	CPU address miss counter MSB
10	Reserved	RES 0x0	Reserved.
11	SingleCPU	RW 0x0	0 = Dual CPU. ARTRY# or HIT# might terminate a data transaction 1 = Single CPU. ARTRY# is asserted only in response to address only transactions. **NOTE:** If CPU data intervention is enabled, must be set to 0x0, even if interfacing a single CPU.
12	Endianess	RW 0x0	CPU Bus Byte Orientation. Must be ’0’.
13	Pipeline	RW 0x0	Pipeline Enable 0 = Disabled. The MV64360/1/2 will not respond with AACK# to a new CPU transaction, before the previous transaction data phase completes. 1 = Enabled
16:14	Reserved	RES 0x0	Reserved.
17	Stop Retry	RW 0x0	Stop to retry transactions from PCI 0 = Keep Retry all PCI transactions targeted to MV64360/1/2. 1 = Stop PCI transactions retry. Relevant only if PCI Retry reset strapping is enabled.

Table 232: CPU Configuration (Continued)
Offset: 0x000

Bits	Field	Type/ Init Val	Description
18	MultiGTDec	RW Sampled at reset DevAD[9].	Multi-GT Address Decode 0 = Normal address decoding 1 = Multi-GT address decoding
19	DPValid	RW 0x0	CPU DP[0-7] Connection 0 = Not connected. CPU write data parity and 60x bus master read data parity is not checked. 1 = Connected
21:20	Reserved	RES 0x0	Reserved.
22	PErrProp	RW 0x0	Parity Error Propagation 0 = MV64360/1/2 always drives good parity on DP[0-7] during CPU reads. 1 = MV64360/1/2 drives bad parity on DP[0-7] in case the read response from the target interface comes with erroneous data indication (e.g. ECC error from SDRAM interface).
24:23	Reserved	RES 0x1	Reserved.
25	AACK Delay2	RW 0x1	AACK# earliest assertion following TS#. Useful for CPUs clocks ratios that requires “late” ARTRY# window 0 = AACK# earliest assertion is two cycles after TS# 1 = AACK earliest assertion is three cycles after TS# **NOTE:** Not supported in multi-MV mode. If multi-MV is enabled, it is impossible to interface CPUs in which the ARTRY* window is delayed. For example: If a system is using MPC7450 CPU, the CPU core clock ratio must be selected to be higher than or equal to 1:5.
26	APValid	RW 0x0	CPU AP[0-3] Connection 0 = Not connected. The CPU address parity is not checked. 1 = Connected
27	RemapWrDis	RW 0x0	Address Remap Registers Write Control 0 = Write to Low Address decode register. Results in writing of the corresponding Remap register. 1 = Write to Low Address decode register. No affect on the corresponding Remap register.
31:28	Reserved	RES 0x0	Reserved.

Table 233: CPU Mode
Offset: 0x120

Bits	Field	Type/ Init Val	Description
1:0	MultiMVID	RW Sampled at reset on DevAD [11:10].	Multi-MV ID Represents the ID to which the MV64360/1/2 responds to during a multi-MV address decoding period. Set during reset initialization. Read only.
2	MultiMV	RW Sampled at reset on DevAD[9].	Set during the reset initialization. Read only. 0 = Single MV configuration 1 = Multi-MV configuration
3	RetryEn	RW Sampled at reset DevAD[16].	Set during reset initialization. Read Only. 0 = Don’t Retry PCI transactions 1 = Retry PCI transactions
7:4	CPUType/Init Val	RW Sampled at reset on DevAD[7:6]	Read Only (reset and bonding configuration). 0x0-0x3 = Reserved 0x4 = 64-bit PowerPC CPU, 60x bus 0x5 = 64-bit PowerPC CPU, MPX bus 0x6-0xF = Reserved
31:8	Reserved	RES 0x0	Reserved.

Table 234: CPU Pads Calibration
Offset: 0x3B4

Bits	Field	Type/ Init Val	Description
4:0	DrvN	RW 0x14	Pad Nchannel Driving Strength **NOTE:** Only applicable when dynamic tune is disabled.
9:5	DrvP	RW 0x14	Pad Pchannel Driving Strength **NOTE:** Only applicable when dynamic tune is disabled.
15:10	Reserved	RES 0x0	Read Only
16	TuneEn	Sampled at reset on DevAD[8].	Set to ‘1’ enables the dynamic tuning of pad driving strength.
21:17	LockN	RO 0x0	When dynamic tune is enabled, represents the final locked value of the Nchannel Driving Strength. Read Only
26:22	LockP	RO 0x0	When dynamic tune is enabled, represents the final locked value of the Pchannel Driving Strength Read Only

Table 234: CPU Pads Calibration (Continued)
Offset: 0x3B4

Bits	Field	Type/ Init Val	Description
30:27	Reserved	RO 0x0	Read Only
31	WrEn	RW 0x0	Write Enable CPU Pads Calibration register 0 = Register is read only (except for bit[31]). 1 = Register is writable.

Table 235: Reset Sample (Low)
Offset: 0x3C4

Bits	Field	Type/ Init Val	Description
0	SInitEn	RO Sampled at reset on DevAD[0].	Serial ROM Initialization Enable 0 = Disabled 1 = Enabled
1	DRAMPadCal	RO Sampled at reset on DevAD[1].	DRAM Pads Calibration 0 = Disable 1 = Enable
3:2	SROMAdd	RO Sampled at reset on DevAD [3:2].	Serial ROM Address 00 = ROM address is 1010000 01 = ROM address is 1010001 10 = ROM address is 1010010 11 = ROM address is 1010011 **NOTE:** If not using serial ROM initialization (DevAD[0] sampled LOW), DevAD[3:2] can be left with no pullup/pulldown
4	Reserved	RO 0x0	Reserved.
5	DefIntSpc	RO Sampled at reset on DevAD[5].	Default Internal Space Base 0 = 0x1400.0000 1 = 0xF100.0000
7:6	CPUBusConfig	RO Sampled at reset on DevAD [7:6].	CPU Bus Configuration 00 = Reserved 01= 60x bus 10 = MPX bus 11 = Reserved
8	Int60xArb	RO Sampled at reset on DevAD[4].	Internal 60x bus Arbiter 0 = Disable 1 = Enable

**Table 235: Reset Sample (Low) (Continued)
Offset: 0x3C4**

Bits	Field	Type/ Init Val	Description
9	MultiGTEn	RO Sampled at reset on DevAD[9].	Multiple MV64360/1/2 Support 0 = Disable 1 = Enable
11:10	MultiMVID	RO Sampled at reset on DevAD [11:10].	Multi-MV64360/1/2 Address ID 00 = MV64360/1/2 responds to A[9-10] =’00’ 01 = MV64360/1/2 responds to A[9-10] =’01’ 10 = MV64360/1/2 responds to A[9-10] =’10’ 11 = MV64360/1/2 responds to A[9-10] =’11’ **NOTE:** A[9-10] refers to the MV64360/1/2 signals. If not using multi-GT mode (DevAD[9] sampled LOW), DevAD[11:10] can be left with no pullup/pulldown.
12	PCI0PadsCal	RO Sampled at reset on DevAD[12].	PCI_0 Pads Calibration 0 = Disable 1 = Enable
13	PCI1PadsCal	RO Sampled at reset on DevAD[13].	**NOTE:** Only valid for the MV64360 and MV64361. Reserved in the MV64362 device. PCI_1 Pads Calibration 0 = Disable 1 = Enable
15:14	BootCSWidth	RO Sampled at reset on DevAD [15:14].	BootCS# Device Width 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = Reserved
16	PCIRty	RO Sampled at reset on DevAD[16].	PCI Retry 0 = Disable 1 = Enable
17	Reserved	RO Sampled at reset on DevAD[17].	Must pull up.
18	SDRAMClkSel	RO Sampled at reset on DevAD[18].	DDR SDRAM Clock Select 0 = DRAM is running at a higher frequency than the core clock 1 = DRAM is running with core clock **NOTE:** If set to ’1’, the DRAM PLL is not used.

**Table 235: Reset Sample (Low) (Continued)
Offset: 0x3C4**

Bits	Field	Type/ Init Val	Description
19	SDRAMAddDel	RO Sampled at reset on DevAD[19].	DDR SDRAM Address/Control Delay 0 = DRAM address/control signals toggle on falling edge of DRAM clock 1 = DRAM address/control signals toggle on rising edge of DRAM clock **NOTE:** The DevAD[19] setting depends on board topology and DRAM load. Detailed recommendations will follow silicon testing.
21:20	SDRAM PipeSel	RO Sampled at reset on DevAD [21:20].	DDR SDRAM Control Path Pipeline Select 00 = Reserved 01 = Two pipe stages. (Up to 133 MHz) 10 = Reserved 11 = Three pipe stages. (Up to 183 MHz) **NOTE:** For DRAM frequency up to 133MHz, set to ’10’. This is the recommended setting when running the DRAM with the core clock. For DRAM frequency up to 183 MHz, set to ’11’.
22	SDRAMSynch	RO Sampled at reset on DevAD[22].	DDR SDRAM Read Data Synchronization Select 0 = Read data is synchronized to core clock 1 = Read data is synchronized to FBClkIn **NOTE:** The DevAD[24:22] setting depends on board topology and DRAM load. Detailed recommendations will follow silicon testing
23	SDRAMRdDel	RO Sampled at reset on DevAD[23].	DDR SDRAM Read Control Logic Delay 0 = Disabled 1 = Enabled The clock to the read control logic is delayed (using DFCDL). **NOTE:** The DevAD[24:22] setting depends on board topology and DRAM load. Detailed recommendations will follow silicon testing
24	SDRAMRd DataDel	RO Sampled at reset on DevAD[24].	DDR SDRAM Read Data Delay 0 = Disabled 1 = Enabled The read data sample clock is delayed (using DFCDL). **NOTE:** The DevAD[24:22] setting depends on board topology and DRAM load. Detailed recommendations will follow silicon testing
25	BbEP2En	RO Sampled at reset on DevAD[25].	**NOTE:** Only valid for the MV64360. Reserved in the MV64361 and MV64362 devices. GbE port2 Enable 0 = Disable 1 = Enable. **NOTE:** The third GbE port is multiplexed on the upper bits of PCI_1 bus (see Section 26. “Pins Multiplexing” on page 343).

Table 235: Reset Sample (Low) (Continued)
Offset: 0x3C4

Bits	Field	Type/ Init Val	Description
28:26	PCI1DllCon	RO Sampled at reset on DevAD [28:26].	**NOTE:** Only valid for the MV64360 and MV64361. Reserved in the MV64362 device. PCI_1 DLL control 000 = DLL disable 001 = Conventional PCI mode at 66MHz 101 = PCI-X mode at 133MHz 110 = PCI-X mode at 66MHz **NOTE:** If interfacing conventional 33MHz PCI (M66EN sampled LOW), DLL is disabled, and DevAD[28:26] can be left with no pullup/pull-down. DevAD[28:26] setting is preliminary and might be changed after silicon testing.
31:29	PCI0DllCon	RO Sampled at reset on DevAD [31:29].	PCI_0 DLL control 000 = DLL disable 001 = Conventional PCI mode at 66MHz 101 = PCI-X mode at 133MHz 110 = PCI-X mode at 66MHz **NOTE:** If interfacing conventional 33MHz PCI (M66EN sampled LOW), DLL is disabled, and DevAD[28:26] can be left with no pullup/pull-down. DevAD[31:29] setting is preliminary and might be changed after silicon testing.

Table 236: Reset Sample (High)
Offset: 0x3D4

Bits	Field	Type/ Init Val	Description
3:0	DRAMPLLN Div[3:0]	RO Sampled at reset on DevDP [3:0].	DRAM PLL N Divider [3:0] See 11.11 “DRAM Clocking” on page 144 on N[7:0] and M[5:0] settings. **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), the initial value of N[7:0] is 0xA0 and M[5:0] is 0x28.
5:4	DRAMPLLN Div[5:4]	RO Sampled at reset on DevWE [1:0].	DRAM PLL N Divider [5:4] See 11.11 “DRAM Clocking” on page 144 on N[7:0] and M[5:0] settings. **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), the initial value of N[7:0] is 0xA0 and M[5:0] is 0x28.
6	CPUPadCal	RO Sampled at reset on DevAD[8].	CPU Pads Calibration 0 = Disable 1 = Enable

Table 236: Reset Sample (High) (Continued)
Offset: 0x3D4

Bits	Field	Type/ Init Val	Description
8:7	DRAMPLLN Div[7:6]	RO Sampled at reset on DevWE [3:2].	DRAM PLL N Divider [7:6] See 11.11 “DRAM Clocking” on page 144 on N[7:0] and M[5:0] settings. **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), the initial value of N[7:0] is 0xA0 and M[5:0] is 0x28.
9	DRAMPLLNP	RO Sampled at reset on BAdr[0].	DRAM PLL NP **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), the initial value of NP is 0x0.
10	DRAMPLL HIKVCO	RO Sampled at reset on BAdr[1].	DRAM PLL HIKVCO **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), the initial value of HIKVCO is 0x1.
11	DRAMPLLPU	RO Sampled at reset on BAdr[2].	DRAM PLL PU 0 = PLL power down 1 = PLL power up (normal operation) **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), the initial value of PU is 0x0.
12	GB0Sel	RO Sampled at reset on TxD0[0].	GbE port0 GMII/TBI Select 0 = MII/GMII 1 = TBI
18:13	DRAMPLL MDiv	RO Sampled at reset on TxD0[6:1].	DRAM PLL M Divider See 11.11 “DRAM Clocking” on page 144 on N[7:0] and M[5:0] settings. **NOTE:** If running the DRAM with core clock (DevAD[18] sampled HIGH), the initial value of N[7:0] is 0xA0 and M[5:0] is 0x28.
19	GB1Sel	RO Sampled at reset on TxD1[0].	**NOTE:** Only valid for the MV64360 and MV64361. Reserved in the MV64362 device. GbE port1 GMII/TBI Select 0 = MII/GMII 1 = TBI
20	GB2Sel	RO Sampled at reset on TxD2[0].	**NOTE:** Only valid for the MV64360. Reserved in the MV64361 and MV64362 devices. GbE port2 GMII/TBI Select 0 = MII/GMII 1 = TBI
21	JTAGCalBy	RO Sampled at reset on TxD0[7].	JTAG Pad Calb Bypass 0 = Normal Operation 1 = Bypass pad calibration **NOTE:** When JTAG is working, set to ’1’.
31:22	Reserved	RO 0x0	Reserved

Table 237: CPU Master Control
Offset: 0x160

Bits	Field	Type/ Init Val	Description
7:0	Reserved	RES 0x35	Reserved.
8	IntArb	RW Sampled at reset on DevAD[4].	CPU Bus Internal Arbiter Enable 0 = Disabled. External arbiter is required. 1 = Enabled. Use the MV64360/1/2 CPU bus arbiter.
9	MaskBR1	RW 0x1	Mask CPU1 bus request 0 = MV64360/1/2 bus arbiter gives arbitration to both CPU0 and CPU1. 1 = MV64360/1/2 bus arbiter ignores BR1# and does not assert BG1#.
10	MWrTrig	RW 0x0	Master Write Transaction Trigger 0 = With first valid write data 1 = With last valid write data
11	MRdTrig	RW 0x1	Master Read Response Trigger 0 = With first valid read data 1 = With last valid read data **NOTE:** If using the integrated SRAM, must be set to ‘1’.
12	CleanBlock	RW 0x0	Clean Block Snoop Transaction Support 0 = CPU does not support clean block (603e,750) 1 = CPU supports clean block (604e,G4)
13	FlushBlock	RW 0x0	Flush Block Snoop Transaction Support 0 = CPU does not support flush block (603e,750) 1 = CPU supports flush block (604e,G4)
31:14	Reserved	RES 0x0	Reserved.

**Table 238: CPU Interface Crossbar Control (Low)
Offset: 0x150**

Bits	Field	Type/ Init Val	Description
3:0	Arb0	RW 0x0	Slice 0 of CPU Master “pizza” Arbiter 0x0 = SDRAM interface snoop request 0x1 = Reserved 0x2 = NULL request 0x3 = PCI_0 access 0x4 = PCI_1 access (MV64360 and MV64361 only.) 0x5 = Comm unit access 0x6 = IDMA access 0x7 = Ethernet controller access 0x8-0xF = Reserved
7:4	Arb1	RW 0x3	Slice 1 of CPU Master “pizza” Arbiter
11:8	Arb2	RW 0x4	Slice 2 of CPU Master “pizza” Arbiter
15:12	Arb3	RW 0x5	Slice 3 of CPU Master “pizza” Arbiter
19:16	Arb4	RW 0x6	Slice 4 of CPU Master “pizza” Arbiter
23:20	Arb5	RW 0x7	Slice 5 of CPU Master “pizza” Arbiter
27:24	Arb6	RW 0x2	Slice 6 of CPU Master “pizza” Arbiter
31:28	Arb7	RW 0x2	Slice 7 of CPU Master “pizza” Arbiter

Table 239: CPU Interface Crossbar Control (High)
Offset: 0x158

Bits	Field	Type/ Init Val	Description
3:0	Arb8	RW 0x0	Slice 8 of CPU Master “pizza” Arbiter
7:4	Arb9	RW 0x3	Slice 9 of CPU Master “pizza” Arbiter
11:8	Arb10	RW 0x4	Slice 10 of CPU Master “pizza” Arbiter
15:12	Arb11	RW 0x5	Slice 11 of CPU Master “pizza” Arbiter
19:16	Arb12	RW 0x6	Slice 12 of CPU Master “pizza” Arbiter
23:20	Arb13	RW 0x7	Slice 13 of CPU Master “pizza” Arbiter
27:24	Arb14	RW 0x2	Slice 14 of CPU Master “pizza” Arbiter
31:28	Arb15	RW 0x2	Slice 15 of CPU Master “pizza” Arbiter

Table 240: CPU Interface Crossbar Time Out
Offset: 0x168

Bits	Field	Type/ Init Val	Description
7:0	Timeout	RW 0xFF	Crossbar Arbiter Time Out Preset Value
15:8	Reserved	RES 0x0	Reserved.
16	TimeoutEn	RW 0x1	Crossbar Arbiter Timer Enable 0 = Enable 1 = Disable
31:17	Reserved	RES 0x0	Reserved.

A.5 SMP Registers

Note

For proper operation of the following registers, the MV64360/1/2 CPU bus arbiter must be used.

Table 241: Who Am I
Offset: 0x200

Bits	Field	Type/ Init Val	Description
1:0	ID	RO 0x0	Read Only virtual register. If read by CPU0, returns 0x0. If read by CPU1, return 0x1. If read by PCI agent, returns 0x2.
31:2	Reserved	RES 0x0	Reserved.

Table 242: CPU0 Doorbell
Offset: 0x214

Bits	Field	Type/ Init Val	Description
7:0	Doorbell	RW 0x0	Bit is set to ‘1’ upon a write value of ‘1’. Write 0 has no affect. **NOTE:** Bit is cleared only by writing to CPU0 Doorbell Clear register.
31:8	Reserved	RES 0x0	Reserved.

Table 243: CPU0 Doorbell Clear
Offset: 0x21C

Bits	Field	Type/ Init Val	Description
7:0	Clear	RW 0x0	Virtual register. CPU0 write a value of ‘1’ clears the corresponding bit in CPU0 Doorbell register. Write of ‘0’ has no affect. Write from CPU1 or some PCI agent has no affect. Read returns CPU0 Doorbell register value.
31:8	Reserved	RES 0x0	Reserved.

Table 244: CPU1 Doorbell
Offset: 0x224

NOTE: Only valid for the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Description
7:0	Doorbell	RW 0x0	Bit is set to '1' upon write a value of '1'. Write '0' has no affect. **NOTE:** Bit is cleared only by writing to CPU1 Doorbell Clear register.
31:8	Reserved	RES 0x0	Reserved.

Table 245: CPU1 Doorbell Clear
Offset: 0x22c

NOTE: Only valid for the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Description
7:0	Clear	RW 0x0	Virtual register. CPU1 write a value of 1 clears the corresponding bit in CPU1 Doorbell register. Write of 0 has no affect. Write from CPU0 or some PCI agent has no affect. Read returns CPU1 Doorbell register value.
31:8	Reserved	RES 0x0	Reserved.

Table 246: CPU0 Doorbell Mask
Offset: 0x234

Bits	Field	Type/ Init Val	Description
7:0	Mask	RW 0x0	Mask bit per doorbell cause bit. 0 = The corresponding doorbell interrupt is masked; 1 = Enabled.
31:8	Reserved	RES 0x0	Reserved.

Table 247: CPU1 Doorbell Mask
Offset: 0x23C

NOTE: Only valid for the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Description
7:0	Mask	RW 0x0	Same as CPU0 Doorbell Mask register
31:8	Reserved	RES 0x0	Reserved.

Table 248: Semaphore0
Offset: 0x244

Bits	Field	Type/ Init Val	Description
7:0	Semaphore	RW 0x0	Read only. Can be locked by CPU0, CPU1, or PCI agent. The first one to read it before it is locked, receives it's ID value - 0x0 for CPU0, 0x1 for CPU1, 0x2 for PCI agent. Once locked by one of the three agents, a read by any of the other two will return the owner ID. Unlock the semaphore by writing back a value of 0xFF. A write of any other value is ignored. **NOTE:** Only the MV64360 and MV64361 can lock the CPU1 agent. Setting 0x1 is reserved for the MV64362.
31:8	Reserved	RES 0x0	Reserved.

Table 249: Semaphore1
Offset: 0x24C

Bits	Field	Type/ Init Val	Description
7:0	Semaphore	RW 0x0	Same as Semaphore0
31:8	Reserved	RES 0x0	Reserved.

Table 250: Semaphoer2
Offset: 0x254

Bits	Field	Type/ Init Val	Description
7:0	Semaphore	RW 0x0	Same as Semaphore0
31:8	Reserved	RES 0x0	Reserved.

Table 251: Semaphore3
Offset: 0x25C

Bits	Field	Type/ Init Val	Description
7:0	Semaphore	RW 0x0	Same as Semaphoer0
31:8	Reserved	RES 0x0	Reserved.

Table 252: Semaphore4
Offset: 0x264

Bits	Field	Type/ Init Val	Description
7:0	Semaphore	RW 0x0	Same as Semaphore0
31:8	Reserved	RES 0x0	Reserved.

Table 253: Semaphore5
Offset: 0x26C

Bits	Field	Type/ Init Val	Description
7:0	Semaphore	RW 0x0	Same as Semaphore0
31:8	Reserved	RES 0x0	Reserved.

Table 254: Semaphore6
Offset: 0x274

Bits	Field	Type/ Init Val	Description
7:0	Semaphore	RW 0x0	Same as Semaphore0
31:8	Reserved	RES 0x0	Reserved.

Table 255: Semaphore7
Offset: 0x27C

Bits	Field	Type/ Init Val	Description
7:0	Semaphore	RW 0x0	Same as Semaphore0
31:8	Reserved	RES 0x0	Reserved.

A.6 CPU Sync Barrier Registers

Table 256: CPU0 Sync Barrier Trigger
Offset: 0x0C0

Bits	Field	Type/ Init Val	Description

3:0	SBTrig	RW
		0x0
			Sync Barrier Trigger.
			A write to this register triggers sync barrier process. The four bits, define which buffers should be flushed.
			Bit[0] = PCI_0 slave write buffer
			Bit[1] = PCI_1 slave write buffer
			bit[2] = SDRAM snoop queue
			bit[3] = Integrated SRAM snoop queue
			NOTE: The PCI_1 slave write buffer and integrated SRAM snoop queue is only valid for the MV64360 and MV64361. These bits are reserved in the MV64362.

31:4	Reserved	RES 0x0	Reserved.

Table 257: CPU0 Sync Barrier Virtual
Offset: 0x0C8

Bits	Field	Type/ Init Val	Description
31:0	SBStat	RW 0x0	As long as sync barrier is in progress, read from this register results in value of 0xFFFF.FFFF. As soon as sync barrier is resolved, read results in 0x0.

Table 258: CPU1 Sync Barrier Trigger
Offset: 0x0D0

NOTE: Only valid for the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Description
31:0	SBTrig	RW 0x0	Same as CPU0 Sync Barrier Trigger register

Table 259: CPU1 Sync Barrier Virtual
Offset: 0x0D8

NOTE: Only valid for the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Description
31:0	SBStat	RW 0x0	Same as CPU0 Sync Barrier Virtual register

A.7 CPU Access Protect Registers

Table 260: CPU Protect Window 0 Base Address
Offset: 0x180

Bits	Field	Type/ Init Val	Description
19:0	Base	RW 0x0	CPU Protect Region 0 Base Address Corresponds to address bits[35:16]
20	AccProtect	RW 0x0	CPU Access Protect. 0 = Access allowed. 1 = Access forbidden.
21	WrProtect	RW 0x0	CPU Write Protect 0 = Write allowed. 1 = Write forbidden.
22	CacheProtect	RW 0x0	CPU caching protect 0 = Caching (block read) is allowed. 1 = Caching is forbidden.
30:23	Reserved	RES 0x0	Reserved.
31	En	RW 0x0	Window Enable. 0 = Window disabled. CPU address is not checked against this window 1 = Window is enabled. CPU address is checked to not violate the access attributes.

Table 261: CPU Protect Window 0 Size
Offset: 0x188

Bits	Field	Type/ Init Val	Description
15:0	Size	RW 0x0	CPU Protect Region 0 Bank Size Corresponds to base address bits[15:0]
31:16	Reserved	RES 0x0	Reserved.

Table 262: CPU Protect Window 1 Base Address
Offset: 0x190

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as Protect Window 0 Base Address.

**Table 263: CPU Protect Window 1 Size
Offset: 0x198**

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as Protect Window 0 Size

**Table 264: CPU Protect Window 2 Base Address
Offset: 0x1A0**

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as Protect Window 0 Base Address.

**Table 265: CPU Protect Window 2 Size
Offset: 0x1A8**

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as Protect Window 0 Size.

**Table 266: CPU Protect Window 3 Base Address
Offset: 0x1B0**

Bits	Field	Type/ Init Val	Description
31:19	Various	RW 0x0	Same as Protect Window 0 Base Address.

**Table 267: CPU Protect Window 3 Size
Offset: 0x1B8**

Bits	Field	Type/ Init Val	Description
31:0	Various	RW 0x0	Same as Protect Window 0 Size.

A.8 CPU Error Report Registers

Table 268: CPU Error Address (Low)
Offset: 0x070[1]

Bits	Field	Type/ Init Val	Description
31:0	ErrAddr	RO 0x0	Latched address bits [31:0] of a CPU transaction in case of: • illegal address (failed address decoding) • access protection violation • bad data parity • bad address parity Read Only.

1. In case of multiple errors, only the first one is latched. New error report latching is enabled only after the CPU Error Address (Low) register is being read.

Note

In case of a transaction with an erroneous address that was retargeted, the latched address in the Error Report Register is the retargeted address.

Table 269: CPU Error Address (High)
Offset: 0x078

NOTE: Once data is latched, no new data can be registered (due to additional error condition), until CPU Error Low Address is being read (which implies, it should be the last being read by the interrupt handler).

Bits	Field	Type/ Init Val	Description
3:0	ErrAddr	RO 0x0	Latched address bits [35:32] of a CPU transaction in case of: • illegal address (failed address decoding) • access protection violation • bad data parity. Read Only.
8:4	ErrPar	RO 0x0	Latched address parity bits in case of bad CPU address parity detection. Read Only.
9	HIT	RW 0x0	If set to '1', indicates that data intervention is taking place (HIT# was asserted), and that the latched address is the address of the cache line to be snarfed (address bits[4:0] are 0x0, and not the original address initiated by the CPU)
31:9	Reserved	RES 0x0	Reserved.

Table 270: CPU Error Data (Low)
Offset: 0x128

NOTE: Once data is latched, no new data can be registered (due to additional error condition), until CPU Error Low Address is being read (which implies, it should be the last being read by the interrupt handler).

Bits	Field	Type/ Init Val	Description
31:0	PErrData	RO 0x0	Latched data bits [31:0] in case of bad data parity sampled on write transactions or on master read transactions on the 60x bus. Read only.

Table 271: CPU Error Data (High)
Offset: 0x130

NOTE: Once data is latched, no new data can be registered (due to additional error condition), until CPU Error Low Address is being read (which implies, it should be the last being read by the interrupt handler).

Bits	Field	Type/ Init Val	Description
31:0	PErrData	RO 0x0	Latched data bits [63:32] in case of bad data parity sampled on write transactions or on master read transactions on the 60x bus. Read only.

Table 272: CPU Error Parity
Offset: 0x138

NOTE: Once data is latched, no new data can be registered (due to additional error condition), until CPU Error Low Address is being read (which implies, it should be the last being read by the interrupt handler).

Bits	Field	Type/ Init Val	Description
7:0	PErrPar	RO 0x0	Latched data parity bus in case of bad data parity sampled on write transactions or on master read transactions on the 60x bus. Read only.
31:10	Reserved	RES 0x0	Reserved.

Table 273: CPU Error Cause
Offset: 0x140

NOTE: Bits[7:0] are clear only. A cause bit is set upon an error condition occurrence. Writing a ‘0’ value clears the bit. Writing a ‘1’ value has no affect.

Bits	Field	Type/ Init Val	Description
0	AddrOut	RW 0x0	CPU Address Out of Range
1	AddrPErr	RW 0x0	Bad Address Parity Detected

**Table 273: CPU Error Cause (Continued)
Offset: 0x140**

NOTE: Bits[7:0] are clear only. A cause bit is set upon an error condition occurrence. Writing a '0' value clears the bit. Writing a '1' value has no affect.

Bits	Field	Type/ Init Val	Description
2	TTErr	RW 0x0	Transfer Type/Init Val Violation. The CPU attempts to burst (read or write) to an internal register.
3	AccErr	RW 0x0	Access to a Protected Region
4	WrErr	WO 0x0	Write to a Write Protected Region
5	CacheErr	RO 0x0	Read from a Caching protected region
6	WrDataPErr	WO 0x0	Bad Write Data Parity Detected
7	RdDataPErr	RO 0x0	Bad Read Data Parity Detected
26:8	Reserved	RES 0x0	Reserved.
31:27	Sel	RO 0x0	Specifies the error event currently being reported in Error Address, Error Data, and Error Parity registers: 0x0 = AddrOut 0x1 = AddrPErr 0x2 = TTErr 0x3 = AccErr 0x4 = WrErr 0x5 = CacheErr 0x6 = WrDataPErr 0x7 = RdDataPErr 0x8-0x1f = Reserved Read Only.

**Table 274: CPU0 Error Mask
Offset: 0x148**

Bits	Field	Type/ Init Val	Description
0	AddrOut	RW 0x0	AddrOut Interrupt 0 = Disabled 1 = Enabled
1	AddrPErr	0x0	AddrPErr Interrupt 0 = Disabled 1 = Enabled

Table 274: CPU0 Error Mask (Continued)
Offset: 0x148

Bits	Field	Type/ Init Val	Description
2	TTErr	RW 0x0	TTErr Interrupt 0 = Disabled 1 = Enabled
3	AccErr	RW 0x0	AccErr interrupt 0 = Disabled 1 = Enabled
4	WrErr	RW 0x0	WrErr Interrupt 0 = Disabled 1 = Enabled
5	CacheErr	RW 0x0	CacheErr Interrupt 0 = Disabled 1 = Enabled
6	WrDataPErr	RW 0x0	WrDataPErr Interrupt 0 = Disabled 1 = Enabled
7	RdDataPErr	RW 0x0	RdDataPErr interrupt 0 = Disabled 1 = Enabled
31:8	Reserved	RES 0x0	Reserved.

Appendix B. Integrated SRAM Registers

Note

These registers only apply to the MV64360 and MV64361 devices. The MV64362 does not have integrated SRAM

B.1 Integrated SRAM Register Map

Table 275: Integrated SRAM Register Map

Register	Offset	Page
SRAM Configuration	0x380	Table 276, p.484
SRAM Test Mode	0x3F4	Table 277, p.485
SRAM Error Cause	0x388	Table 278, p.486
SRAM Error Address	0x390	Table 279, p.486
SRAM Error Address (High)	0x3f8	Table 280, p.486
SRAM Error Data (Low)	0x398	Table 281, p.486
SRAM Error Data (High)	0x3a0	Table 282, p.487
SRAM Error Parity	0x3a8	Table 283, p.487

B.2 SRAM Control Registers

Table 276: SRAM Configuration
Offset: 0x380

Bits	Field	Type/ Init Val	Description
1:0	CCEn	RW 0x0	Cache Coherency Enable 0x0 = Disable 0x1 = WT cache 0x2 = WB cache 0x3 = Reserved
3:2	Reserved	RES 0x0	Reserved.

Table 276: SRAM Configuration (Continued)
Offset: 0x380

Bits	Field	Type/ Init Val	Description
4	ParEn	RW 0x1	Parity Enable 0 = Disabled SRAM parity is not checked nor generated. 1 = Enabled
5	PerrProp	RW 0x0	Parity Errors Propagation Enable 0 = Disabled. The device always generates correct parity on write access to SRAM. 1 = Enabled. The device forces bad parity if a parity error indication is sent from the originating unit.
6	ForceParEn	RW 0x0	Forces user specific parity to SRAM. **NOTE:** For debug purposes only. 0 = Disabled 1 = Enabled
7	Park	RW 0x0	Integrated SRAM Arbiter Parking Select 0 = Parked on CPU access 1 = Parked on crossbar request
15:8	ForcePar	RW 0x0	User defined parity to be forced to SRAM. ForceParEn bit must be set to ’1’ for ForcePar to apply.
18:16	RTC	RES 0x6	Reserved for Marvell Technology usage.
20:19	WTC	RES 0x2	Reserved for Marvell Technology usage.
31:21	Reserved	RES 0x0	Reserved.

Table 277: SRAM Test Mode
Offset: 0x3F4

Bits	Field	Type/ Init Val	Description
31:0	Reserved	RES 0x0	Reserved.

B.3 SRAM Error Report Registers

Table 278: SRAM Error Cause
Offset: 0x388

Bits	Field	Type/ Init Val	Description
7:0	Err	RW 0x0	Bit Error Per Byte Bit[0] indicates parity error on data byte[7:0]. Bit[1] indicates parity error on byte[15:8] and so on. If any of these bits is set, the SRAM bit in the Main Interrupt Cause (Low) register is set. Write ’0’ to clear interrupt.
31:8	Reserved	RES 0x0	Reserved.

Table 279: SRAM Error Address
Offset: 0x390

NOTE: If multiple errors occur, only the first error is latched. New error report latching is only enabled after the SRAM Error Address register is read.

Bits	Field	Type/ Init Val	Description
31:0	Addr	RW 0x0	Error Address bits[31:0] Latched upon SRAM parity error detection.

Table 280: SRAM Error Address (High)
Offset: 0x3f8

NOTE: If multiple errors occur, only the first error is latched. New error report latching is only enabled after the SRAM Error Address register is read.

Bits	Field	Type/ Init Val	Description
3:0	Addr	RW 0x0	Error Address bits[35:32] Latched upon SRAM parity error detection.
31:4	Reserved	RES 0x0	Reserved.

Table 281: SRAM Error Data (Low)
Offset: 0x398

NOTE: If multiple errors occur, only the first error is latched. New error report latching is only enabled after the SRAM Error Address register is read.

Bits	Field	Type/ Init Val	Description
31:0	Data	RW 0x0	Bits[31:0] of the error data. Latched upon SRAM parity error detection.

**Table 282: SRAM Error Data (High)
Offset: 0x3a0**

NOTE: If multiple errors occur, only the first error is latched. New error report latching is only enabled after the SRAM Error Address register is read.

Bits	Field	Type/ Init Val	Description
31:0	Data	RW 0x0	Bits[63:32] of the error data. Latched upon SRAM parity error detection

**Table 283: SRAM Error Parity
Offset: 0x3a8**

NOTE: If multiple errors occur, only the first error is latched. New error report latching is enabled only after the SRAM Error Address register is read.

Bits	Field	Type/ Init Val	Description
7:0	Par	RW 0x0	Error parity. Latched upon SRAM parity error detection
31:8	Reserved	RES 0x0	Reserved.

Appendix C. DDR SDRAM Controller Registers

C.1 SDRAM Interface Register Maps

Table 284: SDRAM Control Register Map

Register	Offset	Page
SDRAM Configuration	0x1400	Table 287, p.489
Dunit Control (Low)	0x1404	Table 288, p.490
Dunit Control (High)	0x1424	Table 289, p.492
SDRAM Timing (Low)	0x1408	Table 290, p.493
SDRAM Timing (High)	0x140C	Table 291, p.493
SDRAM Address Control	0x1410	Table 292, p.494
SDRAM Open Pages Control	0x1414	Table 293, p.494
SDRAM Operation	0x1418	Table 294, p.495
SDRAM Mode	0x141c	Table 295, p.495
SDRAM Interface Crossbar Control (Low)	0x1430	Table 296, p.496
SDRAM Interface Crossbar Control (High)	0x1434	Table 297, p.496
SDRAM Interface Crossbar Timeout	0x1438	Table 298, p.497
SDRAM Address/Control Pads Calibration	0x14C0	Table 299, p.497
SDRAM Data Pads Calibration	0x14C4	Table 300, p.498

Table 285: Error Report Register Map

Register	Offset	Page
SDRAM Error Data (Low)	0x1444	Table 302, p.499
SDRAM Error Data (High)	0x1440	Table 303, p.499
SDRAM Error Address	0x1450	Table 301, p.499
SDRAM Received ECC	0x1448	Table 304, p.500
SDRAM Calculated ECC	0x144C	Table 305, p.500
SDRAM ECC Control	0x1454	Table 306, p.500
SDRAM ECC Counter	0x1458	Table 307, p.501

Table 286: Controlled Delay Line (CDL) Register Map

Register	Offset	Page
DFCDL Configuration0	0x1480	Table 308, p.501
DFCDL Configuration1	0x1484	Table 309, p.502
SRAM Address	0x1490	Table 310, p.503
SRAM Data0	0x1494	Table 311, p.503
DFCDL Probe	0x14A0	Table 312, p.504

C.2 SDRAM Control Registers

Table 287: SDRAM Configuration
Offset: 0x1400

Bits	Field	Type/ Init Val	Description
13:0	Refresh	RW 0x0400	Refresh interval count value.
14	PInter	RW 0x0	Enable Physical banks (CS[3:0]#) Interleaving 0 = Interleaving enabled 1 = Interleaving disabled
15	VInter	RW 0x0	Enable Virtual banks (within the same SDRAM device) Interleaving 0 = Interleaving enabled 1 = Interleaving disabled
16	Reserved	RES 0x0	Reserved.
17	RegDRAM	RW 0x0	Enable Registered DRAM 0 = Non-registered DRAM 1 = Registered DRAM
18	ECC	RW 0x0	Enable ECC 0 = ECC not enabled 1 = ECC enabled When ECC is enabled, every partial write to the DRAM results in RMW.
19	Reserved	RES 0x0	Must be ’0’.

Table 287: **SDRAM Configuration (Continued)**
Offset: 0x1400

Bits	Field	Type/ Init Val	Description
21:20	DQS	RW 0x2	Data DQS Pins 0x0 = Two pins (DQS[4,0]) for x32 devices 0x1 = Reserved 0x2 = Eight pins (DQS[7:0] for x8 and x16 devices 0x3 = Sixteen pins (DQS[16:9,7:0]) for x4 devices **NOTE:** When using x4 devices, data mask (DM) is not supported. When using x4 devices, use DQS[17,8] as the data strobe for the ECC byte. In all other configurations, use DQS[8].
25:22	Reserved	RES 0x0	Reserved.
31:26	RdBuff	RW 0x1	Read buffer assignment per each interface. If set to '0', the corresponding unit receives read data from read buffer 0. If set to '1', it receives read data from read buffer 1. [26] = CPU read [27] = PCI_0 read [28] = PCI_1 read [29] = MPSC read [30] = IDMA read [31] = Gb read **NOTE:** In the MV64362, bit [28] is reserved.

Table 288: **Dunit Control (Low)**
Offset: 0x1404

Bits	Field	Type/ Init Val	Description
0	ClkSync	RW Sampled at reset on DevAD[18]	Clock Domains Synchronization 0 = Core signals are synchronized in the Dunit 1 = Core signals synchronizers are bypassed **NOTE:** Synchronizers bypass is only allowed when the core and the Dunit run with the same clock.
1	RdSyncSel	RW Sampled at reset on DevAD[22]	Read Data Synchronization Select 0 = Read data is synchronized to core clock 1 = Read data is synchronized to FBClkIn
2	RdCtrltDel	RW Sampled at reset on DevAD[23]	Read Control Logic Delay 0 = Disabled 1 = Enabled The clock to the read control logic is delayed (using DFCDL).

**Table 288: Dunit Control (Low)
Offset: 0x1404 (Continued)**

Bits	Field	Type/ Init Val	Description
3	RdDataDel	RW Sampled at reset on DevAD[24]	Read Data Delay 0 = Disabled 1 = Enabled The read data sample clock is delayed (using DFCDL). **NOTE:** If running in sync mode (ClkSync bit is set to 1), bits[3:1] must be 000 or 100. If running async mode, bits[3:1] must be 001 or 111.
5:4	CtrlPipe	RW Sampled at reset on DevAD [21:20]	Number of pipeline stages in the Dunit control path. 0x0 = Reserved 0x1 = Two pipe stages (Up to 133 MHz) 0x2 = Reserved 0x3 = Three pipe stages (Up to 183 MHz)
6	CtrlPos	RW Sampled at reset on DevAD[19]	Address/Control Output Timing 0 = On falling edge of clock 1 = On rising edge of clock
7	RdPipe	RW 0x1	Number of pipeline stages in the read data path. 0x0 = Bypass 0x1 = One stage
8	RdSyncEn	RW 0x1	Read Data Path Synchronization 0 = Disabled 1 = Enabled
9	RMWSyncEn	RW 0x1	RMW Path Synchronization 0 = Disabled 1 = Enabled
15:10	Prio	RW 0x1	Priority assignment for each interface. Set to 1, has high priority. If set to 0, normal priority. 0 = Normal priority 1 = High priority [10] = CPU [11] = PCI_0 [12] = PCI_1 [13] = MPSC [14] = IDMA [15] = Gb **NOTE:** In the MV64362, bit [12] is reserved.
19:16	LCnt	RW 0x1	Arbiter Low Priority Counter
23:20	HCnt	RW 0x1	Arbiter High Priority Counter
26:24	StBurstDel	RW 0x3	Number of sample stages on StartBurstIn. Program StBurstDel based on CL, registered/non-buffered DIMM, and frequency.

Table 288: Dunit Control (Low)
Offset: 0x1404 (Continued)

Bits	Field	Type/ Init Val	Description
27	StBurstNeg	RW 0x0	If set to '1', StartBurstIn is first sampled on the falling edge of clock.
28	StBurstSrc	RW 0x0	StartBurst source 0 = Generated in the Dunit internally. 1 = Generated from StartBurstIn input pin.
29	RdDataNeg	RW 0x0	If set to '1', read data is first sampled with falling edge of clock. Depends on CL and frequency
31:30	Reserved	RES 0x0	Reserved.

Table 289: Dunit Control (High)
Offset: 0x1424

Bits	Field	Type/ Init Val	Description
3:0	WrBuff		Reserved
7:4	RdBuff		Reserved
11:8	TxQue		Reserved
15:12	WrTrig		Reserved
19:16	RdTrig		Reserved
23:20	RMWTrig		Reserved
24	SnoopPipe	RW 0x0	Snoops pipeline enable 0 = Disable (Single Snoop) 1 = Enable
28:25	SnoopDepth		Reserved
31:29	Reserved	RES 0x0	Reserved.

Table 290: SDRAM Timing (Low)
Offset: 0x1408

Bits	Field	Type/ Init Val	Description
3:0	t_{DQSS}	R 0x0	Write Command to DQS Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on. Must be ’0’.
7:4	t_{RCD}	RW 0x2	Activate to Command Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on. The only valid values are ’1’, ’2’, or ’3’.
11:8	t_{RP}	RW 0x2	Precharge Command Period Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on. The only valid values are ’1’, ’2’, or ’3’.
15:12	t_{WR}	RW 0x1	Write Command to Precharge Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on. The only valid values are ’1’, ’2’, or ’3’.
19:16	t_{WTR}	RW 0x0	Write Command to Read Command Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on. The only valid values are ’0’ and ’1’.
23:20	t_{RAS}	RW 0x5	Minimum Row Active Time Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on. The only valid values are ’4’, ’5’, ’6’, ’7’, or ’8’.
27:24	t_{RRD}	RW 0x1	Activate Bank A to Activate Bank B Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on. The only valid values are ’0’, ’1’, or ’2’.
31:28	Reserved	RES 0x1	Reserved.

Table 291: SDRAM Timing (High)
Offset: 0x140C

Bits	Field	Type/ Init Val	Description
3:0	t_{RFC}	RW 0x9	Refresh Command Period Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on. **NOTE:** The value must be equal to or greater than 0x7.
5:4	t_{RD2RD}	RW 0x0	Minimum Gap Between DRAM Read Accesses Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on. The only valid values are ’0’ or ’1’.
7:6	t_{RD2WR}	RW 0x0	Minimum Gap Between DRAM Read and Write Accesses Value ‘0’ means one cycle; value of ‘1’ means two cycles; and so on The only valid values are ’0’ or ’1’.
31:8	Reserved	RES 0x0	Reserved.

Table 292: SDRAM Address Control
Offset: 0x1410

Bits	Field	Type/ Init Val	Description
3:0	AddrSel	RW 0x2	SDRAM Address Select Determines what address bits to drive on DA[13:0] and BA[1:0] during activate and command phases (row and column addresses). **NOTE:** See 11.4.2 “SDRAM Address Control” on page 136.
5:4	DCfg	RW 0x1	SDRAM Device Configuration Affects DRAM row and column address bits multiplexing (11.2 “DRAM Size” on page 132). 0x0 = 64 Mb or 128 Mb 0x1 = 256 Mb or 512 Mb 0x2 = 1 Gb or 2 Gb 0x3 = Reserved
31:6	Reserved	RES 0x0	Reserved.

Table 293: SDRAM Open Pages Control
Offset: 0x1414

Bits	Field	Type/ Init Val	Description
15:0	OPEn	RW 0xFFFF	Open Page Enable Bit per bank (bit[0] corresponds to CS[0]# bank0, bit[1] to CS[0]# bank1, bit[2] to CS[0]# bank2, bit[3] to CS[0]# bank3, bit[4] to CS[1]# bank0, bit[5] to CS[1]# bank1 and so on). 0 = The MV64360/1/2 keeps the corresponding bank page open whenever it can. 1 = The MV64360/1/2 always closes the page at the end of the transaction.
31:16	Reserved	RES 0x0	Reserved.

Table 294: SDRAM Operation
Offset: 0x1418

Bits	Field	Type/ Init Val	Description
2:0	Cmd	RW 0x0	DRAM Mode Select 0x0 = Normal SDRAM Mode 0x1 = Precharge all banks command 0x2 = Refresh all banks command 0x3 = Mode Register Set command 0x4 = Extended Mode Register Set command 0x5 = NOP command 0x6,0x7 = Reserved Setting Cmd results in the Dunit execution of the required command to the DRAM. Then, the Dunit resets Cmd to the default 0x0 value.
31:3	Reserved	RES 0x0	Reserved.

Table 295: SDRAM Mode
Offset: 0x141c

Bits	Field	Type/ Init Val	Description
2:0	BL	RW 0x2	Burst Length 0x0-0x1 = Reserved 0x2 = BL = 4 0x3-0x7 = Reserved
3	BT	RW 0x0	Burst Type/Init Val Must be set to ‘0’.
6:4	CL	RW 0x2	CAS Latency 0x0-0x1 = Reserved 0x2 = CL = 2 0x3 = CL = 3 0x4 = Reserved 0x5 = CL = 1.5 0x6 = CL = 2.5 0x7 = Reserved
13:7	OM	RW 0x0	Operation Mode 0x0 = Normal operation 0x1 = Reserved 0x2 = Normal operation, reset DLL 0x3 - 1f = Reserved
31:14	Reserved	RES 0x0	Reserved.

Table 296: SDRAM Interface Crossbar Control (Low)
Offset: 0x1430

Bits	Field	Type/ Init Val	Description
3:0	Arb0	RW 0x2	Slice 0 of device controller “pizza” arbiter 0x0 = NULL request 0x1 = Reserved 0x2 = CPU access 0x3 = PCI_0 access 0x4 = PCI_1 access 0x5 = MPSCs access 0x6 = IDMA access 0x7 =Ethernet controller access 0x8 - 0xF = Reserved **NOTE:** Setting 0x3 only applies to the MV64360 and MV64361. Reserved in the MV64362.
7:4	Arb1	RW 0x3	Slice 1 of device controller “pizza” arbiter
11:8	Arb2	RW 0x4	Slice 2 of device controller “pizza” arbiter
15:12	Arb3	RW 0x5	Slice 3 of device controller “pizza” arbiter
19:16	Arb4	RW 0x6	Slice 4 of device controller “pizza” arbiter
23:20	Arb5	RW 0x7	Slice 5 of device controller “pizza” arbiter
27:24	Arb6	RW 0x0	Slice 6 of device controller “pizza” arbiter
31:28	Arb7	RW 0x0	Slice 7 of device controller “pizza” arbiter

Table 297: SDRAM Interface Crossbar Control (High)
Offset: 0x1434

Bits	Field	Type/ Init Val	Description
3:0	Arb8	RW 0x2	Slice 8 of device controller “pizza” arbiter.
7:4	Arb9	RW 0x3	Slice 9 of device controller “pizza” arbiter.
11:8	Arb10	RW 0x4	Slice 10 of device controller “pizza” arbiter.
15:12	Arb11	RW 0x5	Slice 11 of device controller “pizza” arbiter.

Table 297: SDRAM Interface Crossbar Control (High) (Continued)
Offset: 0x1434 (Continued)

Bits	Field	Type/ Init Val	Description
19:16	Arb12	RW 0x6	Slice 12 of device controller "pizza" arbiter.
23:20	Arb13	RW 0x7	Slice 13 of device controller "pizza" arbiter.
27:24	Arb14	RW 0x0	Slice 14 of device controller "pizza" arbiter.
31:28	Arb15	RW 0x0	Slice 15 of device controller "pizza" arbiter.

Table 298: SDRAM Interface Crossbar Timeout
Offset: 0x1438

Bits	Field	Type/ Init Val	Description
7:0	Timeout	RW 0xFF	CrossBar Arbiter Timeout Preset Value
15:8	Reserved	RES 0x0	Reserved.
16	TimeoutEn	RW 0x1	CrossBar Arbiter Timer Enable 0 = Enable 1 = Disable
31:17	Reserved	RES 0x0	Reserved.

Table 299: SDRAM Address/Control Pads Calibration
Offset: 0x14C0

Bits	Field	Type/ Init Val	Description
4:0	DrvN	RW 0x0A	Pad Nchannel Driving Strength **NOTE:** Only applicable when dynamic tune is disabled.
9:5	DrvP	RW 0x0A	Pad Pchannel Driving Strength **NOTE:** Only applicable when dynamic tune is disabled.
15:10	Reserved	RES 0x0	Read Only

Table 299: SDRAM Address/Control Pads Calibration (Continued)
Offset: 0x14C0

Bits	Field	Type/ Init Val	Description
16	TuneEn	RW Sampled at reset. PCI_0: DevAD[12] PCI_1: DevAD[13]	Set to ‘1’ enables the dynamic tuning of pad driving strength.
21:17	LockN	RO 0x0	When dynamic tune is enabled, represents the final locked value of the Nchannel Driving Strength. Read Only
26:22	LockP	RO 0x0	When dynamic tune is enabled, represents the final locked value of the Pchannel Driving Strength Read Only
30:27	Reserved	RES 0x0	Read Only
31	WrEn	RW 0x0	Write Enable CPU Pads Calibration register 0 = Register is read only (except for bit[31]). 1 = Register is writable.

Table 300: SDRAM Data Pads Calibration
Offset: 0x14C4

Bits	Field	Type/ Init Val	Description
31:0	Various	RW RO RES 0x144	Same as Address/Control Pads Calibration register.

C.3 SDRAM Error Report Registers

Notes

- There is no DRAM Interrupt Cause register. If ECC is enabled, and ECC error is detected, ECC bit in the Main Interrupt Cause register is set. A write of 0x0 to the SDRAM Error Address register, clears the interrupt, and enables registration of new error if occurs.
- If multiple errors occur, only the first error is latched. New error report latching is only enabled after the SDRAM Error Address register is being read, and interrupt is cleared.

Table 301: SDRAM Error Address
Offset: 0x1450

Bits	Field	Type/ Init Val	Description
0	ErrType/Init Val	RW 0x0	Indicates what Type/Init Val of ECC error is currently reported: 0 = Single bit error threshold expired 1 = Two bit error
2:1	CS	RW 0x0	DRAM chip select. Indicates in which of the four DRAM banks the error occurred: 0x0 = CS[0]# 0x1 = CS[1]# 0x2 = CS[2]# 0x3 = CS[3]# Useful for software to reproduce of the original 36-bit address
31:2	ECCAddr	RW 0x0	Sampled address of the last data with ECC error.

Table 302: SDRAM Error Data (Low)
Offset: 0x1444

Bits	Field	Type/ Init Val	Description
31:0	ECCData	RW 0x0	Sampled 32 low bits of the last data with ECC error.

Table 303: SDRAM Error Data (High)
Offset: 0x1440

Bits	Field	Type/ Init Val	Description
31:0	ECCData	RW 0x0	Sampled 32 high bits of the last data with ECC error.

Table 304: SDRAM Received ECC
Offset: 0x1448

Bits	Field	Type/ Init Val	Description
7:0	ECCReg	RW 0x0	ECC code being read from SDRAM
31:8	Reserved	RES 0x0	Reserved.

Table 305: SDRAM Calculated ECC
Offset: 0x144C

Bits	Field	Type/ Init Val	Description
7:0	ECCCalc	RW 0x0	ECC code calculated by Dunit
31:8	Reserved	RES 0x0	Reserved.

Table 306: SDRAM ECC Control
Offset: 0x1454

Bits	Field	Type/ Init Val	Description
7:0	ForceECC	RW 0x0	User defined ECC byte to be written to ECC bank.
8	ForceECC	RW 0x0	Force user defined ECC byte on SDRAM writes. 0 = Write calculated ECC byte 1 = Write user defined ECC byte
9	PerrProp	RW 0x0	Propagate parity errors to ECC bank. 0 = Dunit always generate correct ECC on write access to DRAM. 1 = Dunit generates an uncorrected ECC error on write access to DRAM, in case of parity error indication from the originating interface.
15:10	Reserved	RES 0x0	Reserved.
23:16	ThrEcc	RW 0x0	Threshold ECC Interrupt Number of single bit errors to happen before the Dunit generates an interrupt. **NOTE:** If set to 0x0, the Dunit does not generate an interrupt in case of a single bit error.
31:24	Reserved	RES 0x0	Reserved.

**Table 307: SDRAM ECC Counter
Offset: 0x1458**

Bits	Field	Type/ Init Val	Description
31:0	Count	R 0x0	Number of single bit ECC errors detected. If the number of detected errors reach 2^{32}, it wraps around to 0x0

C.4 CDL Registers

Note

DFCDL Configuration 0/1 and DFCDL Probe registers are used for debug, and should not be changed during normal operation. The only exception is DFCDL Configuration 0 register’s bit[15], that should be cleared after SRAM initialization, in order to enable delay lines update.

**Table 308: DFCDL Configuration0
Offset: 0x1480**

Bits	Field	Type/ Init Val	Description
7:0	UpdWin	RW 0x0	The window size, after the refresh command, in which DFCDL update is allowed 0 = one cycle 1 = two cycles and so on
12:8	Reserved	RES 0x0	Reserved.
13	ForceUpdSync	RW 0x0	If set to ’1’, forces the delay line update as soon as DFCDL is synchronized. Cleared by hardware as soon as delay line is updated.
14	ForceUpdW	RW 0x0	If set to ’1’, forces delay line update as soon as update window arrives. Cleared by hardware as soon as delay line is updated.
15	BlockUpd	RW 0x1	If set to ’1’, disables delay line update (unless using ForceUpdSync or ForceUpdW bits). After initialization of DFCDL SRAM, must be set to ’0’ to enable dynamic delay line update.
16	UpdNoSync	RW 0x0	If set to ’1’, enables dynamic update without reaching sync condition.
17	UpdNoWin	RW 0x0	If set to ’1’, enables dynamic update without reaching update window.
18	ForceAcc	RW 0x0	If set to ’1’, forces the filter state machine to accept bad values.
27:19	MaxDiff	RW 0x6	**NOTE:** Maximum difference between consecutive updates Filtering is performed on values multiplied by 4

Table 308: DFCDL Configuration0 (Continued)
Offset: 0x1480

Bits	Field	Type/ Init Val	Description
31:28	Reserved	RES 0x0	Reserved.

Table 309: DFCDL Configuration1
Offset: 0x1484

Bits	Field	Type/ Init Val	Description
5:0	DelPVal	RW 0x0	Delay counter preset value.
6	4Cell	RW 0x0	Delay unit selects 0 = Four cells 1 = Two cells Must be ’0’.
7	ISense	RW 0x0	If set to ’1’, multiply by two the value found by the search machine. This results in 180^0, instead of 90^0 Must be ’0’.
13:8	PhaseD	RW 0x1	Delay Counter Phase Delta Size of the delta when the search machine is searching the second edge. Must be ’1’.
14	SinglePhase	RW 0x0	If set to ’1’, the search machine only searches for first phase. Must be ’0’.
15	Reserved	RES 0x0	Reserved.
16	PhaseMode	RW 0x0	Phase Mode Jump 0 = When search machine begins search of the second edge, it continues from the current counters value. 1 = When search machine begins search of the second edge, it continues from ’0’. Must be ’0’.
17	Reserved	RES 0x0	Reserved.
19:18	Avg	RW 0x2	Average Value Calculation for Filter Process 0x0,0x3 = Average of one value 0x1 = Average of two values 0x2 = Average of four values
21:20	GoodHits	RW 0x1	For the sync machine to enter the previous sync state, the number of times the good value must be received after the bad value.
23:22	GoodSync	RW 0x3	For the sync machine to enter sync state, the number of times the good value must be received after loss of sync.

Table 309: DFCDL Configuration1 (Continued)
Offset: 0x1484

Bits	Field	Type/ Init Val	Description
24	ForceSync	RW 0x0	If set to’1’, forces the sync machine to enter the sync state.
25	HoldSync	RW 0x0	If set to’1’ and the sync machine is in sync state, forces the sync machine to maintain this state.
26	ReSync	RW 0x0	If set to’1’, forces the sync machine to enter a loss of sync state.
28:27	AvgRd	RW 0x0	Average used for read address of the SRAM. 0x0,0x3 = Average of one value 0x1 = Average of two values 0x2 = Average of four values
29	StopImid	RW 0x0	If set to ’1’ and the filter machine is in idle state, forces the filter machine to enter stop state
30	StopSync	RW 0x0	If set to ’1’ and the filter machine is in idle state, forces it to enter stop state, if there is sync condition
31	GoInit	RW 0x0	If set to ’1’ and the filter machine is in idle state, forces it to remain in this state.

Table 310: SRAM Address
Offset: 0x1490

NOTE: Integrated SRAM only applies to the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Description
31:0	Addr	RW 0x0	SRAM address

Table 311: SRAM Data0
Offset: 0x1494

NOTE: Integrated SRAM only applies to the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Description
31:0	Data	RW 0x0	SRAM Write Data To initialize the DFCDL SRAM, the SRAM address must be updated first and then the SRAM data is written. Consecutive writes to the SRAM Data register are written to consecutive SRAM addresses. The SRAM address is automatically incremented with each write. **NOTE:** For detailed information about the recommended DFCDL initialization sequence, see 11.13 “DRAM Read Data Sample” on page 145.

Table 312: DFCDL Probe
Offset: 0x14A0

Bits	Field	Type/ Init Val	Description
3:0	BusSel	RW 0x0	Select DFCDL bus to be probed.
4	ProbEn	RW 0x0	Probe Enabled 0 = Disabled 1 = Enabled
31:5	Reserved	RES 0x0	Reserved.

Appendix D. Device Controller Registers

D.1 Device Interface Registers

Table 313: Device Control Register Map

Register	Offset	Page
Device Bank0 Parameters	0x45C	Table 315, p.505
Device Bank1 Parameters	0x460	Table 316, p.507
Device Bank2 Parameters	0x464	Table 317, p.507
Device Bank3 Parameters	0x468	Table 318, p.507
Boot Device Bank Parameters	0x46C	Table 319, p.507
Device Interface Control	0x4C0	Table 320, p.508
Device Interface Cross Bar Control (Low)	0x4C8	Table 321, p.509
Device Interface Cross Bar Control (High)	0x4CC	Table 322, p.509
Device Interface Cross Bar Timeout	0x4C4	Table 323, p.510

Table 314: Device Interrupts Register Map

Register	Offset	Page
Device Interrupt Cause	0x4D0	Table 324, p.510
Device Interrupt Mask	0x4D4	Table 325, p.511
Device Error Address	0x4D8	Table 326, p.512
Device Error Data	0x4DC	Table 327, p.512
Device Error Parity	0x4E0	Table 328, p.512

D.2 Device Control Registers

Table 315: Device Bank0 Parameters
Offset: 0x45C

Bits	Field	Type/ Init Val	Function
2:0	TurnOff	RW 0x7	The number of cycles in a read access between the de-assertion of CSTiming# to a new device bus cycle.

Table 315: Device Bank0 Parameters (Continued)
Offset: 0x45C

Bits	Field	Type/ Init Val	Function
6:3	Acc2First	RW 0xF	The number of cycles in a read access between the assertion of CSTiming# to the cycle that the first data is sampled by the MV64360/1/2.
10:7	Acc2Next	RW 0xF	The number of cycles in a burst read access between the cycle that the first data is sampled by the MV64360/1/2 to the cycle that the next data is sampled.
13:11	ALE2Wr	RW 0x7	The number of cycles in a write access from the ALE de-assertion to the assertion of Wr#.
16:14	WrLow	RW 0x7	The number of cycles in a write access that the Wr# signal is kept active. **NOTE:** If WrLow is set to ’0’, Ready# is not supported.
19:17	WrHigh	RW 0x7	The number of cycles in a burst write access that the Wr# signal is kept de-asserted. **NOTE:** If WrHigh is set to ’0’, Ready# is not supported.
21:20	DevWidth	RW 0x2	Device Width 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = Reserved
22	TurnOffExt	RW 0x1	TurnOff Extension The MSB of the TurnOff parameter.
23	Acc2FirstExt	RW 0x1	Acc2First Extension The MSB of the Acc2First parameter.
24	Acc2NextExt	RW 0x1	Acc2Next Extension The MSB of the Acc2Next parameter.
25	ALE2WrExt	RW 0x1	ALE2Wr Extension The MSB of the ALE2Wr parameter.
26	WrLowExt	RW 0x1	WrLow Extension The MSB of the WrLow parameter.
27	WrHighExt	RW 0x1	WrHigh Extension The MSB of the WrHigh parameter.
29:28	BadrSkew	RW 0x0	Cycles gap between BAdr toggle to read data sample. Useful when interfacing Sync Burst SRAM 0x0 - No gap (default setting) 0x1 - One cycle gap 0x2 - Two cycle gaps 0x3 - Reserved
30	DPEn	RW 0x0	Data Parity enable 0 = Disabled. Parity is not checked 1 = Enabled. Device controller checks data parity on DevDP lines

Table 315: Device Bank0 Parameters (Continued)
Offset: 0x45C

Bits	Field	Type/ Init Val	Function
31	Reserved	RES 0x1	Reserved.

Table 316: Device Bank1 Parameters
Offset: 0x460

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x8FEFFF FF	Fields function as in Device Bank0.

Table 317: Device Bank2 Parameters
Offset: 0x464

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x8FEFFF FF	Fields function as in Device Bank0.

Table 318: Device Bank3 Parameters
Offset: 0x468

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x8FEFFF FF	Fields function as in Device Bank0.

Table 319: Boot Device Bank Parameters
Offset: 0x46C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x8F?FFF FF[1]	Fields function as in Device Bank0.

1. The boot device width (bits[21:20]) are sampled by DevAD[15:14] at reset.

Table 320: Device Interface Control
Offset: 0x4C0

Bits	Field	Type/ Init Val	Function
15:0	Timeout	RW 0xFFFF	Timeout Timer Preset Value. If the device access is not completed within this preset value's period (due to a lack of Ready# assertion), the device controller completes the transaction as if Ready# was asserted and asserts an interrupt. **NOTE:** If set to 0x0, the device controller waits for Ready# assertions, forever.
16	RdTrig	RW 0x0	Read Trigger Control 0 = Drives the read data to the requesting unit only after the last data arrives from the device. 1 = Drives the read data to the requesting unit as soon as the first 64-bit data arrives from the device. **NOTE:** Only relevant on burst access to 32-bit wide devices.
17	Reserved	RES 0x0	
18	Reserved	RES 0x1	Must be 1.
19	PerrProp	RW 0x0	Parity error propagation enable 0 = Disabled. Always generate correct parity 1 = Enabled. Device controller generates bad data parity in case of erroneous data indication received from the originator unit
20	ParSel	RW 0x0	Even or Odd parity select 0 - Even 1 - Odd
21	ForceParEn	RW 0x0	Force Parity Enable 0 - Disable. Device controller drives correct parity on DP[3:0] 1 - Enable. Device controller drives user defined parity on DP[3:0] during write data phase.
23:22	Reserved	RES 0x0	
27:24	ForcePar	RW 0x0	User defined data parity. If ForceParEn is set to 1, device controller drives this parity on DevDP[3:0] during write data phase (it does not affect address parity).
29:28	Reserved	RES 0x0	
31:30	Reserved	RES 0x0	Must be 0.

Table 321: Device Interface Cross Bar Control (Low)
Offset: 0x4C8

Bits	Field	Type/ Init Val	Function
3:0	Arb0	RW 0x2	Slice 0 of the device controller “pizza” arbiter. 0x0 = Reserved 0x1 = NULL request 0x2 = CPU access 0x3 = PCI_0 access 0x4 = PCI_1 access 0x5 = MPSC access 0x6 = IDMA access 0x7 = Ethernet controller access 0x8 – 0xF = Reserved **NOTE:** The PCI_1 access setting, 0x3, only applies to the MV64360 and MV64361.
7:4	Arb1	RW 0x3	Slice 1 of the device controller “pizza” arbiter.
11:8	Arb2	RW 0x4	Slice 2 of the device controller “pizza” arbiter.
15:12	Arb3	RW 0x5	Slice 3 of the device controller “pizza” arbiter.
19:16	Arb4	RW 0x6	Slice 4 of the device controller “pizza” arbiter.
23:20	Arb5	RW 0x7	Slice 5 of the device controller “pizza” arbiter.
27:24	Arb6	RW 0x1	Slice 6 of the device controller “pizza” arbiter.
31:28	Arb7	RW 0x1	Slice 7 of the device controller “pizza” arbiter.

Table 322: Device Interface Cross Bar Control (High)
Offset: 0x4CC

Bits	Field	Type/ Init Val	Function
3:0	Arb8	RW 0x2	Slice 8 of the device controller “pizza” arbiter.
7:4	Arb9	RW 0x3	Slice 9 of the device controller “pizza” arbiter.
11:8	Arb10	RW 0x4	Slice 10 of the device controller “pizza” arbiter.
15:12	Arb11	RW 0x5	Slice 11 of the device controller “pizza” arbiter.

**Table 322: Device Interface Cross Bar Control (High)
Offset: 0x4CC (Continued)**

Bits	Field	Type/ Init Val	Function
19:16	Arb12	RW 0x6	Slice 12 of the device controller “pizza” arbiter.
23:20	Arb13	RW 0x7	Slice 13 of the device controller “pizza” arbiter.
27:24	Arb14	RW 0x1	Slice 14 of the device controller “pizza” arbiter.
31:28	Arb15	RW 0x1	Slice 15 of the device controller “pizza” arbiter.

**Table 323: Device Interface Cross Bar Timeout
Offset: 0x4C4**

Bits	Field	Type/ Init Val	Function
7:0	Timeout	RW 0xFF	CrossBar Arbiter Timeout Preset Value
15:8	Reserved	RES 0x0	Reserved.
16	TimeoutEn	RW 0x1	CrossBar Arbiter Timer Enable 0 = Enable 1 = Disable
31:17	Reserved	RES 0x0	Reserved.

D.3 Device Interrupts

**Table 324: Device Interrupt Cause
Offset: 0x4D0**

NOTE: All cause bits are clear only. They are set upon error condition cleared upon a value write of ‘0’. Writing a value of ‘1’ has no affect.

Bits	Field	Type/ Init Val	Function
0	DBurstErr	RW 0x0	Burst violation An attempt to burst more data than device controller is capable of handling.
1	DRdyErr	RW 0x0	Ready Timer Expired.

Table 324: Device Interrupt Cause
Offset: 0x4D0 (Continued)

NOTE: All cause bits are clear only. They are set upon error condition cleared upon a value write of '0'. Writing a value of '1' has no affect.

Bits	Field	Type/ Init Val	Function
2	PErr0	RW 0x0	Parity error detected on AD[7:0] during device read access
3	PErr1	RW 0x0	Parity error detected on AD[15:8] during device read access
4	PErr2	RW 0x0	Parity error detected on AD[23:16] during device read access
5	PErr3	RW 0x0	Parity error detected on AD[31:24] during device read access
26:6	Reserved	RES 0x0	Reserved.
31:27	Sel	RO 0x0	Specifies the error event currently being reported in the Error Address register. 0x0 = DBurstErr 0x1 = DRdyErr 0x2= PErr0, PErr1, PErr2 or PErr3 0x3 = 0x1f - reserved Read Only.

Table 325: Device Interrupt Mask
Offset: 0x4D4

Bits	Field	Type/ Init Val	Function
0	DBurstErr	RW 0x0	DBurstErr Interrupt 0 = Disable 1 = Enable
1	DRdyErr	RW 0x0	DRdyErr Interrupt 0 = Disable 1 = Enable
2	PErr0	RW 0x0	PErr0 interrupt 0 = Disable 1 = Enable
3	PErr1	RW 0x0	PErr1 Interrupt 0 = Disable 1 = Enable
4	PErr2	RW 0x0	PErr2 Interrupt 0 = Disable 1 = Enable

**Table 325: Device Interrupt Mask
Offset: 0x4D4**

Bits	Field	Type/ Init Val	Function
5	PErr3	RW 0x0	PErr3 Interrupt 0 = Disable 1 = Enable
31:6	Reserved	RES 0x0	Reserved.

**Table 326: Device Error Address
Offset: 0x4D8**

Bits	Field	Type/ Init Val	Function
31:0	Addr	RW 0x0	Latched Address Upon Device Error Condition After the address is latched, no new address is latched (due to additional error condition) until the register is being read.

**Table 327: Device Error Data
Offset: 0x4DC**

NOTE: No new data is latched (due to additional error condition) until the Address Error register is read.

Bits	Field	Type/ Init Val	Function
31:0	Data	RW 0x0	Latched data upon parity error detection.

**Table 328: Device Error Parity
Offset: 0x4E0**

NOTE: No new data is latched (due to additional error condition) until the Address Error register is read.

Bits	Field	Type/ Init Val	Function
3:0	Par	RW 0x0	Latched parity upon parity error detection.
31:4	Reserved	RES 0x0	Read Only.

Appendix E. PCI Interface Registers

E.1 Programing the PCI Interface Registers

In the MV64360 and MV64361, the same set of registers are duplicated for both PCI_0 and PCI_1. The only difference is that PCI_0 and PCI_1 registers are located at different offsets.

Note

For the MV64362, only use the PCI_0 offsets. Do not use the P2P registers and the PCI_1 offsets.

The PCI_1 interface contains the same set of INTERNAL registers as PCI_0 interface. However, unless specified otherwise, the PCI_1 registers offsets are PCI_0 registers offsets + 0x080. For example, the PCI_0 CS[0] Size register is located at offset 0xC08. The PCI_1 CS[0] Size register is located at offset 0xC88.

All PCI CONFIGURATION registers are located at their standard offset in the configuration header, as defined in the PCI spec, when accessed from their corresponding PCI bus. For example, if a master on PCI_0 performs a PCI configuration cycle on PCI's Status and Command Register, the register is located at 0x004. Likewise, if a master on PCI_1 performs a PCI configuration cycle on PCI_1's Status and Command Register, the register is located at 0x004.

On the other hand, if a master on PCI_0 performs a PCI configuration cycle on PCI_1's Status and Command Register, the register is located at 0x084. Likewise, if a master on PCI_1 performs a PCI configuration cycle on PCI's Status and Command Register, the register is located at 0x084.

A CPU access to the MV64360/1/2's PCI_0 configuration registers is performed via the PCI_0 Configuration Address and PCI_0 Configuration Data registers (internal registers offset 0xcf8 and 0xcfc respectively). A CPU access to the MV64360/1/2's PCI_1 configuration registers is performed via the PCI_1 Configuration Address and PCI_1 Configuration Data registers (internal registers offset 0xc78 and 0xc7c respectively).

Note

A write to unmapped offsets of the MV64360/1/2's internal registers space may result in a destructive write to existing PCI interface registers. The following is a list of these cases:

– Write to offset 0x0f3c results in a write to register at offset 0xD3C.
– Write to offset 0x0e04 results in a write to register at offset 0xC04.
– Write to offset 0x0e38 results in a write to register at offset 0xC38.
– Write to offset 0x1f04 results in a write to register at offset 0x1D04.
– Write to offset 0x1f14 results in a write to register at offset 0x1D14.
– Write to offset 0x0ef8 results in a write to register at offset 0xCF8.
– Write to offset 0x0e28 results in a write to register at offset 0xC28.
– Write to offset 0x1f5c results in a write to register at offset 0x1D5C.
– Write to offset 0x1f08 results in a write to register at offset 0x1D08.
– Write to offset 0x1f0c results in a write to register at offset 0x1D0C.

E.2 PCI Interface Register Maps

Table 329: PCI Slave Address Decoding Register Map
NOTE: The PCI_1 register offsets and the P2P registers only apply to the MV64360 and MV64361.

Register	Offsets	Page
CS[0]# BAR Size	PCI_0 0xC08, PCI_1 0xC88	Table 338, p.523
CS[1]# BAR Size	PCI_0 0xD08, PCI_1 0xD88	Table 339, p.523
CS[2]# BAR Size	PCI_0 0xC0C PCI_1 0xC8C	Table 340, p.523
CS[3]# BAR Size	PCI_0 0xD0C, PCI_1 0xD8C	Table 341, p.523
DevCS[0]# BAR Size	PCI_0 0xC10, PCI_1 0xC90	Table 342, p.524
DevCS[1]# BAR Size	PCI_0 0xD10, PCI_1 0xD90	Table 343, p.524
DevCS[2]# BAR Size	PCI_0 0xD18, PCI_1 0xD98	Table 344, p.524
DevCS[3]# BAR Size	PCI_0 0xC14, PCI_1 0xC94	Table 345, p.524
Boot CS# BAR Size	PCI_0 0xD14, PCI_1 0xD94	Table 346, p.524
P2P Mem0 BAR Size	PCI_0 0xD1C, PCI_1 0xD9C	Table 347, p.525
P2P Mem1 BAR Size	PCI_0 0xD20, PCI_1 0xDA0	Table 348, p.525
P2P I/O BAR Size	PCI_0 0xD24, PCI_1 0xDA4	Table 349, p.525
CPU BAR Size	PCI_0 0xD28, PCI_1 0xDA8	Table 350, p.525
Integrated SRAM BAR Size	PCI_0 0xE00, PCI_1 0xE80	Table 351, p.525
Expansion ROM BAR Size	PCI_0 0xD2C, PCI_1 0xDAC	Table 352, p.526
Base Address Registers Enable	PCI_0 0xC3C, PCI_1 0xCBC	Table 353, p.526
CS[0]# Base Address Remap	PCI_0 0xC48, PCI_1 0xCC8	Table 354, p.527
CS[1]# Base Address Remap	PCI_0 0xD48, PCI_1 0xDC8	Table 355, p.528
CS[2]# Base Address Remap	PCI_0 0xC4C, PCI_1 0xCCC	Table 356, p.528
CS[3]# Base Address Remap	PCI_0 0xD4C, PCI_1 0xDCC	Table 357, p.528
DevCS[0]# Base Address Remap	PCI_0 0xC50, PCI_1 0xCD0	Table 358, p.528
DevCS[1]# Base Address Remap	PCI_0 0xD50, PCI_1 0xDD0	Table 359, p.528
DevCS[2]# Base Address Remap	PCI_0 0xD58, PCI_1 0xDD8	Table 360, p.529
DevCS[3]# Base Address Remap	PCI_0 0xC54, PCI_1 0xCD4	Table 361, p.529
BootCS# Base Address Remap	PCI_0 0xD54, PCI_1 0xDD4	Table 362, p.529
P2P Mem0 Base Address Remap (Low)	PCI_0 0xD5C, PCI_1 0xDDC	Table 363, p.529
P2P Mem0 Base Address Remap (High)	PCI_0 0xD60, PCI_1 0xDE0	Table 364, p.529

Table 329: PCI Slave Address Decoding Register Map (Continued)
NOTE: The PCI_1 register offsets and the P2P registers only apply to the MV64360 and MV64361.

Register	Offsets	Page
P2P Mem1 Base Address Remap (Low)	PCI_0 0xD64, PCI_1 0xDE4	Table 365, p.530
P2P Mem1 Base Address Remap (High)	PCI_0 0xD68, PCI_1 0xDE8	Table 366, p.530
P2P I/O Base Address Remap	PCI_0 0xD6C, PCI_1 0xDEC	Table 367, p.530
PCI CPU Base Address Remap (Low)	PCI_0 0xD70, PCI_1 0xDF0	Table 368, p.530
PCI CPU Base Address Remap (High)	PCI_0 0xD74, PCI_1 0xDF4	Table 369, p.531
Integrated SRAM Base Address Remap	PCI_0 0xF00, PCI_1 0xF80	Table 370, p.531
Expansion ROM Base Address Remap	PCI_0 0xF38, PCI_1 0xFB8	Table 371, p.531
PCI Address Decode Control	PCI_0 0xD3C, PCI_1 0xDBC	Table 372, p.531
Headers Retarget Control	PCI_0 0xF40, PCI_1 0xFC0	Table 373, p.532
Headers Retarget Base	PCI_0 0xF44, PCI_1 0xFC4	Table 374, p.532
Headers Retarget Base (High)	PCI_0 0xF48, PCI_1 0xFC8	Table 375, p.533

Table 330: PCI Control Register Map
NOTE: The PCI_1 register offsets and the P2P registers only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI DLL Status and Control	PCI_0 0x1D20, PCI_1 0x1DA0	Table 376, p.533
PCI/MPP Pads Calibration	PCI_0/MPP[31:16] 0x1D1C, PCI_1/MPP[15:0] 0x1D9C	Table 377, p.535
PCI Command	PCI_0 0xC00, PCI_1 0xC80	Table 378, p.535
PCI Mode	PCI_0 0xD00, PCI_1 0xD80	Table 379, p.538
PCI Retry	PCI_0 0xC04, PCI_1 0xC84	Table 380, p.539
PCI Discard Timer	PCI_0 0xD04, PCI_1 0xD84	Table 381, p.540
MSI Trigger Timer	PCI_0 0xC38, PCI_1 0xCB8	Table 382, p.540
PCI Arbiter Control	PCI_0 0x1D00, PCI_1 0x1D80	Table 383, p.540
PCI Interface Crossbar Control (Low)	PCI_0 0x1D08, PCI_1 0x1D88	Table 384, p.541
PCI Interface Crossbar Control (High)	PCI_0 0x1D0C, PCI_1 0x1D8C	Table 385, p.542
PCI Interface Crossbar Timeout	PCI_0 0x1D04, PCI_1 0x1D84	Table 386, p.543
PCI Sync Barrier Trigger Register	PCI_0 0x1D18, PCI_1 0x1D98	Table 387, p.543
PCI Sync Barrier Virtual Register	PCI_0 0x1D10, PCI_1 0x1D90	Table 387, p.543
PCI P2P Configuration	PCI_0 0x1D14, PCI_1 0x1D94	Table 389, p.544

Table 330: PCI Control Register Map (Continued)

NOTE: The PCI_1 register offsets and the P2P registers only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI Access Control Base 0 (Low)	PCI_0 0x1E00, PCI_1 0x1E80	Table 390, p.544
PCI Access Control Base 0 (High)	PCI_0 0x1E04, PCI_1 0x1E84	Table 391, p.546
PCI Access Control Size 0	PCI_0 0x1E08, PCI_1 0x1E88	Table 392, p.546
PCI Access Control Base 1 (Low)	PCI_0 0x1E10, PCI_1 0x1E90	Table 393, p.546
PCI Access Control Base 1 (High)	PCI_0 0x1E14, PCI_1 0x1E94	Table 394, p.546
PCI Access Control Size 1	PCI_0 0x1E18, PCI_1 0x1E98	Table 395, p.546
PCI Access Control Base 2 (Low)	PCI_0 0x1E20, PCI_1 0x1EA0	Table 396, p.547
PCI Access Control Base 2 (High)	PCI_0 0x1E24, PCI_1 0x1EA4	Table 397, p.547
PCI Access Control Size 2	PCI_0 0x1E28, PCI_1 0x1EA8	Table 398, p.547
PCI Access Control Base 3 (Low)	PCI_0 0x1E30, PCI_1 0x1EB0	Table 399, p.547
PCI Access Control Base 3 (High)	PCI_0 0x1E34, PCI_1 0x1EB4	Table 400, p.547
PCI Access Control Size 3	PCI_0 0x1E38, PCI_1 0x1EB8	Table 401, p.548
PCI Access Control Base 4 (Low)	PCI_0 0x1E40, PCI_1 0x1EC0	Table 402, p.548
PCI Access Control Base 4 (High)	PCI_0 0x1E44, PCI_1 0x1EC4	Table 403, p.548
PCI Access Control Size 4	PCI_0 0x1E48, PCI_1 0x1EC8	Table 404, p.548
PCI Access Control Base 5 (Low)	PCI_0 0x1E50, PCI_1 0x1ED0	Table 405, p.548
PCI Access Control Base 5 (High)	PCI_0 0x1e54, PCI_1 0x1ed4	Table 406, p.549
PCI Access Control Size 5	PCI_0 0x1E58, PCI_1 0x1ED8	Table 407, p.549

Table 331: PCI Configuration Access Register Map

NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI Configuration Address	PCI_0 0xCF8, PCI_1 0xC78	Table 408, p.549
PCI Configuration Data	PCI_0 0xCFC, PCI_1 0xC7C	Table 409, p.550
PCI Interrupt Acknowledge	PCI_0 0xC34, PCI_1 0xCB4	Table 410, p.550

Table 332: PCI Error Report Register Map

NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI SERR# Mask	PCI_0 0xC28, PCI_1 0xCA8	Table 411, p.550
PCI Error Address (Low)	PCI_0 0x1D40, PCI_1 0x1DC0	Table 414, p.554

Table 332: PCI Error Report Register Map (Continued)
NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI Error Address (High)	PCI_0 0x1D44, PCI_1 0x1DC4	Table 415, p.554
PCI Error Attribute	PCI_0 0x1D48, PCI_1 0x1DC8	Table 416, p.554
PCI Error Command	PCI_0 0x1D50, PCI_1 0x1dd0	Table 412, p.552
PCI Interrupt Cause	PCI_0 0x1D58, PCI_1 0x1DD8	Table 412, p.552
PCI Interrupt Mask	PCI_0 0x1D5C, PCI_1 0x1DDC	Table 413, p.553

Table 333: PCI Configuration, Function 0, Register Map
NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI Device and Vendor ID	PCI_0 from CPU or PCI_0 0x00, PCI_0 from PCI_1 0x80, PCI_1 from CPU or PCI_0 0x80, PCI_1 from PCI_1 0x00	Table 418, p.555
PCI Status and Command	PCI_0 from CPU or PCI_0 0x04, PCI_0 from PCI_1 0x84, PCI_1 from CPU or PCI_0 0x84, PCI_1 from PCI_1 0x04	Table 419, p.555
PCI Class Code and Revision ID	PCI_0 from CPU or PCI_0 0x08, PCI_0 from PCI_1 0x88, PCI_1 from CPU or PCI_0 0x88, PCI_1 from PCI_1 0x08	Table 420, p.557
PCI BIST, Header Type/ Init Val, Latency Timer, and Cache Line	PCI_0 from CPU or PCI_0 0x0C, PCI_0 FROM PCI_1 0x8C, PCI_1 from CPU or PCI_0 0x8C, PCI_1 FROM PCI_1 0x0C	Table 421, p.558
PCI CS[0]# Base Address (low)	PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90, PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10	Table 422, p.559
PCI CS[0]# Base Address (high)	PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94, PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14	Table 423, p.559
PCI CS[1]# Base Address (low)	PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98, PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18	Table 424, p.559

Table 333: PCI Configuration, Function 0, Register Map (Continued)
NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI CS[1]# Base Address (high)	PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C, PCI_1 from CPU or PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C	Table 425, p.559
PCI Internal Regs Mem Mapped Base Address (low)	PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0, PCI_1 from CPU or PCI_0 0xA0, PCI_1 from PCI_1 0x20	Table 426, p.560
PCI Internal Regs Mem Mapped Base Address (high)	PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4, PCI_1 from CPU or PCI_0 0xA4, PCI_1 from PCI_1 0x24	Table 427, p.560
PCI Subsystem Device and Vendor ID	PCI_0 from CPU or PCI_0 0x2C, PCI_0 FROM PCI_1 0xAC, PCI_1 from CPU or PCI_0 0xAC, PCI_1 FROM PCI_1 0x2C	Table 428, p.560
PCI Expansion ROM Base Address Register	PCI_0 from CPU or PCI_0 0x30, PCI_0 FROM PCI_1 0xB0, PCI_1 from CPU or PCI_0 0xB0, PCI_1 from PCI_1 0x30	Table 429, p.561
PCI Capability List Pointer Register	PCI_0 from CPU or PCI_0 0x34, PCI_0 from PCI_1 0xB4, PCI_1 FROM CPU OR PCI_0 0xB4, PCI_1 from PCI_1 0x34	Table 430, p.561
PCI Interrupt Pin and Line	PCI_0 from CPU or PCI_0 0x3C, PCI_0 FROM PCI_1 0xBC, PCI_1 FROM CPU OR PCI_0 0xBC, PCI_1 FROM PCI_1 0x3C	Table 431, p.561
PCI Power Management	PCI_0 from CPU or PCI_0 0x40, PCI_0 from PCI_1 0xC0, PCI_1 FROM CPU OR PCI_0 0xC0, PCI_1 from PCI_1 0x40	Table 432, p.561
PCI Power Management Control and Status	PCI_0 from CPU or PCI_0 0x44, PCI_0 from PCI_1 0xC4, PCI_1 FROM CPU OR PCI_0 0xC4, PCI_1 from PCI_1 0x44	Table 433, p.562
PCI VPD Address	PCI_0 from CPU or PCI_0 0x48, PCI_0 from PCI_1 0xC8, PCI_1 FROM CPU OR PCI_0 0xC8, PCI_1 from PCI_1 0x48	Table 434, p.563

Table 333: PCI Configuration, Function 0, Register Map (Continued)
NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI VPD Data	PCI_0 from CPU or PCI_0 0x4C, PCI_0 FROM PCI_1 0xCC, PCI_1 FROM CPU OR PCI_0 0xCC, PCI_1 FROM PCI_1 0x4C	Table 435, p.564
PCI MSI Message Control	PCI_0 from CPU or PCI_0 0x50, PCI_0 from PCI_1 0xD0, PCI_1 FROM CPU OR PCI_0 0xD0, PCI_1 from PCI_1 0x50	Table 436, p.564
PCI MSI Message Address	PCI_0 from CPU or PCI_0 0x54, PCI_0 from PCI_1 0xD4, PCI_1 FROM CPU OR PCI_0 0xD4, PCI_1 from PCI_1 0x54	Table 437, p.565
PCI MSI Message Upper Address	PCI_0 from CPU or PCI_0 0x58, PCI_0 from PCI_1 0xD8, PCI_1 FROM CPU OR PCI_0 0xD8, PCI_1 from PCI_1 0x58	Table 438, p.565
PCI Message Data	PCI_0 from CPU or PCI_0 0x5C, PCI_0 FROM PCI_1 0xDC, PCI_1 FROM CPU OR PCI_0 0xDC, PCI_1 FROM PCI_1 0x5C	Table 439, p.565
PCI-X Command	PCI_0 from CPU or PCI_0 0x60, PCI_0 from PCI_1 0xE0, PCI_1 FROM CPU OR PCI_0 0xE0, PCI_1 from PCI_1 0x60	Table 440, p.565
PCI-X Status	PCI_0 from CPU or PCI_0 0x64, PCI_0 from PCI_1 0xE4, PCI_1 FROM CPU OR PCI_0 0xE4, PCI_1 from PCI_1 0x64	Table 441, p.566
CompactPCI HotSwap	PCI_0 from CPU or PCI_0 0x68, PCI_0 FROM PCI_1 0xE8 PCI_1 FROM CPU OR PCI_0 0xE8, PCI_1 from PCI_1 0x68	Table 442, p.568

Table 334: PCI Configuration, Function 1, Register Map
NOTE: The PCI_1 register offsets and the integrated SRAM registers only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI CS[2]# Base Address (low)	PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90, PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10	Table 443, p.569
PCI CS[2]# Base Address (high)	PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94, PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14	Table 444, p.570
PCI CS[3]# Base Address (low)	PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98, PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18	Table 445, p.570
PCI CS[3]# Base Address (high)	PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C, PCI_1 FROM CPU OR PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C	Table 446, p.570
PCI Integrated SRAM Base Address (low)	PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0, PCI_1 FROM CPU OR PCI_0 0xA0, PCI_1 from PCI_1 0x20	Table 447, p.570
PCI Integrated SRAM Base Address (high)	PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4, PCI_1 FROM CPU OR PCI_0 0xA4, PCI_1 from PCI_1 0x24	Table 448, p.570

Table 335: PCI Configuration, Function 2, Register Map
NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI DevCS[0] Base Address (low)	PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90, PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10	Table 449, p.571
PCI DevCS[0]# Base Address (high)	PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94, PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14	Table 450, p.571
PCI DevCS[1]# Base Address (low)	PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98, PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18	Table 451, p.571

Table 335: PCI Configuration, Function 2, Register Map
NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI DevCS[1]# Base Address (high)	PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C, PCI_1 FROM CPU OR PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C	Table 452, p.572
PCI DevCS[2]# Base Address (low)	PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0, PCI_1 FROM CPU OR PCI_0 0xA0, PCI_1 from PCI_1 0x20	Table 453, p.572
PCI DevCS[2]# Base Address (high)	PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4, PCI_1 FROM CPU OR PCI_0 0xA4, PCI_1 from PCI_1 0x24	Table 454, p.572

Table 336: PCI Configuration, Function 3, Register Map
NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI DevCS[3]# Base Address (Low)	PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90, PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10	Table 455, p.572
PCI DevCS[3]# Base Address (High)	PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94, PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14	Table 456, p.573
PCI BootCS# Base Address (Low)	PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98, PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18	Table 457, p.573
PCI BootCS# Base Address (High)	PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C, PCI_1 FROM CPU OR PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C	Table 458, p.573
PCI CPU Base Address (Low)	PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0, PCI_1 FROM CPU OR PCI_0 0xA0, PCI_1 from PCI_1 0x20	Table 459, p.573

Table 336: PCI Configuration, Function 3, Register Map (Continued)
NOTE: The PCI_1 register offsets only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI CPU Base Address (High)	PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4, PCI_1 FROM CPU OR PCI_0 0xA4, PCI_1 from PCI_1 0x24	Table 460, p.573

Table 337: PCI Configuration, Function 4, Register Map
NOTE: The PCI_1 register offsets and the P2P registers only apply to the MV64360 and MV64361.

Register	Offsets	Page
PCI P2P Mem0 Base Address (Low)	PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90, PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10	Table 461, p.574
PCI P2P Mem0 Base Address (High)	PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94, PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14	Table 462, p.574
PCI P2P Mem1 Base Address (Low)	PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98, PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18	Table 463, p.574
PCI P2P Mem1 Base Address (High)	PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C, PCI_1 FROM CPU OR PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C	Table 464, p.574
PCI P2P I/O Base Address	PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0, PCI_1 FROM CPU OR PCI_0 0xA0, PCI_1 from PCI_1 0x20	Table 465, p.575
PCI Internal Regs I/O Mapped Base Address	PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4, PCI_1 FROM CPU OR PCI_0 0xA4, PCI_1 from PCI_1 0x24	Table 466, p.575

E.3 PCI Slave Address Decoding Registers

Note

The PCI_1 register offsets only apply to the MV64360 and MV64361 devices.

Table 338: CS[0]# BAR Size
Offset: PCI_0 0xC08, PCI_1 0xC88

Bits	Field	Type/ Init Val	Function
11:0	Reserved	RO 0x0	Read only.
31:12	BARSize	RW 0x007FF	CS[0]# BAR Address Bank Size Must be programed from LSB to MSB as sequence of ‘1s’ followed by sequence of ‘0s’. For example, a 0x00FF.F000 size register value represents a BAR size of 16 MB.

Table 339: CS[1]# BAR Size
Offset: PCI_0 0xD08, PCI_1 0xD88

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x007FF000	Same as CS[0] Bar Size.

Table 340: CS[2]# BAR Size
Offset: PCI_0 0xC0C PCI_1 0xC8C

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x007FF000	Same as CS[0] Bar Size.

Table 341: CS[3]# BAR Size
Offset: PCI_0 0xD0C, PCI_1 0xD8C

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x007FF000	Same as CS[0] Bar Size.

Table 342: **DevCS[0]# BAR Size**
Offset: PCI_0 0xC10, PCI_1 0xC90

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x007FF000	Same as CS[0] Bar Size.

Table 343: **DevCS[1]# BAR Size**
Offset: PCI_0 0xD10, PCI_1 0xD90

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x007FF000	Same as CS[0]# Bank Size.

Table 344: **DevCS[2]# BAR Size**
Offset: PCI_0 0xD18, PCI_1 0xD98

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x00FFF000	Same as CS[0]# Bank Size.

Table 345: **DevCS[3]# BAR Size**
Offset: PCI_0 0xC14, PCI_1 0xC94

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x007FF000	Same as CS[0]# Bank Size.

Table 346: **Boot CS# BAR Size**
Offset: PCI_0 0xD14, PCI_1 0xD94

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x007FF000	Same as CS[0]# Bank Size.

Table 347: P2P Mem0 BAR Size
Offset: PCI_0 0xD1C, PCI_1 0xD9C

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x01FFF000	Same as CS[0]# Bank Size

Table 348: P2P Mem1 BAR Size
Offset: PCI_0 0xD20, PCI_1 0xDA0

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x01FF.F000	Same as CS[0]# Bank Size.

Table 349: P2P I/O BAR Size
Offset: PCI_0 0xD24, PCI_1 0xDA4

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x01FFF000	Same as CS[0]# Bank Size.

Table 350: CPU BAR Size
Offset: PCI_0 0xD28, PCI_1 0xDA8

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x01FFF000	Same as CS[0]# Bank Size.

Table 351: Integrated SRAM BAR Size
Offset: PCI_0 0xE00, PCI_1 0xE80

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x0003F000	Same as CS[0]# Bank Size.

Table 352: **Expansion ROM BAR Size**
Offset: PCI_0 0xD2C, PCI_1 0xDAC

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x007FF000	Same as CS[0]# Bank Size.

Table 353: **Base Address Registers Enable**
Offset: PCI_0 0xC3C, PCI_1 0xCBC

Bits	Field	Type/ Init Val	Function
0	CS0En	RW 0x0	CS[0]# BAR Enable 0 = Enable 1 = Disable
1	CS1En	RW 0x0	CS[1]# BAR Enable 0 = Enable 1 = Disable
2	CS2En	RW 0x0	CS[2]# BAR Enable 0 = Enable 1 = Disable
3	CS3En	RW 0x0	CS[3]# BAR Enable 0 = Enable 1 = Disable
4	DevCS0En	RW 0x0	DevCS[0]# BAR Enable 0 = Enable 1 = Disable
5	DevCS1En	RW 0x0	DevCS[1]# BAR Enable 0 = Enable 1 = Disable
6	DevCS2En	RW 0x0	DevCS[2]# BAR Enable 0 = Enable 1 = Disable
7	DevCS3En	RW 0x0	DevCS[3]# BAR Enable 0 = Enable 1 = Disable
8	BootCSEn	RW 0x0	BootCS# BAR Enable 0 = Enable 1 = Disable
9	IntMemEn	RW 0x0	Memory Mapped Internal Registers BAR Enable 0 = Enable 1 = Disable

Table 353: Base Address Registers Enable (Continued)
Offset: PCI_0 0xC3C, PCI_1 0xCBC

Bits	Field	Type/ Init Val	Function
10	IntIOEn	RW 0x1	I/O Mapped Internal Registers BAR Enable 0 = Enable 1 = Disable
NOTE: The MV64360/1/2 prevents disabling both memory mapped and I/O mapped BARs. In other words (bits [9] and [10] cannot simultaneously be set to ‘1’.			
11	P2PMem0En	RW 0x1	P2P Mem0 BAR Enable 0 = Enable 1 = Disable **NOTE:** Only applies to the MV64360 and MV64361. Reserved in the MV64362.
12	P2PMem1En	RW 0x1	P2P Mem1 BAR Enable 0 = Enable 1 = Disable **NOTE:** Only applies to the MV64360 and MV64361. Reserved in the MV64362.
13	P2PIOEn	RW 0x1	P2P IO BAR Enable 0 = Enable 1 = Disable **NOTE:** Only applies to the MV64360 and MV64361. Reserved in the MV64362.
14	CPUEn	RW 0x1	CPU BAR Enable 0 = Enable 1 = Disable
15	IntSRAMEn	RW 0x0	Integrated SDRAM BAR Enable 0 = Enable 1 = Disable **NOTE:** Only applies to the MV64360 and MV64361. Reserved in the MV64362.
31:16	Reserved	RES 0xFFFF	Reserved.

Table 354: CS[0]# Base Address Remap
Offset: PCI_0 0xC48, PCI_1 0xCC8

Bits	Field	Type/ Init Val	Function
11:0	Reserved	RO 0x0	Read only.
31:12	CS0Remap	RW 0x0	CS[0]# BAR Remap Address

**Table 355: CS[1]# Base Address Remap
Offset: PCI_0 0xD48, PCI_1 0xDC8**

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x00800000	Same as CS[0]# Base Address Remap.

**Table 356: CS[2]# Base Address Remap
Offset: PCI_0 0xC4C, PCI_1 0xCCC**

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x01000000	Same as CS[0]# Base Address Remap.

**Table 357: CS[3]# Base Address Remap
Offset: PCI_0 0xD4C, PCI_1 0xDCC**

Bits	Field	Type/ Init Val	Function
31:0	Various	RO RW 0x01800000	Same as CS[0]# Base Address Remap.

**Table 358: DevCS[0]# Base Address Remap
Offset: PCI_0 0xC50, PCI_1 0xCD0**

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x1C000000	Same as CS[0]# Base Address Remap.

**Table 359: DevCS[1]# Base Address Remap
Offset: PCI_0 0xD50, PCI_1 0xDD0**

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x1C800000	Same as CS[0]# Base Address Remap.

Table 360: DevCS[2]# Base Address Remap
Offset: PCI_0 0xD58, PCI_1 0xDD8

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x1D000000	Same as CS[0]# Base Address Remap.

Table 361: DevCS[3]# Base Address Remap
Offset: PCI_0 0xC54, PCI_1 0xCD4

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0xFF000000	Same as CS[0]# Base Address Remap.

Table 362: BootCS# Base Address Remap
Offset: PCI_0 0xD54, PCI_1 0xDD4

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0xFF800000	Same as CS[0]# Base Address Remap.

Table 363: P2P Mem0 Base Address Remap (Low)
Offset: PCI_0 0xD5C, PCI_1 0xDDC

Bits	Field	Type/ Init Val	Function
31:0	Various	RW PCI_0: 0x22000000 PCI_1: 0x12000000	Same as CS[0]# Base Address Remap.

Table 364: P2P Mem0 Base Address Remap (High)
Offset: PCI_0 0xD60, PCI_1 0xDE0

NOTE: Only applies to the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Function
31:0	P2P0Remap	RW 0x0	P2P Mem0 BAR Remap Address.

Table 365: P2P Mem1 Base Address Remap (Low)
Offset: PCI_0 0xD64, PCI_1 0xDE4

NOTE: Only applies to the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Function
31:0	Various	RW PCI_0: 0x24000000 PCI_1: 0xF2000000	Same as CS[0]# Base Address Remap.

Table 366: P2P Mem1 Base Address Remap (High)
Offset: PCI_0 0xD68, PCI_1 0xDE8

NOTE: Only applies to the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Function
31:0	P2P1Remap	RW 0x0	P2P Mem1 BAR Address Remap.

Table 367: P2P I/O Base Address Remap
Offset: PCI_0 0xD6C, PCI_1 0xDEC

NOTE: Only applies to the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Function
31:0	Various	RW PCI_0: 0x20000000 PCI_1: 0x10000000	Same as CS[0]# Base Address Remap.

Table 368: PCI CPU Base Address Remap (Low)
Offset: PCI_0 0xD70, PCI_1 0xDF0

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x40000000	Same as CS[0]# Base Address Remap.

Table 369: PCI CPU Base Address Remap (High)
Offset: PCI_0 0xD74, PCI_1 0xDF4

Bits	Field	Type/ Init Val	Function
3:0	Remap	RW 0x0	Remap Value When the CPU is configured to extended address, corresponds to 60x bus address bits[35:32].
31:4	Reserved	RES 0x0	Reserved.

Table 370: Integrated SRAM Base Address Remap
Offset: PCI_0 0xF00, PCI_1 0xF80

NOTE: Only applies to the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x42000000	Same as CS[0]# Base Address Remap.

Table 371: Expansion ROM Base Address Remap
Offset: PCI_0 0xF38, PCI_1 0xFB8

Bits	Field	Type/ Init Val	Function
31:0	Various	0xFF000000	Same as CS[0]# Base Address Remap.

Table 372: PCI Address Decode Control
Offset: PCI_0 0xD3C, PCI_1 0xDBC

Bits	Field	Type/ Init Val	Function
0	RemapWrDis	RW 0x0	Address Remap Registers Write Disable 0 = Writes to a BAR result in updating the corresponding remap register with the BAR’s new value. 1 = Writes to a BAR have no affect on the corresponding Remap register value.
2:1	Reserved	RES 0x0	
3	MsgAcc	RW 0x1	Messaging registers access 0 = Messaging unit registers are only accessible as part of the MV64360/1/2 internal space and also on the lowest 4 KB of CS[0] BAR space. 1 = Messaging unit registers are only accessible as part of the MV64360/1/2 internal space. **NOTE:** I2O queues are only accessible from PCI via CS[0] BAR space. This bit must be set to ’0’ when using I2O queues.

Table 372: PCI Address Decode Control (Continued)
Offset: PCI_0 0xD3C, PCI_1 0xDBC

Bits	Field	Type/ Init Val	Function
7:4	Reserved	RES 0x0	Reserved.
24:8	VPDHighAddr	RW 0x0	Bits [31:15] of the VPD address.
31:25	Reserved	RES 0x0	Reserved.

Table 373: Headers Retarget Control
Offset: PCI_0 0xF40, PCI_1 0xFC0

Bits	Field	Type/ Init Val	Function
0	En	RW 0x0	Header target enable bit 0x0 = Disable 0x1 = Enable
3:1	Bsize	RW 0x0	Buffer Size 0x0 = 256 bytes 0x1 = 512 Bytes 0x2 = 1KBs 0x3 = 2 KBs 0x4 = 4 KBs 0x5 = 8 KBs 0x6 = 0x7 Reserved
15:4	Reserved	RES 0x0	Reserved.
31:16	Mask1	RW 0x0	Define the total space of the buffers to be manipulated. Size must be set from LSB to MSB as a sequence of 1's, followed by sequence of 0's. Size granularity is 64 KBs. For example, in order to retarget the headers of 1K buffers of 1 KB size, which means 1 MB of buffers space, Mask1 should be set to 0x000f **NOTE:** The total address space of retargeted headers must not exceed integrated SRAM size (256 KB). The minimum buffers space to be manipulated is 64 KB

Table 374: Headers Retarget Base
Offset: PCI_0 0xF44, PCI_1 0xFC4

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RES 0x0	Reserved.

Table 374: Headers Retarget Base
Offset: PCI_0 0xF44, PCI_1 0xFC4

Bits	Field	Type/ Init Val	Function
31:16	Base	RW 0x0	Base Address Retarget is executed if address matches Base. Corresponds to address bits[31:16]

Table 375: Headers Retarget Base (High)
Offset: PCI_0 0xF48, PCI_1 0xFC8

Bits	Field	Type/ Init Val	Function
31:0	Base	RW 0x0	Base address Retarget is executed if address matches base. Corresponds to address bits[63:32] (in case of DAC transaction)

E.4 PCI Control Registers

Table 376: PCI DLL Status and Control
Offset: PCI_0 0x1D20, PCI_1 0x1DA0

Bits	Field	Type/ Init Val	Function
0	En	RO Sampled at reset. PCI_0: DevAD [28:26] PCI_1: DevAD [31:29]	DLL Enable 0 = Disabled 1 = Enabled Read Only.
2:1	Mode	RW Sampled at reset PLL_0: DevAD [31:29] PLL_1: DevAD [28:26]	DLL Mode 0x0 = Use PCLK as reference clock 0x1 = Use inverted PCLK as reference clock 0x2, 0x3 = Reserved
4:3	Err	RW 0x0	DLL Error indication - DLL reach delay line boundary 0x0 = No error 0x1 = Delay line pointer at start, and down count 0x2 = Delay line pointer at end, and up count 0x3 = Reserved

Table 376: PCI DLL Status and Control (Continued)
Offset: PCI_0 0x1D20, PCI_1 0x1DA0

Bits	Field	Type/ Init Val	Function
8:5	RCnt	RO 0x0	Row Counter. Current row number of the delay line. Calculate the TAPs number, using row and column numbers. Read only.
12:9	CCnt	RO 0x0	Column Counter. Current column number of the delay line. Calculate the TAPs number, using row and column numbers. Read only.
14:13	Init	RW Sampled at reset PLL_0: DevAD [31:29] PLL_1: DevAD [28:26]	Delay line initial state. 0x0 = 4 Taps from the beginning of delay line. 0x1 = 8 Taps from the beginning of delay line. 0x2 = 16Taps from the beginning of delay line. 0x3 = 50 Taps from the beginning of delay line.
15	Reserved	RES 0x0	Reserved.
16	CompEn	RW Sampled at reset PLL_0: DevAD [31:29] PLL_1: DevAD [28:26]	Enable Pad Delay Compensation. 0 = Disable 1 = Enable **NOTE:**
19:17	Comp	RW Sampled at reset PLL_0: DevAD [31:29] PLL_1: DevAD [28:26]	Pad Delay Compensation 0x0 is the minimum compensation, 0x7 is the maximum. Only relevant if CompEn bit [16] is set to ’1’. To change the initial reset configuration value, the value must be changed step by step, to enable DLL recovery time.
30:20	Reserved	RES 0x0	Reserved.
31	WrEn	RW 0x0	DLL Status and Control Register Write Enable 0 = The register becomes read only (except of bit[31]). 1 = The register can be written to.

Table 377: PCI/MPP Pads Calibration
Offset: PCI_0/MPP[31:16] 0x1D1C, PCI_1/MPP[15:0] 0x1D9C

Bits	Field	Type/ Init Val	Function
3:0	DrvN	RW 0xF	PCI Pad Nchannel Driving Strength **NOTE:** Only applicable when dynamic tune is disabled.
7:4	DrvP	RW 0xF	PCI Pad Pchannel Driving Strength **NOTE:** Only applicable when dynamic tune is disabled.
11:8	MppDrvN	RW 0xF	MPP Pad Nchannel Driving Strength Useful for changing MPP drive strength after the automatic tuning completes.
15:12	MppDrvP	RW 0xF	MPP Pad Pchannel Driving Strength Useful for changing MPP drive strength after the automatic tuning completes.
16	TuneEn	RW Sampled at reset. PCI_0: DevAD[12] PCI_1: DevAD[13]	Dynamic Tuning of Pad Driving Strength 0 = Disabled 1 = Enabled
17	MPPSetEn	RW 0x1	MPP Drive Strength Enable 0 = Enables changing MPPs drive strength via MppDrvN and MppDrvP. 1 = MPPs drive strength is equal to PCI pads drive strength.
21:18	LockN	RO 0x0	When dynamic tune is enabled, represent the final locked value of the Nchannel Driving Strength. Read Only
25:22	LockP	RO 0x0	When dynamic tune is enabled, represent the final locked value of the Pchannel Driving Strength Read Only
30:26	Reserved	RES 0x0	
31	WrEn	RW 0x0	If set to 1, PCI/MPP Pads Drive Control register is writable. If set to 0, the register becomes read only (except of bit[31])

Table 378: PCI Command
Offset: PCI_0 0xC00, PCI_1 0xC80

Bits	Field	Type/ Init Val	Function
0	MByteSwap	RW	PCI Master Byte Swap When set to ‘0’, the MV64360/1/2 PCI master swaps the bytes of the incoming and outgoing PCI data (swap the 8 bytes of a long-word).
3:1	Reserved	RES 0x0	Read Only.

Table 378: **PCI Command** (Continued)
Offset: PCI_0 0xC00, PCI_1 0xC80

Bits	Field	Type/ Init Val	Function
4	MWrCom	RW 0x1	PCI Master Write Combine Enable When set to '1', write combining is enabled.
5	MRdCom	RW 0x1	PCI Master Read Combine Enable When set to '1', read combining is enabled. **NOTE:** Only relevant for conventional PCI.
6	MWrTrig	RW 0x1	PCI Master Write Trigger 0 = Accesses the PCI bus only when the whole burst is written into the master write buffer. 1 = Accesses the PCI bus when the first data is written into the master write buffer. **NOTE:** Only relevant for conventional PCI.
7	MRdTrig	RW 0x0	PCI Master Read Trigger 0 = Returns read data to the initiating unit only when the whole burst is written into master read buffer. 1 = Returns read data to the initiating unit when the first read data is written into master read buffer. **NOTE:** Must be set to '0' for PCI-X mode.
8	MRdLine	RW 0x1	PCI Master Memory Read Line Enable 0 = Disable 1 = Enable **NOTE:** Only relevant for conventional PCI.
9	MRdMul	RW 0x1	PCI Master Memory Read Multiple Enable 0 = Disable 1 = Enable **NOTE:** Only relevant for conventional PCI.
10	MWordSwap	RW 0x0	PCI Master Word Swap When set to '1', the MV64360/1/2 PCI master swaps the 32-bit words of the incoming and outgoing PCI data.
11	SWordSwap	RW 0x0	PCI Slave Word Swap When set to '1', the MV64360/1/2 PCI slave swaps the bytes of the incoming and outgoing PCI data (swap the two words of a long-word).
12	SRdMode	RW 0x0	PCI-X Mode Only; If set, PCI slave return completion data as soon as its first buffer is valid. **NOTE:** Can be used only with single master which does not issue multiple read requests.
13	Reserved	RES 0x1	
14	MNS	RW 0x1	Master No Snoop (NS) attribute control 0 = Master always drive NS attribute to 0. 1 = Master drives NS attribute according to the NS attribute it receives from the transaction originator. **NOTE:** Only relevant for PCI-X.

Table 378: PCI Command (Continued)
Offset: PCI_0 0xC00, PCI_1 0xC80

Bits	Field	Type/ Init Val	Function
15	MReq64	RW 0x1	PCI Master REQ64# Enable 0 = Disable 1 = Enable
16	SByteSwap	RW	PCI Slave Byte Swap When set to '0', the MV64360/1/2 PCI slave swaps the bytes of the incoming and outgoing PCI data (swap the 8 bytes of a long-word).
17	MDACEn	RW 0x1	PCI Master DAC Enable 0 = The PCI master never drives the DAC cycle. 1 = The PCI master drives the DAC cycle, if the upper 32-bit address is not '0'
18	M64Allign	RW 0x1	PCI Master REQ64# on Burst Access 0 = The PCI master asserts REQ64# on burst access, only if the start address is 6-bit aligned. 1 = The PCI master always asserts REQ64# on burst access, regardless of the start address alignment **NOTE:** Only relevant to conventional PCI. Only applies to the MV64360 and MV64362. Reserved in the MV64361.
19	PErrProp	RW 0x0	Parity/ECC Errors Propagation Enable 0 = The PCI interface always drives correct parity on the PAR signal. 1 = In case of slave read bad ECC from SDRAM, or master write with bad parity/ECC indication from the initiator, the PCI interface drives bad parity on the PAR signal.
20	SSwapEn	RW 0x0	PCI Slave Swap Enable 0 = PCI slave data swapping is determined via SByteSwap and SWordSwap bits (bits 16 and 11). 1 = PCI slave data swapping is determined via `PCISwap` bits [7:6] in the PCI Access Control registers. **NOTE:** If PCI address does not match any of the Access Control windows, the PCI slave data swapping works according to SByteSwap and SWordSwap bits, even if the SSwapEn bit is set to 1.
21	MSwapEn	RW 0x0	PCI Master Swap Enable 0 = PCI master data swapping is determined via MByteSwap and MWordSwap bits (bits 0 and 10). 1 = PCI master data swapping is determined via `PCISwap` bits in the different units Base Address registers.
22	MIntSwapEn	RES 0x0	PCI Master Configuration Transactions Swap Enable 0 = PCI master configuration transaction to the PCI bus is always in Little Endian convention. 1 = PCI master configuration transaction to the PCI bus is determined according to the setting of MSwapEn bit. **NOTE:** Reserved for Marvell Technology usage.

Table 378: PCI Command (Continued)
Offset: PCI_0 0xC00, PCI_1 0xC80

Bits	Field	Type/ Init Val	Function
23	LBEn	RW 0x0	Loop Back Enable 0 = PCI slave never responds to MV64360/1/2 PCI master initiated transactions. 1 = PCI slave responds to MV64360/1/2 PCI master initiated transactions that are targeted to the slave address space. **NOTE:** Loop back mode violates PCI spec AC requirements
25:24	SIntSwap	RW] Bit[25]: 0x0	PCI Slave data swap control on PCI accesses to the MV64360/1/2 internal and configuration registers. 00 = Byte Swap 01 = No swapping 10 = Both byte and word swap 11 = Word swap
27:26	SEndMode	RES 0x0	PCI Slave Endianess Mode Only relevant to PCI-X mode. 00 = Little Endian PCI bus 01 = Big Endian 64-bit PCI bus 10 = Big Endian 32-bit PCI bus 11 = Reserved **NOTE:** Reserved for Marvell usage
28	MEndMode	RES 0x0	PCI Master Endianess Mode 0 = Little Endian PCI bus or 64-bit Big Endian bus 1 = Big Endian 32-bit PCI bus **NOTE:** Reserved for Marvell usage
31:29	Reserved	RO 0x0	Read only.

Table 379: PCI Mode
Offset: PCI_0 0xD00, PCI_1 0xD80

Bits	Field	Type/ Init Val	Function
0	PciID	RO PCI_0: 0x0 PCI_1: 0x1	PCI Interface ID Read Only.
1	Reserved	RES 0x0	Reserved.
2	Pci64	RO Sampled on reset. 64EN#	64-bit PCI Interface When set to ‘1’, the PCI interface is configured to a 64-bit interface. Read Only. **NOTE:** Only applies to the MV64360 and MV64362. Reserved in the MV64361.

Table 379: PCI Mode (Continued)
Offset: PCI_0 0xD00, PCI_1 0xD80

Bits	Field	Type/ Init Val	Function
3	Reserved	RES 0x0	Reserved.
5:4	PciMode	RW Sampled at reset. PCI_0: DEVSEL0#, STOP0#, TRDY0# PCI_1: DEVSEL1#, STOP1#, TRDY1#	PCI interface mode of operation 0x0 = Conventional PCI 0x1 = 66MHz PCI-X 0x2 = 100MHz PCI-X 0x3 = 133MHz PCI-X **NOTE:** Must not be changed after reset. Setting 0x3 is not valid for the MV64360/1/2 industrial grade part.
7:6	Reserved	RES 0x0	Reserved.
8	ExpRom	RO 0x0	Expansion ROM Active When set to ‘1’, the expansion ROM BAR is supported. Read Only from PCI.
30:9	Reserved	RES 0x0	Reserved.
31	PRst	RO 0x1	PCI Interface Reset Indication Set to ‘0’ as long as the PCI RST# pin is asserted. Read Only.

Table 380: PCI Retry
Offset: PCI_0 0xC04, PCI_1 0xC84

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RES 0x0	Reserved
23:16	RetryCtr	RW 0x0	Retry Counter Specifies the number of times the MV64360/1/2 retries a transaction before it quits. Applies to the PCI Master when acting as a requester Applies to the PCI slave when acting as a completer (PCI-X mode). A 0x00 value means a “retry forever”.
31:24	Reserved	RES 0x0	Reserved.

Table 381: PCI Discard Timer
Offset: PCI_0 0xD04, PCI_1 0xD84

NOTE: This register should not be updated while read buffers are not cleared.

Bits	Field	Type/ Init Val	Function
15:0	Timer	RW 0x0	**In conventional PCI mode:** Specifies the number of PCLK cycles the MV64360/1/2 PCI slave keeps a non-accessed read buffers (non-completed delayed read) before invalidating the buffer. Set to '0' to disable the timer. The PCI slave waits for delayed read completion forever. **In PCI-X mode:** Specifies the number of PCLK cycles the MV64360/1/2 PCI master waits for Split completion transaction, before it invalidates the pre-allocated read buffer. Set to '0' to disable the timer. The PCI master waits for split completion forever. **NOTE:** Must be set to a number greater than 0x7f, unless using the "wait for ever" setting 0x0. Must not be updated while there are pending read requests.
31:16	Reserved	RES 0x0	Reserved.

Table 382: MSI Trigger Timer
Offset: PCI_0 0xC38, PCI_1 0xCB8

Bits	Field	Type/ Init Val	Function
15:0	Timer	RW 0xFFFF	Specifies the number of SysClk cycles between consecutive MSI requests.
31:16	Reserved	RES 0x0	Reserved.

Table 383: PCI Arbiter Control
Offset: PCI_0 0x1D00, PCI_1 0x1D80

NOTE: Arbiter setting can not be changed while in work. It should only be set once.

Bits	Field	Type/ Init Val	Function
0	Reserved	RES 0x0	Reserved.
1	BDEn	RW 0x0	Broken Detection Enable If set to '1', broken master detection is enabled. A master is said to be broken if it fails to respond to grant assertion within a window specified in BV field.

Table 383: PCI Arbiter Control (Continued)
Offset: PCI_0 0x1D00, PCI_1 0x1D80

NOTE: Arbiter setting can not be changed while in work. It should only be set once.

Bits	**Field**	**Type/ Init Val**	**Function**
2	Reserved	RES 0x0	Reserved.
6:3	BV	RW 0x6	Broken Value The value sets the maximum number of cycles that the arbiter waits for a PCI master to respond to its grant assertion. If a PCI master fails to assert FRAME# within this time, the PCI arbiter aborts the transaction and performs a new arbitration cycle and a maskable interrupt is generated. **NOTE:** Must be greater than '1' for conventional PCI and greater than '5' for PCI-X.
13:7	Reserved	RES 0x0	Reserved.
20:14	PD[6:0]	RW 0x0	Parking Disable 0 = Enables parking on the associated PCI master. 1 = Disables parking on the associated PCI master. **NOTE:** The arbiter parks on the last master granted unless disabled through the PD bit. Also, if all of the PD bits are set to '1', the PCI arbiter parks on the internal PCI master. PD0 corresponds to the internal master. PD1 corresponds to GNT0. PD2 corresponds to GNT1, and so on
30:21	Reserved	RES 0x0	Reserved.
31	EN	RW 0x0	Enable Internal Arbiter Operation 0 = Disable 1 = Enable

Table 384: PCI Interface Crossbar Control (Low)
Offset: PCI_0 0x1D08, PCI_1 0x1D88

Bits	**Field**	**Type/ Init Val**	**Function**
3:0	Arb0	RW 0x2	Slice 0 of the PCI master "pizza" arbiter. 0x0,0x1 = Reserved 0x2 = CPU access 0x3 = PCI_0: NULL request PCI_1: PCI_0 access 0x4 = PCI_0: PCI_1 access PCI_1: NULL request 0x5 = MPSC access 0x6 = IDMA access 0x7 = Gb Ethernet access 0x8 - 0xF = Reserved **NOTE:** Settings 0x4 only applies to the MV64360 and MV64361. It is reserved in the MV64362.

Table 384: **PCI Interface Crossbar Control (Low)** (Continued)
Offset: PCI_0 0x1D08, PCI_1 0x1D88

Bits	Field	Type/ Init Val	Function
7:4	Arb1	RW PCI_0: 0x4 PCI_1: 0x3	Slice 1 of PCI master “pizza” arbiter.
11:8	Arb2	RW 0x5	Slice 2 of PCI Lunit “pizza” arbiter.
15:12	Arb3	RW 0x6	Slice 3 of PCI Lunit “pizza” arbiter.
19:16	Arb4	RW 0x7	Slice 4 of PCI Lunit “pizza” arbiter.
23:20	Arb5	RW PCI_0: 0x3 PCI_1: 0x4	Slice 5 of PCI Lunit “pizza” arbiter.
27:24	Arb6	RW PCI_0: 0x3 PCI_1: 0x4	Slice 6 of PCI Lunit “pizza” arbiter.
31:28	Arb7	RW PCI_0: 0x3 PCI_1: 0x4	Slice 7 of PCI Lunit “pizza” arbiter.

Table 385: **PCI Interface Crossbar Control (High)**
Offset: PCI_0 0x1D0C, PCI_1 0x1D8C

Bits	Field	Type/ Init Val	Function
3:0	Arb8	RW 0x2	Slice 8 of PCI Lunit “pizza” arbiter.
7:4	Arb9	RW PCI_0: 0x4 PCI_1: 0x3	Slice 9 of PCI Lunit “pizza” arbiter.
11:8	Arb10	RW 0x5	Slice 10 of PCI Lunit “pizza” arbiter.
15:12	Arb11	RW 0x6	Slice 11 of PCI Lunit “pizza” arbiter.
19:16	Arb12	RW 0x7	Slice 12 of PCI Lunit “pizza” arbiter.
23:20	Arb13	RW PCI_0: 0x3 PCI_1: 0x4	Slice 13 of PCI Lunit “pizza” arbiter.

Table 385: PCI Interface Crossbar Control (High) (Continued)
Offset: PCI_0 0x1D0C, PCI_1 0x1D8C

Bits	Field	Type/ Init Val	Function
27:24	Arb14	RW PCI_0: 0x3 PCI_1: 0x4	Slice 14 of PCI Lunit “pizza” arbiter.
31:28	Arb15	RW PCI_0: 0x3 PCI_1: 0x4	Slice 15 of PCI Lunit “pizza” arbiter.

Table 386: PCI Interface Crossbar Timeout
Offset: PCI_0 0x1D04, PCI_1 0x1D84

Bits	Field	Type/ Init Val	Function
7:0	Timeout	RW 0xFF	CrossBar Arbiter Timeout Preset Value
15:8	Reserved	RES 0x0	Reserved.
16	TimeoutEn	RW 0x1	CrossBar Arbiter Timer Enable 0 = Enable 1 = Disable
31:17	Reserved	RES 0x0	Reserved.

Table 387: PCI Sync Barrier Trigger Register
Offset: PCI_0 0x1D18, PCI_1 0x1D98

Bits	Field	Type/ Init Val	Function
31:0	Trig	RO 0x0	PCI write to this register (any value) triggers sync barrier action. Read has no affect. Read Only 0x0.

Table 388: PCI Sync Barrier Virtual Register
Offset: PCI_0 0x1D10, PCI_1 0x1D90

Bits	Field	Type/ Init Val	Function
31:0	Sync	RO 0x0	PCI read from this register returns data of 0xFFFF.FFFF as long as sync barrier is not resolved. Once resolved, read returns 0x0. Write has no affect. Read Only.

Table 389: PCI P2P Configuration
Offset: PCI_0 0x1D14, PCI_1 0x1D94

Bits	Field	Type/ Init Val	Function
7:0	2ndBusL	RW 0xFF	Secondary PCI Interface Bus Range Lower Boundary
15:8	2ndBusH	RW 0x0	Secondary PCI Interface Bus Range Upper Boundary
23:16	BusNumber	RW/RO Conven-tional PCI: 0x0 PCI-X: 0xFF	The PCI bus number to which the PCI interface is connected to. **NOTE:** Read Only in PCI-X mode (copy of PCI-X Status and Command register's BN field).
28:24	DevNum	RW/RO Conven-tional PCI: 0x0 PCI-X: 0x1f	The PCI interface's Device number **NOTE:** Read Only in PCI-X mode (copy of PCI-X Status and Command register's DN field).
31:29	Reserved	RES 0x0	Reserved

Table 390: PCI Access Control Base 0 (Low)
Offset: PCI_0 0x1E00, PCI_1 0x1E80

Bits	Field	Type/ Init Val	Function
0	En	RW 0x0	Access control window enable 0 = Disable 1 = Enable
1	Req64	RW 0x0	Force REQ64 0 = Disable PCI master REQ64 policy is based on transaction size. 1 = Enable PCI master always asserts REQ64#. Only relevant if the target is the peer PCI interface (P2P memory trans-action). **NOTE:** Only applies to the MV64360. Reserved in the MV64361 and MV64362.

Table 390: PCI Access Control Base 0 (Low) (Continued)
Offset: PCI_0 0x1E00, PCI_1 0x1E80

Bits	Field	Type/ Init Val	Function
3:2	Snoop	RW 0x0	Cache Coherency support 00 = Non cache coherent region 01 = Cache coherent WT region 10 = Cache coherent WB region 11 = Reserved **NOTE:** Only relevant if the target is SDRAM. In PCI-X mode - if bit[30] in the attribute phase (no Snoop) is asserted, these bit are ignored.
4	AccProt	RW 0x0	Access Protect 0 = PCI access to this region is allowed. 1 = PCI access to this region is forbidden.
5	WrProt	RW 0x0	Write Protect 0 = PCI write is to this region is allowed. 1 = PCI write to this region is forbidden
7:6	PCISwap	RW 0x0	PCI slave Data Swap Control 00 = Byte Swap 01 = No swapping 10 = Both byte and word swap 11 = Word swap
9:8	MBurst	RW 0x0	Max Burst Specifies the maximum burst size for a single transaction between a PCI slave and the other interfaces. 00 = 32 bytes 01 = 64 bytes 10 = 128 bytes 11 = Reserved **NOTE:** In case of cache coherent region (bit[3:2] are 0x1 or 0x2), MBurst must be set to 00.
11:10	RdSize	RW 0x0	Typical PCI read transaction Size. Defines the amount of data the slave prefetch from the target unit: 00 = 32 bytes 01 = 64 bytes 10 = 128 bytes 11 = 256 bytes Only relevant to conventional PCI mode **NOTE:** If the transaction address hits a non prefetchable space (prefetch bit of the BAR is cleared), the slave reads a single data, regardless of RdSize setting.
31:12	Base	RW 0x0	Access Control Base Address Corresponds to address bits[31:12].

Table 391: PCI Access Control Base 0 (High)
Offset: PCI_0 0x1E04, PCI_1 0x1E84

Bits	Field	Type/ Init Val	Function
31:0	Base	RW 0x0	Base Address High. Corresponds to address bits[63:32].

Table 392: PCI Access Control Size 0
Offset: PCI_0 0x1E08, PCI_1 0x1E88

Bits	Field	Type/ Init Val	Function
11:0	Reserved	RES 0x0	Reserved.
31:12	Size	RW 0x0	PCI access window size Must be programed from LSB to MSB as sequence of ‘1s’ followed by sequence of ‘0s’. For example, a 0x00FF.F000 size register value represents a window size of 16 MB.

Table 393: PCI Access Control Base 1 (Low)
Offset: PCI_0 0x1E10, PCI_1 0x1E90

Bits	Field	Type/ Init Val	Function
31:0	Various	RW RES 0x0	Same as in Access Control Base 0 (Low).

Table 394: PCI Access Control Base 1 (High)
Offset: PCI_0 0x1E14, PCI_1 0x1E94

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as in Access Control Base 0 (High).

Table 395: PCI Access Control Size 1
Offset: PCI_0 0x1E18, PCI_1 0x1E98

Bits	Field	Type/ Init Val	Function
31:0	Various	RES RW 0x0	Same as in Access Control Base Top 0.

Table 396: PCI Access Control Base 2 (Low)
Offset: PCI_0 0x1E20, PCI_1 0x1EA0

Bits	Field	Type/ Init Val	Function
31:0	Various	RW RES 0x0	Same as in Access Control Base 0 (Low).

Table 397: PCI Access Control Base 2 (High)
Offset: PCI_0 0x1E24, PCI_1 0x1EA4

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as in Access Control Base 0 (High).

Table 398: PCI Access Control Size 2
Offset: PCI_0 0x1E28, PCI_1 0x1EA8

Bits	Field	Type/ Init Val	Function
31:0	Various	RES RW 0x0	Same as in Access Control Base 0.

Table 399: PCI Access Control Base 3 (Low)
Offset: PCI_0 0x1E30, PCI_1 0x1EB0

Bits	Field	Type/ Init Val	Function
31:0	Various	RW RES 0x0	Same as in Access Control Base 0 (Low).

Table 400: PCI Access Control Base 3 (High)
Offset: PCI_0 0x1E34, PCI_1 0x1EB4

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as in Access Control Base 0 (High).

Table 401: PCI Access Control Size 3
Offset: PCI_0 0x1E38, PCI_1 0x1EB8

Bits	Field	Type/ Init Val	Function
31:0	Various	RES RW 0x0	Same as in Access Control Base 0.

Table 402: PCI Access Control Base 4 (Low)
Offset: PCI_0 0x1E40, PCI_1 0x1EC0

Bits	Field	Type/ Init Val	Function
31:0	Various	RW RES 0x0	Same as in Access Control Base 0 (Low).

Table 403: PCI Access Control Base 4 (High)
Offset: PCI_0 0x1E44, PCI_1 0x1EC4

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as in Access Control Base 0 (High).

Table 404: PCI Access Control Size 4
Offset: PCI_0 0x1E48, PCI_1 0x1EC8

Bits	Field	Type/ Init Val	Function
31:0	Various	RES RW 0x0	Same as in Access Control Base 0.

Table 405: PCI Access Control Base 5 (Low)
Offset: PCI_0 0x1E50, PCI_1 0x1ED0

Bits	Field	Type/ Init Val	Function
31:0	Various	RW RES 0x0	Same as in Access Control Base 0 (Low).

Table 406: PCI Access Control Base 5 (High)
Offset: PCI_0 0x1e54, PCI_1 0x1ed4

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as in Access Control Base 0 (High).

Table 407: PCI Access Control Size 5
Offset: PCI_0 0x1E58, PCI_1 0x1ED8

Bits	Field	Type/ Init Val	Function
31:0	Various	RES RW 0x0	Same as in Access Control Base 0.

E.5 PCI Configuration Access Registers

Notes

- PCI Configuration Address, PCI Configuration Data, and PCI Interrupt Acknowledge registers are only accessible from the CPU. They must not be accessed from the PCI.
- The PCI_1 register offsets only apply to the MV64360 and MV64361.

Table 408: PCI Configuration Address
Offset: PCI_0 0xCF8, PCI_1 0xC78

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Read Only.
7:2	RegNum	RW 0x0	Register number.
10:8	FunctNum	RW 0x0	Function number.
15:11	DevNum	RW 0x0	Device number.
23:16	BusNum	RW 0x0	Bus number.
30:24	Reserved	RES 0x0	Read Only.
31	ConfigEn	RW 0x0	When set, an access to the Configuration Data register is translated into a Configuration or Special cycle on the PCI bus.

Table 409: PCI Configuration Data
Offset: PCI_0 0xCFC, PCI_1 0xC7C

Bits	Field	Type/ Init Val	Function
31:0	ConfigData	RW 0x0	The data is transferred to/from the PCI bus when the CPU accesses this register and the ConfigEn bit in the Configuration Address register is set. A CPU access to this register causes the MV64360/1/2 to perform a Configuration or Special cycle on the PCI bus.

Table 410: PCI Interrupt Acknowledge
Offset: PCI_0 0xC34, PCI_1 0xCB4

Bits	Field	Type/ Init Val	Function
31:0	IntAck	RO 0x0	A CPU read access to this register forces an interrupt acknowledge cycle on the PCI bus. **NOTE:** This register is READ ONLY.

E.6 PCI Error Report Registers

Note

The PCI_1 register offsets only apply to the MV64360 and MV64361.

Table 411: PCI SERR# Mask
Offset: PCI_0 0xC28, PCI_1 0xCA8

NOTE: MV64360/1/2 only asserts SERR# if the PCI Status and Command register's `SErrEn` bit [8] is enabled, see Table 419 on page 555.

Bits	Field	Type/ Init Val	Function
0	Reserved	RES 0x0	Reserved
1	SWrPerr	RW 0x0	If set to '1', asserts SERR# upon PCI slave detection of bad write data parity.
2	SRdPerr	RW 0x0	If set to '1', asserts SERR# upon a PERR# response to read data driven by the PCI slave.
3	Reserved	RES 0x0	Reserved
4	MIOErr	RW 0x0	If set to '1', asserts SERR# upon an attempt to issue an I/O transaction that crosses DWORD boundary. **NOTE:** Only applies to PCI-X mode.
5	MWrPerr	RW 0x0	If set to '1', asserts SERR# upon a PERR# response to write data driven by the PCI master.

Table 411: PCI SERR# Mask (Continued)
Offset: PCI_0 0xC28, PCI_1 0xCA8

NOTE: MV64360/1/2 only asserts SERR# if the PCI Status and Command register's `SErrEn` bit [8] is enabled, see Table 419 on page 555.

Bits	Field	Type/ Init Val	Function
6	MRdPerr	RW 0x0	If set to '1', asserts SERR# upon a bad data parity detection during a PCI master read transaction or upon bad parity detection during split completion to a write transaction initiated by the master.
7	MCTabort	RW 0x0	If set to '1', assert SERR# upon Master generates Target Abort for split completion. **NOTE:** Only applies to PCI-X mode.
8	MMabort	RW 0x0	If set to '1', asserts SERR# upon a PCI master generation of master abort.
9	MTabort	RW 0x0	If set to '1', asserts SERR# upon a PCI master detection of target abort.
10	MDis	RW 0x0	If set to '1', assert SERR# upon an attempt to generate a PCI transaction while master is disabled.
11	MRetry	RW 0x0	If set to '1', asserts SERR# upon a PCI master reaching retry counter limit.
12	MDiscard	RW 0x0	If set to '1', asserts SERR# upon a PCI master Split Completion Discard Timer expiration. **NOTE:** Only applies to PCI-X mode.
13	MUnExp	RW 0x0	If set to '1', asserts SERR# upon a PCI master receives Split Completion which its byte count does no much the original request. **NOTE:** Only applies to PCI-X mode.
14	MErrMsg	RW 0x0	If set to '1', asserts SERR# upon a PCI master receiving Massage error. **NOTE:** Only applies to PCI-X mode.
15	Reserved	RES 0x0	Reserved
16	SCMabort	RW 0x0	If set to '1', asserts SERR# upon a PCI slave generate master abort as a completer. **NOTE:** Only applies to PCI-X mode.
17	STabort	RW 0x0	If set to '1', asserts SERR# upon a PCI slave generate Target Abort.
18	SCTabort	RW 0x0	If set to '1', asserts SERR# upon a PCI slave as a completer, detection of target abort. **NOTE:** Only applies to PCI-X mode.
19	Reserved	RES 0x0	Reserved

Table 411: PCI SERR# Mask (Continued)
Offset: PCI_0 0xC28, PCI_1 0xCA8

NOTE: MV64360/1/2 only asserts SERR# if the PCI Status and Command register's `SErrEn` bit [8] is enabled, see Table 419 on page 555.

Bits	Field	Type/ Init Val	Function
20	SRdBuf	RW 0x0	If set to '1', asserts SERR# if the PCI slave's read buffer, discard timer expires. **NOTE:** Only applies to conventional PCI mode.
21	Arb	RW 0x0	If set to '1', asserts SERR# upon the internal PCI arbiter detection of a "broken" PCI master.
22	SRetry	RW 0x0	If set to '1', asserts SERR# upon a PCI slave reaching the retry counter limit when attempting to generate a Split Completion. **NOTE:** Only applies to PCI-X mode.
23	SCDestRd	RW 0x0	If set to '1', asserts SERR# upon PCI slave Split Completion transaction from a non prefetchable BAR space, that is terminated with Master or Target abort. Indicates that a destructive read event happened (since the read buffer is discarded).
31:24	Reserved	RES 0x0	Reserved.

Table 412: PCI Interrupt Cause
Offset: PCI_0 0x1D58, PCI_1 0x1DD8

NOTE: All bits are Read/Write Clear only. A cause bit sets upon error event occurrence. A write of '0' clears the bit. A write of '1' has no affect.

Bits	Field	Type/ Init Val	Function
23:0	Cause	RWC 0x0	Cause bit per event, as described in the SERR# Mask register.
24	BIST	RWC 0x0	PCI BIST Interrupt
25	PMG	RWC 0x0	PCI Power Management Interrupt

Table 412: PCI Interrupt Cause (Continued)
Offset: PCI_0 0x1D58, PCI_1 0x1DD8

NOTE: All bits are Read/Write Clear only. A cause bit sets upon error event occurrence. A write of '0' clears the bit. A write of '1' has no affect.

Bits	Field	Type/ Init Val	Function
26	PRST	RWC 0x0	PCI Reset Assert
31:27	Sel	RWC 0x0	Specifies the error event currently being reported in the Error Address registers: 0x0 = Reserved 0x1 = SWrPerr 0x2 = SRdPerr 0x3 - 0x4 = Reserved 0x5 = MWrPerr 0x6 = MRdPerr 0x7 = MCTabort 0x8 = MMabort 0x9 = MTabort 0xA = Reserved 0xB = MRetry 0xC = Reserved 0xD = MUnExp 0xE = MErrMsg 0xF = Reserved 0x10 = SCMabort 0x11 = STabort 0x12 - SCTabort 0x13 - 0x1f = Reserved

Table 413: PCI Interrupt Mask
Offset: PCI_0 0x1D5C, PCI_1 0x1DDC

Bits	Field	Type/ Init Val	Function
26:0	Mask	RW 0x0	Mask bit per cause bit. If a bit is set to 1, the corresponding event is enabled. Mask does not affect setting of the Interrupt Cause register bits, it only affects the assertion of interrupt.
31:27	Reserved	RES 0x0	Reserved.

Table 414: PCI Error Address (Low)
Offset: PCI_0 0x1D40, PCI_1 0x1DC0

Bits	Field	Type/ Init Val	Function
31:0	ErrAddr	RO 0x0	PCI address bits[31:0] are latched as a result of an error latched in the Interrupt Cause Register. Upon address latched, no new address can be registered (due to another error) until the register is being read. An error which is masked in the Interrupt Mask Register will not set this register. **NOTE:** Read Only.

Table 415: PCI Error Address (High)
Offset: PCI_0 0x1D44, PCI_1 0x1DC4

Bits	Field	Type/ Init Val	Function
31:0	ErrAddr	RW 0x0	PCI address bits[63:32] are latched as a result of an error latched in the Interrupt Cause Register. Upon address latched, no new address can be registered (due to another error) until the Address Low register is being read. An error which is masked in the Interrupt Mask Register will not set this register. **NOTE:** Applicable only when running DAC cycle.

Table 416: PCI Error Attribute
Offset: PCI_0 0x1D48, PCI_1 0x1DC8

Bits	Field	Type/ Init Val	Function
31:0	ErrAttr	RW 0x0	PCI-X Attribute bits[31:0] are latched as a result of an error latched in the Interrupt Cause Register. Upon address latched, no new address can be registered (due to another error) until the Address Low register is being read. An error which is masked in the Interrupt Mask Register will not set this register. **NOTE:** Applies to PCI-X only.

Table 417: PCI Error Command
Offset: PCI_0 0x1D50, PCI_1 0x1dd0

Bits	Field	Type/ Init Val	Function
3:0	ErrCmd	RW 0x0	PCI Command bits[3:0] are latched as a result of an error latched in the Interrupt Cause Register. When latched, no new command can be registered (due to another error) until the Address Low register is being read. An error which is masked in the Interrupt Mask Register will not set this register.

Table 417: PCI Error Command
Offset: PCI_0 0x1D50, PCI_1 0x1dd0

Bits	Field	Type/ Init Val	Function
4	DAC	RW 0x0	If set to '1', indicates that the transaction latched in the error registers is a DAC transaction (meaning the address is composed of Error Address Low and High registers).
31:5	Reserved	RES 0x0	Reserved

Note

Error Address is not latched with the following interrupt events - `MIllegal` bit[10], `MDiscard` bit[12], `SRdBuf` bit[20], `Arb` bit[21], `SRetry` bit[22], `BIST` bit[24], `PMG` bit[25], and `PRST` bit[26].

E.7 Function 0 Configuration Registers

Note

The PCI_1 register offsets only apply to the MV64360 and MV64361.

Table 418: PCI Device and Vendor ID
Offset: PCI_0 from CPU or PCI_0 0x00, PCI_0 from PCI_1 0x80,
PCI_1 from CPU or PCI_0 0x80, PCI_1 from PCI_1 0x00

Bits	Field	Type/ Init Val	Function
15:0	VenID	RW 0x11AB	Marvell's Vendor ID. Read only from PCI.
31:16	DevID	RW 0x6460	MV64360/1/2 Device ID. Read only from PCI.

Table 419: PCI Status and Command
Offset: PCI_0 from CPU or PCI_0 0x04, PCI_0 from PCI_1 0x84,
PCI_1 from CPU or PCI_0 0x84, PCI_1 from PCI_1 0x04

Bits	Field	Type/ Init Val	Function
0	IOEn	RW 0x0	Controls the MV64360/1/2's ability to response to PCI I/O accesses. 0 = Disable 1 = Enable
1	MEMEn	RW 0x0	Controls the MV64360/1/2's ability to response to PCI Memory accesses. 0 = Disable 1 = Enable

Table 419: PCI Status and Command (Continued)
Offset: PCI_0 from CPU or PCI_0 0x04, PCI_0 from PCI_1 0x84, PCI_1 from CPU or PCI_0 0x84, PCI_1 from PCI_1 0x04

Bits	Field	Type/ Init Val	Function
2	MasEn	RW 0x0	Controls the MV64360/1/2's ability to act as a master on the PCI bus. 0 = Disable 1 = Enable
3	SpecialEn	RO 0x0	Controls the MV64360/1/2's ability to respond to PCI special cycles. Read only '0' (MV64360/1/2 PCI slave does not support special cycles).
4	MemWrInv	RW 0x0	PCI: Controls the MV64360/1/2's ability to generate memory write and invalidate commands on the PCI bus. 0 = Disable 1 = Enable PCI-X: Ignored
5	VGA	RO 0x0	VGA Palette Snoops Not supported. Read only '0'.
6	PErrEn	RW 0x0	Controls the MV64360/1/2's ability to respond to parity errors on the PCI. If this bit is disable the MV64360/1/2 ignores any Parity Errors. 0 = Disable 1 = Enable **NOTE:** The MV64360/1/2 asserts PERR# only when detects data parity error.
7	AddrStep	RW 0x0	PCI: Address Stepping Enable The MV64360/1/2 PCI master performs address stepping only on configuration accesses. PCI-X: Ignored **NOTE:** Read only from the CPU and PCI.
8	SErrEn	RW 0x0	Controls the MV64360/1/2's ability to assert the SERR# pin. 0 = Disable 1 = Enable
9	FastBTBEn	RW 0x0	Controls the MV64360/1/2's ability to generate fast back-to-back transactions. 0 = Disable 1 = Enable **NOTE:** Not relevant in PCI-X mode
19:10	Reserved	RES 0x0	Read only.
20	CapList	RW 0x1	Capability List Support Indicates that the MV64360/1/2 configuration header includes capability list. **NOTE:** Read only from the PCI.

Table 419: PCI Status and Command (Continued)
Offset: PCI_0 from CPU or PCI_0 0x04, PCI_0 from PCI_1 0x84, PCI_1 from CPU or PCI_0 0x84, PCI_1 from PCI_1 0x04

Bits	Field	Type/ Init Val	Function
21	66MHzEn	RW 0x1	66MHz Capable Indicates that the MV64360/1/2 PCI interface is capable of running at 66MHz **NOTE:** Read only from PCI.
22	Reserved	RES 0x0	Read only.
23	TarFastBB	RW 0x1	Indicates that the MV64360/1/2 is capable of accepting fast back-to-back transactions on the PCI bus. **NOTE:** Read only from the PCI.
24	DataPerr	RWC 0x0	Set by the MV64360/1/2 Master when detects or generates Perr. In PCI-X mode, also set by the MV64360/1/2 Master when receiving Split Error Massage of Parity error detection. Clear only by writing ‘1’.
26:25	DevSelTim	RW 0x1	Indicates the MV64360/1/2‘s DEVSEL timing (medium). **NOTE:** Read only from the PCI.
27	SlaveTabort	RWC 0x0	Set when the MV64360/1/2’s slave terminates a transaction with Target Abort. Clear only by writing ‘1’.
28	MasterTabort	RWC 0x0	Set when the MV64360/1/2’s master detects a Target Abort termination. Clear only by writing ‘1’.
29	MAbort	RWC 0x0	Set when the MV64360/1/2’s master generates a Master Abort (except of special cycle). Clear only by writing ‘1’.
30	SysErr	RWC 0x0	Set when the MV64360/1/2 asserts SERR#. Clear only by writing ‘1’.
31	DetParErr	RWC 0x0	Set upon the MV64360/1/2 detection of Parity error (both as master and slave). Clear only by writing ‘1’.

Table 420: PCI Class Code and Revision ID
Offset: PCI_0 from CPU or PCI_0 0x08, PCI_0 from PCI_1 0x88, PCI_1 from CPU or PCI_0 0x88, PCI_1 from PCI_1 0x08

Bits	Field	Type/ Init Val	Function
7:0	RevID	RW 0x1	Indicates the MV64360/1/2 Revision number. **NOTE:** Read only from PCI.
15:8	Reserved	RES 0x0	Read only.

Table 420: PCI Class Code and Revision ID (Continued)
Offset: PCI_0 from CPU or PCI_0 0x08, PCI_0 from PCI_1 0x88,
PCI_1 from CPU or PCI_0 0x88, PCI_1 from PCI_1 0x08

Bits	Field	Type/ Init Val	Function
23:16	SubClass	RW 0x80	Indicates the MV64360/1/2 Subclass. **NOTE:** Read only from PCI.
31:24	BaseClass	RW 0x05	Indicates the MV64360/1/2 Base Class. **NOTE:** Read only from PCI.

Table 421: PCI BIST, Header Type/
Init Val, Latency Timer, and Cache Line
Offset: PCI_0 from CPU or PCI_0 0x0C, PCI_0 FROM PCI_1 0x8C,
PCI_1 from CPU or PCI_0 0x8C, PCI_1 FROM PCI_1 0x0C

Bits	Field	Type/ Init Val	Function
7:0	CacheLine	RW 0x00	Specifies the MV64360/1/2's cache line size.
15:8	LatTimer	Reset Configuration Conventional PCI Mode: 0x0 PCI-x Mode: 0x40	Specifies in units of PCI clock cycles the latency timer value of the MV64360/1/2. Used by the PCI master when acting as a requester. In PCI-X mode, also used by the PCI slave when acting as a completer.
23:16	HeadType/ Init Val	RW 0x80	Specifies Configuration Header Type/ Init Val Read only from PCI.
27:24	BISTComp	RW 0x0	BIST Completion Code Written by the CPU upon BIST completion. Read only from PCI.
29:28	Reserved	RES 0x0	Reserved.
30	BISTAct	RW 0x0	BIST Activate bit Set to '1' by PCI to activate BIST. Cleared by CPU upon BIST completion.
31	BISTCap	RW 0x1	BIST Capable Bit **NOTE:** Read Only from PCI.

Table 422: PCI CS[0]# Base Address (low)
Offset: PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90, PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10

Bits	Field	Type/ Init Val	Function
0	MemSpace	RW 0x0	Memory Space Indicator **NOTE:** Read only from PCI.
2:1	Type/ Init Val	RW 0x2	BAR Type/ Init Val. Locate anywhere in 64-bit address space **NOTE:** Read only from PCI.
3	Prefetch	RW 0x1	Prefetch Enable **NOTE:** Read only from PCI.
11:4	Reserved	RES 0x0	Reserved.
31:12	Base	RW 0x00000	Base address. Corresponds to address bits[31:12]

Table 423: PCI CS[0]# Base Address (high)
Offset: PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94, PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14

Bits	Field	Type/ Init Val	Function
31:0	Base	RW 0x0	Base address. Corresponds to address bits[63:32].

Table 424: PCI CS[1]# Base Address (low)
Offset: PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98, PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0080000C	Same as CS[0]# Base Address (low).

Table 425: PCI CS[1]# Base Address (high)
Offset: PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C, PCI_1 from CPU or PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

Table 426: PCI Internal Regs Mem Mapped Base Address (low)
**Offset: PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0,
PCI_1 from CPU or PCI_0 0xA0, PCI_1 from PCI_1 0x20**

Bits	Field	Type/ Init Val	Function
0	MemSpace	RO 0x0	Memory Space Indicator
2:1	Type/Init Val	RW 0x2	BAR Type/Init Val Located anywhere in the 64-bit address space. **NOTE:** Read only from PCI.
3	Prefetch	RW 0x0	Prefetch Enable **NOTE:** Read only from PCI.
15:4	Reserved	RES 0x0	Read only.
31:16	Base	RW 0x1400	Base Address Corresponds to address bits[31:16].

Table 427: PCI Internal Regs Mem Mapped Base Address (high)
**Offset: PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4,
PCI_1 from CPU or PCI_0 0xA4, PCI_1 from PCI_1 0x24**

Bits	Field	Type/ Init Val	Function
31:0	Base	RW 0x0	Base address Corresponds to address bits[63:32].

Table 428: PCI Subsystem Device and Vendor ID
**Offset: PCI_0 from CPU or PCI_0 0x2C, PCI_0 FROM PCI_1 0xAC,
PCI_1 from CPU or PCI_0 0xAC, PCI_1 FROM PCI_1 0x2C**

Bits	Field	Type/ Init Val	Function
15:0	VenID	RW 0x0	Subsystem Manufacturer ID Number **NOTE:** Read only from PCI.
31:16	DevID	RW 0x0	Subsystem Device ID Number **NOTE:** Read only from PCI.

Table 429: PCI Expansion ROM Base Address Register
Offset: PCI_0 from CPU or PCI_0 0x30, PCI_0 FROM PCI_1 0xB0,
PCI_1 from CPU or PCI_0 0xB0, PCI_1 from PCI_1 0x30

NOTE: If Expansion ROM is not enabled via reset configuration, this register is Reserved (Read Only 0)

Bits	Field	Type/ Init Val	Function
0	ExpROMEn	RW 0x0	Expansion ROM Enable 0 = Disable 1 = Enable
11:1	Reserved	RES 0x0	Reserved.
31:12	Base	RW 0xFF000	Expansion ROM Base Address

Table 430: PCI Capability List Pointer Register
Offset: PCI_0 from CPU or PCI_0 0x34, PCI_0 from PCI_1 0xB4,
PCI_1 FROM CPU OR PCI_0 0xB4, PCI_1 from PCI_1 0x34

Bits	Field	Type/ Init Val	Function
7:0	CapPtr	RW 0x40	Capability List Pointer **NOTE:** Read only from PCI.
31:8	Reserved	RES 0x0	Reserved.

Table 431: PCI Interrupt Pin and Line
Offset: PCI_0 from CPU or PCI_0 0x3C, PCI_0 FROM PCI_1 0xBC,
PCI_1 FROM CPU OR PCI_0 0xBC, PCI_1 FROM PCI_1 0x3C

Bits	Field	Type/ Init Val	Function
7:0	IntLine	RW 0x0	Provides interrupt line routing information.
15:8	IntPin	RW 0x1	Indicates which interrupt pin is used by the MV64360/1/2. **NOTE:** Read only from PCI.
31:16	Reserved	RES 0x0	Read only.

Table 432: PCI Power Management
Offset: PCI_0 from CPU or PCI_0 0x40, PCI_0 from PCI_1 0xC0,
PCI_1 FROM CPU OR PCI_0 0xC0, PCI_1 from PCI_1 0x40

Bits	Field	Type/ Init Val	Function
7:0	CapID	RW 0x1	Capability ID **NOTE:** Read only from PCI.

Table 432: PCI Power Management (Continued)
Offset: PCI_0 from CPU or PCI_0 0x40, PCI_0 from PCI_1 0xC0,
PCI_1 FROM CPU OR PCI_0 0xC0, PCI_1 from PCI_1 0x40

Bits	Field	Type/ Init Val	Function
15:8	NextPtr	RW 0x48	Next Item Pointer **NOTE:** Read only from PCI.
18:16	PMCVer	RW 0x1	PCI Power Management Spec Revision **NOTE:** Read only from PCI.
19	PMEClk	RW 0x1	PME Clock Indicates that the PCI clock is required for the MV64360/1/2 to assert PME# **NOTE:** Read only from PCI.
20	Reserved	RES 0x0	Read only from PCI.
21	DSI	RW 0x0	Device Specific Initialization **NOTE:** Read only from PCI.
24:22	AuxCur	RW 0x0	Auxiliary Current Requirements **NOTE:** Read only from PCI.
25	D1Sup	RW 0x1	D1 Power Management State Support Read only from PCI. 0 = Not supported 1 = Supported
26	D2Sup	RW 0x1	D2 Power Management State Support Read only from PCI. 0 = Not supported 1 = Supported
31:27	PMESup	RW 0xF	PME# Signal Support Indicates in which power states the MV64360/1/2 supports the PME# pin. Each bit corresponds to different state (bit[0] - D0, bit[1] - D1, bit[2] - D2, bit[3] - D3-hot, bit[4] - D3-cold). For example, 'b01001 stands for supporting PME# only on D0 and D3-hot states. **NOTE:** Read only from PCI.

Table 433: PCI Power Management Control and Status
Offset: PCI_0 from CPU or PCI_0 0x44, PCI_0 from PCI_1 0xC4,
PCI_1 FROM CPU OR PCI_0 0xC4, PCI_1 from PCI_1 0x44

Bits	Field	Type/ Init Val	Function
1:0	PState	RW 0x0	Power State 00 = D0 01 = D1 10 = D2 11 = D3-hot

Table 433: PCI Power Management Control and Status (Continued)
Offset: PCI_0 from CPU or PCI_0 0x44, PCI_0 from PCI_1 0xC4,
PCI_1 FROM CPU OR PCI_0 0xC4, PCI_1 from PCI_1 0x44

Bits	Field	Type/ Init Val	Function
7:2	Reserved	RO 0x0	Read only
8	PME_EN	RW 0x0	PME# Pin Assertion Enable
12:9	DSel	RW 0x0	Data Select
14:13	DScale	RW 0x0	Data Scale **NOTE:** Read only from PCI.
15	PME_Stat	RW 0x0	PME# Pin Status CPU set only by writing ‘1’. PCI clear only by writing ‘1’. When set to ‘1’, the MV64360/1/2 asserts PME# pin.
23:16	P2P	RW 0x0	Power Management Status and Control for P2P Bridge **NOTE:** Read only from PCI.
31:24	Data	RW 0x0	State Data **NOTE:** Read only from PCI.

Table 434: PCI VPD Address
Offset: PCI_0 from CPU or PCI_0 0x48, PCI_0 from PCI_1 0xC8,
PCI_1 FROM CPU OR PCI_0 0xC8, PCI_1 from PCI_1 0x48

Bits	Field	Type/ Init Val	Function
7:0	CapID	RW 0x3	Capability ID **NOTE:** Read only from PCI.
15:8	NextPtr	RW 0x50	Next Item Pointer **NOTE:** Read only from PCI.
30:16	Addr	RW 0x0	VPD Address Points to the location of the VPD structure in memory. **NOTE:** The MV64360/1/2 also implements remapping of the high address bits through the PCI Address Decoding Control register.
31	Flag	RW 0x0	Flag Flipped by System or MV64360/1/2 during VPD Access On VPD writes, system sets the flag to ‘1’ indicating VPD write is required. The MV64360/1/2 clears the flag to indicate that the VPD write is done (data from the VPD Data register was written to memory). On VPD reads, the system sets the flag to ‘0’, indicating VPD read is required. The MV64360/1/2 sets the flag to ‘1’ when the read is done (data has been read from memory and put in VPD Data register).

Table 435: **PCI VPD Data**
Offset: PCI_0 from CPU or PCI_0 0x4C, PCI_0 FROM PCI_1 0xCC, PCI_1 FROM CPU OR PCI_0 0xCC, PCI_1 FROM PCI_1 0x4C

Bits	Field	Type/ Init Val	Function
31:0	Data	RW 0x0	VPD Data

Table 436: **PCI MSI Message Control**
Offset: PCI_0 from CPU or PCI_0 0x50, PCI_0 from PCI_1 0xD0, PCI_1 FROM CPU OR PCI_0 0xD0, PCI_1 from PCI_1 0x50

Bits	Field	Type/ Init Val	Function
7:0	CapID	RW 0x5	Capability ID **NOTE:** Read only from PCI.
15:8	NextPtr	RW 0x60	Next Item Pointer **NOTE:** Read only from PCI.
16	MSIEn	RW 0x0	MSI Enable 0 = Disable The MV64360/1/2 generates a PCI interrupt. 1 = Enabled The MV64360/1/2 generates MSI messages instead of interrupts.
19:17	MultiCap	RW 0x0	Multiple Messages Capable The MV64360/1/2 is capable of driving a single message. **NOTE:** Read only from PCI.
22:20	MultiEn	RW 0x0	Multiple Messages Enable The number of messages the system allocates to the MV64360/1/2 (must be smaller or equal to MultiCap).
23	Addr64	RW 0x1	64-bit Addressing Capable Indicates whether the MV64360/1/2 is capable of generating 64-bit message address. Read only from PCI. 0 = Not capable 1 = Capable
31:24	Reserved	RO 0x0	Read only ’0’.

Table 437: PCI MSI Message Address
**Offset: PCI_0 from CPU or PCI_0 0x54, PCI_0 from PCI_1 0xD4,
PCI_1 FROM CPU OR PCI_0 0xD4, PCI_1 from PCI_1 0x54**

Bits	Field	Type/ Init Val	Function
31:0	Addr	RW 0x0	Message Address

Table 438: PCI MSI Message Upper Address
**Offset: PCI_0 from CPU or PCI_0 0x58, PCI_0 from PCI_1 0xD8,
PCI_1 FROM CPU OR PCI_0 0xD8, PCI_1 from PCI_1 0x58**

Bits	Field	Type/ Init Val	Function
31:0	Addr	RW 0x0	Message Upper Address 32 upper address bits. If set to a value other than ‘0’, the MV64360/1/2 issues MSI message as DAC cycle.

Table 439: PCI Message Data
**Offset: PCI_0 from CPU or PCI_0 0x5C, PCI_0 FROM PCI_1 0xDC,
PCI_1 FROM CPU OR PCI_0 0xDC, PCI_1 FROM PCI_1 0x5C**

Bits	Field	Type/ Init Val	Function
15:0	Data	RW 0x0	Message Data Initiated by the system. Bits[1:0] of message data might be changed by the MV64360/1/2 when driving the message on the bus, depending on multiple message enable setting.
31:16	Reserved	RES 0x0	Read only ’0’.

Table 440: PCI-X Command
**Offset: PCI_0 from CPU or PCI_0 0x60, PCI_0 from PCI_1 0xE0,
PCI_1 FROM CPU OR PCI_0 0xE0, PCI_1 from PCI_1 0x60**

Bits	Field	Type/ Init Val	Function
7:0	CapID	RW 0x7	Capability ID **NOTE:** Read only from PCI.
15:8	NextPtr	PCI0:0x68 PCI1:0x0	Next Item Pointer Read only from PCI.

Table 440: PCI-X Command (Continued)
Offset: PCI_0 from CPU or PCI_0 0x60, PCI_0 from PCI_1 0xE0, PCI_1 FROM CPU OR PCI_0 0xE0, PCI_1 from PCI_1 0x60

Bits	Field	Type/ Init Val	Function
16	DPERE	RW 0x0	Data Parity Error Recovery Enable. 0 = Disable 1 = Enable If software can not recover from data parity errors, SERR# Mask register should be set properly, to enable SERR# assertion whenever the Master Data Parity Error bit is set.
17	ERO	RW 0x0	Enable Relaxed Ordering. The master never sets RO attribute bit, regardless of this bit setting.
19:18	MMRBC	RW 0x0	Maximum Memory Read Byte Count. Sets the upper limit of a Sequence Byte Count initiated by the MV64360/1/2 PCI master: 0x0 = 512 bytes 0x1 = 1024 bytes 0x2 = 2048 bytes 0x3 = 4096 bytes MV64360/1/2 PCI master maximum byte count is 128 regardless of the setting of `RdBC`
22:20	MOST	RW 0x3	Maximum Outstanding Split Transactions. Sets the maximum number of split reads the MV64360/1/2 PCI master is allowed to have at a time: 0x0 = Maximum one outstanding transaction 0x1 = Maximum 2 outstanding transactions 0x2 = Maximum 3 outstanding transactions 0x3 = Maximum 4 outstanding transactions 0x4 = Maximum 8 outstanding transactions 0x5 = Maximum 12 outstanding transactions 0x6 = Maximum 16 outstanding transactions 0x7 = Maximum 32 outstanding transactions **NOTE:** MV64360/1/2 PCI master has a maximum of four outstanding split reads at a time, even if `MaxSplit` is set to a higher number.
31:23	Reserved	RES 0x0	

Table 441: PCI-X Status
Offset: PCI_0 from CPU or PCI_0 0x64, PCI_0 from PCI_1 0xE4, PCI_1 FROM CPU OR PCI_0 0xE4, PCI_1 from PCI_1 0x64

Bits	Field	Type/ Init Val	Function
2:0	FN	RW 0x0	Function Number. **NOTE:** Read only from PCI.

Table 441: PCI-X Status (Continued)
**Offset: PCI_0 from CPU or PCI_0 0x64, PCI_0 from PCI_1 0xE4,
PCI_1 FROM CPU OR PCI_0 0xE4, PCI_1 from PCI_1 0x64**

Bits	Field	Type/ Init Val	Function
7:3	DN	RW 0x1F	Device Number. The MV64360/1/2 updates this field with the contents of AD[15:11] of a configuration write transaction's address phase to any of the MV64360/1/2 configuration registers (configuration Type/Init Val 0). **NOTE:** Read only from PCI.
15:8	BN	RW 0xFF	Bus Number. The MV64360/1/2 updates this field with the contents of AD[7:0] of a configuration write transaction's address phase to any of the MV64360/1/2 configuration registers (configuration Type/Init Val 0). **NOTE:** Read only from PCI.
16	Width	RW 0x1	PCI Interface Width 0 = 32-bit 1 = 64-bit **NOTE:** Read only from PCI.
17	133Cap	RO 0x1	133MHz Capable. 0 = Maximum 66MHz 1 = Maximum 133MHz
18	SCD	RW 0x0	Split completion discard buffer. Set when the PCI slave split completion transaction from a non prefetchable BAR space is terminated with target abort or master abort, and it's read buffer is discarded.
19	USC	RW 0x0	Unexpected Split Completion. Set when the byte count of a split completion transaction targeted to the PCI master does not match the requested byte count.
20	DC	RW 0x0	Device Complexity 0 = Simple Device 1 = Bridge Device **NOTE:** Read only from PCI.
22:21	DMMRBC	RW 0x0	Design Maximum Memory Read Byte Count. Specifies the maximum byte count the PCI master requests on memory read burst transactions. 0x0 = Maximum 512 bytes 0x1 = Maximum 1024 bytes 0x2 = Maximum 2048 bytes 0x3 = Maximum 4096 bytes **NOTE:** Read Only from PCI. MV64360/1/2 PCI master request maximum of 128 bytes per transaction.

Table 441: PCI-X Status (Continued)
Offset: PCI_0 from CPU or PCI_0 0x64, PCI_0 from PCI_1 0xE4,
PCI_1 FROM CPU OR PCI_0 0xE4, PCI_1 from PCI_1 0x64

Bits	Field	Type/ Init Val	Function
25:23	DMOST	RW 0x3	Design Maximum Outstanding Split Transactions Specifies the maximum outstanding split transactions the PCI master is capable of handling. 0x0 = One outstanding transaction 0x1 = 2 outstanding transactions 0x3 = 3 outstanding transactions 0x3 = 4 outstanding transactions 0x4 = 8 outstanding transactions 0x5 = 12 outstanding transactions 0x6 = 16 outstanding transactions 0x7 = 32 outstanding transactions **NOTE:** Read Only from PCI.
28:26	DMCRS	RW 0x0	Design Maximum Cumulative Read Size Specifies the total byte count a PCI master can absorb for all of its outstanding reads. 0x0 = 1 KB 0x1 = 2 KB 0x2 = 4 KB 0x3 = 8 KB 0x4 = 16 KB 0x5 = 32 KB 0x6 = 64 KB 0x7 = 128 KB **NOTE:** Read Only from PCI.
29	RSCEM	RWC 0x0	Receive Split Completion Error Message. The MV64360/1/2 sets this bit, if it receives a split completion message with Split Completion error attribute bit set. The bit is clear only (write 1 to clear).
31:30	Reserved	RES 0x0	Reserved.

Table 442: CompactPCI HotSwap
Offset: PCI_0 from CPU or PCI_0 0x68, PCI_0 FROM PCI_1 0xE8
PCI_1 FROM CPU OR PCI_0 0xE8, PCI_1 from PCI_1 0x68

NOTE: CompactPCI Hot Swap is only supported on the PCI_0 interface

Bits	Field	Type/ Init Val	Function
7:0	CapID	RW 0x6	Capability ID **NOTE:** Read only from PCI.
15:8	NextPtr	RW 0x0	Next Item Pointer **NOTE:** Read only from PCI.

Table 442: CompactPCI HotSwap (Continued)
Offset: PCI_0 from CPU or PCI_0 0x68, PCI_0 FROM PCI_1 0xE8
PCI_1 FROM CPU OR PCI_0 0xE8, PCI_1 from PCI_1 0x68

NOTE: CompactPCI Hot Swap is only supported on the PCI_0 interface

Bits	Field	Type/ Init Val	Function
16	Reserved	RES 0x0	Read only ’0’.
17	EIM	RW 0x0	ENUM# Interrupt Mask 0 = Enable signal 1 = Mask signal
18	Reserved	RES 0x0	Read only ’0’.
19	LOO	RW 0x0	LED On/Off 0 = LED off 1 = LED on
21:20	Reserved	RES 0x0	Read only ’0’.
22	Ext	RWC 0x0	Extraction Indicates that the board is about to be extracted (set to ‘1’). **NOTE:** Write ’1’ to clear.
23	Ins	RWC 0x0	Insertion Indicates that the board has just been inserted (set to ‘1’). **NOTE:** Write ’1’ to clear.
31:24	Reserved	RES 0x0	Read only ’0’.

E.8 Function 1 Configuration Registers

Note

The PCI_1 register offsets and the integrated SRAM registers only apply to the MV64360 and MV64361.

Table 443: PCI CS[2]# Base Address (low)
Offset: PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90,
PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0100000C	Same as CS[0]# Base Address (low).

Table 444: PCI CS[2]# Base Address (high)
Offset: PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94,
PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

Table 445: PCI CS[3]# Base Address (low)
Offset: PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98,
PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18

Bits	Field	Type/ Init Val	Function
31:12	Various	RW 0x0180000C	Same as CS[0]# Base Address (low).

Table 446: PCI CS[3]# Base Address (high)
Offset: PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C,
PCI_1 FROM CPU OR PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

Table 447: PCI Integrated SRAM Base Address (low)
Offset: PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0,
PCI_1 FROM CPU OR PCI_0 0xA0, PCI_1 from PCI_1 0x20

NOTE: The integrated SRAM registers only apply to the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x4200000c	Same as CS[0]# Base Address (low).

Table 448: PCI Integrated SRAM Base Address (high)
Offset: PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4,
PCI_1 FROM CPU OR PCI_0 0xA4, PCI_1 from PCI_1 0x24

NOTE: The integrated SRAM registers only apply to the MV64360 and MV64361.

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

E.9 Function 2 Configuration Registers

Note

The PCI_1 register offsets only apply to the MV64360 and MV64361.

Table 449: PCI DevCS[0] Base Address (low)
Offset: PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90,
PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10

Bits	Field	Type/ Init Val	Function
0	MemSpace	RW 0x0	Memory Space Indicator **NOTE:** Read only from PCI.
2:1	Type/ Init Val	RW 0x2	BAR Type/ Init Val. Locate anywhere in 64-bit address space. **NOTE:** Read only from PCI.
3	Prefetch	RW 0x0	Prefetch Enable **NOTE:** Read only from PCI.
11:4	Reserved	RES 0x0	Read only.
31:12	Various	RW 0x1c000	Same as CS[0]# Base Address (low).

Table 450: PCI DevCS[0]# Base Address (high)
Offset: PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94,
PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

Table 451: PCI DevCS[1]# Base Address (low)
Offset: PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98,
PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x1c800004	Same as DevCS[0]# Base Address (low).

Table 452: PCI DevCS[1]# Base Address (high)
Offset: PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C, PCI_1 FROM CPU OR PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

Table 453: PCI DevCS[2]# Base Address (low)
Offset: PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0, PCI_1 FROM CPU OR PCI_0 0xA0, PCI_1 from PCI_1 0x20

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x1d000004	Same as DevCS[0]# Base Address (low).

Table 454: PCI DevCS[2]# Base Address (high)
Offset: PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4, PCI_1 FROM CPU OR PCI_0 0xA4, PCI_1 from PCI_1 0x24

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

E.10 Function 3 Configuration Registers

Note

The PCI_1 register offsets only apply to the MV64360 and MV64361.

Table 455: PCI DevCS[3]# Base Address (Low)
Offset: PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90, PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10

Bits	Field	Type/ Init Val	Function
31:0	Various	0xFF000004	Same as DevCS[0]# Base Address (low).

Table 456: PCI DevCS[3]# Base Address (High)
Offset: PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94,
PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high)

Table 457: PCI BootCS# Base Address (Low)
Offset: PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98,
PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18

Bits	Field	Type/ Init Val	Function
31:0	Various	0xFF800004	Same as DevCS[0]# Base Address (low).

Table 458: PCI BootCS# Base Address (High)
Offset: PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C,
PCI_1 FROM CPU OR PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

Table 459: PCI CPU Base Address (Low)
Offset: PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0,
PCI_1 FROM CPU OR PCI_0 0xA0, PCI_1 from PCI_1 0x20

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x4000000C	Same as CS[0]# Base Address (low).

Table 460: PCI CPU Base Address (High)
Offset: PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4,
PCI_1 FROM CPU OR PCI_0 0xA4, PCI_1 from PCI_1 0x24

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

E.11 Function 4 Configuration Registers

Note

The PCI_1 register offsets only apply to the MV64360 and MV64361.

Table 461: PCI P2P Mem0 Base Address (Low)
Offset: PCI_0 from CPU or PCI_0 0x10, PCI_0 from PCI_1 0x90, PCI_1 from CPU or PCI_0 0x90, PCI_1 from PCI_1 0x10

Bits	Field	Type/ Init Val	Function
31:0	Various	RW PCI_0: 0x2200000C PCI_1: 0x1200000C	Same as CS[0]# Base Address (low).

Table 462: PCI P2P Mem0 Base Address (High)
Offset: PCI_0 from CPU or PCI_0 0x14, PCI_0 from PCI_1 0x94, PCI_1 from CPU or PCI_0 0x94, PCI_1 from PCI_1 0x14

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

Table 463: PCI P2P Mem1 Base Address (Low)
Offset: PCI_0 from CPU or PCI_0 0x18, PCI_0 from PCI_1 0x98, PCI_1 from CPU or PCI_0 0x98, PCI_1 from PCI_1 0x18

Bits	Field	Type/ Init Val	Function
31:0	Various	RW PCI_0: 0x2400000C PCI_1: 0xF200000C	Same as CS[0]# Base Address (low).

Table 464: PCI P2P Mem1 Base Address (High)
Offset: PCI_0 from CPU or PCI_0 0x1C, PCI_0 FROM PCI_1 0x9C, PCI_1 FROM CPU OR PCI_0 0x9C, PCI_1 FROM PCI_1 0x1C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CS[0]# Base Address (high).

Table 465: PCI P2P I/O Base Address
Offset: PCI_0 from CPU or PCI_0 0x20, PCI_0 from PCI_1 0xA0, PCI_1 FROM CPU OR PCI_0 0xA0, PCI_1 from PCI_1 0x20

Bits	Field	Type/ Init Val	Function
0	IOSpace	RW 0x1	I/O Space Indicator **NOTE:** Read only from PCI.
11:1	Reserved	RO 0x0	Reserved.
31:12	Various	RW PCI_0: 0x20000 PCI_1: 0x10000	Same as CS[0]# Base Address (low).

Table 466: PCI Internal Regs I/O Mapped Base Address
Offset: PCI_0 from CPU or PCI_0 0x24, PCI_0 from PCI_1 0xA4, PCI_1 FROM CPU OR PCI_0 0xA4, PCI_1 from PCI_1 0x24

Bits	Field	Type/ Init Val	Function
0	IOSpace	RW 0x1	I/O Space Indicator **NOTE:** Read only from PCI.
15:1	Reserved	RO 0x0	Reserved.
31:16	Various	RW 0x1400	Same as CS[0]# Base Address.

Appendix F. Messaging Unit Interface Registers

F.1 Messaging Unit Register Map

Notes

- The offsets listed below relate to a CPU or PCI access to the Messaging Unit registers through the MV64360/1/2 internal registers space. If accessed from PCI_0 through the CS[0] BAR space, remove the 0x1c prefix from the offset. If accessed from PCI_1 through it's CS[0] BAR space, remove the 0x1c prefix from the offset, and toggle address bit [7]. This way, agent on PCI_0 accesses PCI_0 Outbound Interrupt Cause at offset 0x30, while agent on PCI_1 accesses PCI_1 Outbound Interrupt Cause also at offset 0x30.
- I_2O Inbound and Outbound ports at offsets 0x40 and 0x44 respectively, are only accessible via CS[0] BAR space. If using I_2O circular queues (the Queue Control register's `CirQEn` bit [0] set to '1'), any access to offset 0x40 or 0x44 in CS[0] BAR space results in access to the respective queue location in DRAM.

Table 467: **Messaging Unit Register Map**

Register	Offsets	Page
Inbound Message0	PCI_0 0x1C10, PCI_1 0x1C90	Table 468, p.577
Inbound Message1	PCI_0 0x1C14, PCI_1 0x1C94	Table 469, p.578
Outbound Message0	PCI_0 0x1C18, PCI_1 0x1C98	Table 470, p.578
Outbound Message1	PCI_0 0x1C1, PCI_1 0x1C9C	Table 471, p.578
Inbound Doorbell	PCI_0 0x1C20, PCI_1 0x1CA0	Table 472, p.578
Inbound Interrupt Cause	PCI_0 0x1C24, PCI_1 0x1CA4	Table 473, p.578
Inbound Interrupt Mask	PCI_0 0x1C28, PCI_1 0x1CA8	Table 474, p.579
Outbound Doorbell	PCI_0 0x1C2C, PCI_1 0x1CAC	Table 475, p.580
Outbound Interrupt Cause	PCI_0 0x1C30, PCI_1 0x1CB0	Table 476, p.580
Outbound Interrupt Mask	PCI_0 0x1C34, PCI_1 0x1CB4	Table 477, p.581

Table 467: Messaging Unit Register Map (Continued)

Register	Offsets	Page
Inbound Queue Port Virtual	PCI_0 0x1C40, PCI_1 0x1CC0	Table 478, p.581
Outbound Queue Port Virtual	PCI_0 0x1C44, PCI_1 0x1CC4	Table 479, p.582
Queue Control	PCI_0 0x1C50, PCI_1 0x1CD0	Table 480, p.582
Queue Base Address	PCI_0 0x1C54, PCI_1 0x1CD4	Table 481, p.582
Inbound Free Head Pointer	PCI_0 0x1C60, PCI_1 0x1CE0	Table 482, p.583
Inbound Free Tail Pointer	PCI_0 0x1C64, PCI_1 0x1CE4	Table 483, p.583
Inbound Post Head Pointer	PCI_0 0x1C68, PCI_1 0x1CE8	Table 484, p.583
Inbound Post Tail Pointer	PCI_0 0x1C6C, PCI_1 0x1CEC	Table 485, p.584
Outbound Free Head Pointer	PCI_0 0x1C70, PCI_1 0x1CF0	Table 486, p.584
Outbound Free Tail Pointer	PCI_0 0x1C74, PCI_1 0x1CF4	Table 487, p.585
Outbound Post Head Pointer	PCI_0 0x1C78, PCI_1 0x1CF8	Table 488, p.585
Outbound Post Tail Pointer	PCI_0 0x1C7C, PCI_1 0x1CFC	Table 489, p.585

F.2 Messaging Unit Registers

Table 468: Inbound Message0
Offset: PCI_0 0x1C10, PCI_1 0x1C90

Bits	Field	Type/ Init Val	Function
31:0	InMsg0	RW 0x0	Inbound Message Register **NOTE:** Read only from the CPU, or other PCI interface. When written, sets a bit in the Inbound Interrupt Cause Register and an interrupt is generated to the CPU, or other PCI interface.

Table 469: Inbound Message1
Offset: PCI_0 0x1C14, PCI_1 0x1C94

Bits	Field	Type/ Init Val	Function
31:0	InMsg1	RW 0x0	Same as Inbound Message0.

Table 470: Outbound Message0
Offset: PCI_0 0x1C18, PCI_1 0x1C98

Bits	Field	Type/ Init Val	Function
31:0	OutMsg0	RW 0x0	Outbound Message Register **NOTE:** Read only from the PCI. When written, sets bit in the Outbound Interrupt Cause Register and an interrupt is generated to the PCI.

Table 471: Outbound Message1
Offset: PCI_0 0x1C1, PCI_1 0x1C9C

Bits	Field	Type/ Init Val	Function
31:0	OutMsg1	RW 0x0	Same as Outbound Message0.

Table 472: Inbound Doorbell
Offset: PCI_0 0x1C20, PCI_1 0x1CA0

Bits	Field	Type/ Init Val	Function
31:0	InDoor	RW 0x0	Inbound Doorbell Register The PCI setting a bit in this register to ‘1’ causes a CPU (or other PCI interface) interrupt. Writing ‘1’ to the bit by the CPU (or other PCI interface) clears the bit, and de-asserts the interrupt).

Table 473: Inbound Interrupt Cause
Offset: PCI_0 0x1C24, PCI_1 0x1CA4

Bits	Field	Type/ Init Val	Function
0	InMsg0	RW 0x0	Inbound Message0 Interrupt Set when the Inbound Message0 register is written. The CPU writes a ‘1’ to clear it.
1	InDoorL	RO 0x0	Inbound Doorbell Interrupt bits [15:0] Set when at least one bit [15:0] of the Inbound Doorbell register is set. Read Only.

Table 473: Inbound Interrupt Cause (Continued)
Offset: PCI_0 0x1C24, PCI_1 0x1CA4

Bits	Field	Type/ Init Val	Function
3:2	Reserved	RES 0x0	Reserved.
4	InPQ	RW 0x0	Inbound Post Queue Interrupt Set when Inbound Post Queue gets written. The CPU writes it with a '1' to clear it.
5	OutFQOvr	RW 0x0	Outbound Free Queue Overflow Interrupt Set when Outbound Free Queue is full. The CPU writes it with a '1' to clear it.
15:6	Reserved	RES 0x0	Reserved.
16	InMsg1	RW 0x0	Inbound Message1 Interrupt Set when Inbound Message1 register is written. The CPU writes it with a '1' to clear it.
17	InDoorH	RO 0x0	Inbound Doorbell Interrupt bits [31:16] Set when at least one bit[31:16] of Inbound Doorbell register is set. Read Only.
31:18	Reserved	RES 0x0	Reserved.

Table 474: Inbound Interrupt Mask
Offset: PCI_0 0x1C28, PCI_1 0x1CA8

NOTE: The polarity of the Mask bits is opposite than the in the rest of the MV64360/1/2 Mask registers, in order to maintain compatibility with existing I2O drivers.

Bits	Field	Type/ Init Val	Function
0	InMsg0	RW 0x1	If set to '1', the Inbound Message0 interrupt is masked.
1	InDoorL	RW 0x1	If set to '1', the Inbound Doorbell [15:0] interrupt is masked.
3:2	Reserved	RES 0x3	Reserved.
4	InPQ	RW 0x1	If set to '1', the Inbound Post Queue interrupt is masked.
5	OutFQOvr	RW 0x1	If set to '1', the Outbound Free Queue Overflow interrupt is masked.
15:6	Reserved	0x3FF	Reserved.
16	InMsg1	RW 0x1	If set to '1', the Inbound Message1 interrupt is masked.

**Table 474: Inbound Interrupt Mask (Continued)
Offset: PCI_0 0x1C28, PCI_1 0x1CA8**

NOTE: The polarity of the Mask bits is opposite than the in the rest of the MV64360/1/2 Mask registers, in order to maintain compatibility with existing I2O drivers.

Bits	Field	Type/ Init Val	Function
17	InDoorH	RW 0x1	If set to '1', the Inbound Doorbell [31:16] interrupt is masked.
31:24	Reserved	RES 0x0	Reserved.

**Table 475: Outbound Doorbell
Offset: PCI_0 0x1C2C, PCI_1 0x1CAC**

Bits	Field	Type/ Init Val	Function
31:0	OutDoor	RW 0x0	Outbound Doorbell Register Setting a bit in this register to '1' by the CPU causes a PCI interrupt. Writing '1' to this bit by the PCI clears the bit, and de-assert the interrupt.

**Table 476: Outbound Interrupt Cause
Offset: PCI_0 0x1C30, PCI_1 0x1CB0**

Bits	Field	Type/ Init Val	Function
0	OutMsg0	RW 0x0	Outbound Message0 Interrupt Set when the Outbound Message0 register is written. The PCI writes it with '1' to clear it. **NOTE:** For the CPU, it is read only.
1	OutDoorL	RO 0x0	Outbound Doorbell Interrupt bits[15:0] Set when at least one bit[15:0] of Outbound Doorbell register is set. Read Only.
2	Reserved	RES 0x0	Reserved.
3	OutPQ	RO 0x0	Outbound Post Queue Interrupt Set as long as Outbound Post Queue is not empty. This bit is read only.
15:4	Reserved	RES 0x0	Reserved
16	OutMsg1	RW 0x0	Outbound Message1 Interrupt Set when the Outbound Message1 register is written. The PCI writes it with '1' to clear it. **NOTE:** For the CPU, it is read only.

Table 476: Outbound Interrupt Cause (Continued)
Offset: PCI_0 0x1C30, PCI_1 0x1CB0

Bits	Field	Type/ Init Val	Function
17	OutDoorH	RO 0x0	Outbound Doorbell Interrupt bits[31:16] Set when at least one bit[31:16] of Outbound Doorbell register is set. Read Only.
31:18	Reserved	RES 0x0	Reserved.

Table 477: Outbound Interrupt Mask
Offset: PCI_0 0x1C34, PCI_1 0x1CB4

NOTE: The polarity of the Mask bits is opposite than the in the rest of the MV64360/1/2 Mask registers, in order to maintain compatibility with existing I2O drivers.

Bits	Field	Type/ Init Val	Function
0	OutMsg0	RW 0x1	If set to ‘1’, Outbound Message0 interrupt is masked.
1	OutDoorL	RW 0x1	If set to ‘1’, Outbound Doorbell [15:0] interrupt is masked.
2	Reserved	RES 0x1	Reserved.
3	OutPQ	RW 0x1	If set to ‘1’, Outbound Post Queue interrupt is masked.
15:4	Reserved	RES 0xFFF	Reserved.
16	OutMsg1	RW 0x1	If set to ‘1’, Outbound Message 1 interrupt is masked.
17	OutDoorH	RW 0x1	If set to ‘1’, Outbound Doorbell 31:16] interrupt is masked.
31:18	Reserved	RES 0x0	Reserved.

Table 478: Inbound Queue Port Virtual [1]
Offset: PCI_0 0x1C40, PCI_1 0x1CC0

Bits	Field	Type/ Init Val	Function
31:0	InQPVReg	RW 0x0	Inbound Queue Port Virtual register A PCI write to this port results in a write to the Inbound Post Queue. A read from this port results in a read from the Inbound Free Queue. Reserved from the CPU side.

1. The Inbound Queue Port Virtual register is not accessible by the CPU. PCI access is only supported via the CS[0] BAR space. This virtual register's offset within CS[0] BAR space is 0x40.

**Table 479: Outbound Queue Port Virtual [1]
Offset: PCI_0 0x1C44, PCI_1 0x1CC4**

Bits	Field	Type/ Init Val	Function
31:0	OutQPVReg	RW 0x0	Outbound Queue Port Virtual register A PCI write to this port results in a write to the Outbound Free Queue. A read from this port results in a read from the Outbound Post Queue. Reserved from CPU side.

1. The Outbound Queue Port Virtual register is not accessible by the CPU. The PCI access is only supported via CS[0] the BAR space. The offset of this virtual register within CS[0] BAR space is 0x44.

**Table 480: Queue Control
Offset: PCI_0 0x1C50, PCI_1 0x1CD0**

Bits	Field	Type/ Init Val	Function
0	CirQEn	RW 0x0	Circular Queue Enable If '0', any PCI write to the queue is ignored. Upon a PCI read from the queue, 0xFFFFFFFF is returned. **NOTE:** Read Only from PCI side.
5:1	CirQSize	RW 0x1	Circular Queue Size 00001 = 16 KBs 00010 = 32 KBs 00100 = 64 KBs 01000 = 128 KBs 10000 = 256 KBs **NOTE:** Read Only from the PCI side.
7:6	CirQDev	RW 0x0	Circular Queue Location 00 = CS[0]# space 01 = CS[1]# space 10 = CS[2]# space 11 = CS[3]# space **NOTE:** Read Only from the PCI side.
31:8	Reserved	RES 0x0	Reserved

**Table 481: Queue Base Address
Offset: PCI_0 0x1C54, PCI_1 0x1CD4**

Bits	Field	Type/ Init Val	Function
19:0	Reserved	RES 0x0	Reserved.

Table 481: Queue Base Address
Offset: PCI_0 0x1C54, PCI_1 0x1CD4

Bits	Field	Type/ Init Val	Function
31:20	QBAR	RW 0x0	Queue Base Address register **NOTE:** Read Only from the PCI side.

Table 482: Inbound Free Head Pointer
Offset: PCI_0 0x1C60, PCI_1 0x1CE0

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Reserved.
19:2	InFHPtr	RW 0x0	Inbound Free Head Pointer **NOTE:** Read only from the PCI side. This register is maintained by the CPU software.
31:20	QBAR	RO 0x0	Queue Base Address register Read only.

Table 483: Inbound Free Tail Pointer
Offset: PCI_0 0x1C64, PCI_1 0x1CE4

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Reserved.
19:2	InFTPtr	RW 0x0	Inbound Free Tail Pointer **NOTE:** Read only from the PCI side. This register is incremented by the MV64360/1/2 after the PCI read from the Inbound port.
31:20	QBAR	RO 0x0	Queue Base Address register Read only.

Table 484: Inbound Post Head Pointer
Offset: PCI_0 0x1C68, PCI_1 0x1CE8

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Reserved.

Table 484: Inbound Post Head Pointer
Offset: PCI_0 0x1C68, PCI_1 0x1CE8

Bits	Field	Type/ Init Val	Function
19:2	InPHPtr	RW 0x0	Inbound Post Head Pointer **NOTE:** Read only from PCI side. This register is incremented by the MV64360/1/2 after the PCI write to the Inbound port.
31:20	QBAR	RO 0x0	Queue Base Address register

Table 485: Inbound Post Tail Pointer
Offset: PCI_0 0x1C6C, PCI_1 0x1CEC

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Reserved.
19:2	InPTPtr	RW 0x0	Inbound Post Tail Pointer **NOTE:** Read only from the PCI side. This register is maintained by the CPU software.
31:20	QBAR	RO 0x0	Queue Base Address register

Table 486: Outbound Free Head Pointer
Offset: PCI_0 0x1C70, PCI_1 0x1CF0

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Reserved.
19:2	OutFHPtr	RW 0x0	Outbound Free Head Pointer **NOTE:** Read only from the PCI side. This register is incremented by the MV64360/1/2 after the PCI write to the Outbound port.
31:20	QBAR	RO 0x0	Queue Base Address register Read only.

Table 487: Outbound Free Tail Pointer
Offset: PCI_0 0x1C74, PCI_1 0x1CF4

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Reserved.
19:2	OutFTPtr	RW 0x0	Outbound Free Tail Pointer **NOTE:** Read Only from PCI side. This register is maintained by the CPU software.
31:20	QBAR	RO 0x0	Queue Base Address register

Table 488: Outbound Post Head Pointer
Offset: PCI_0 0x1C78, PCI_1 0x1CF8

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Reserved.
19:2	OutPHPtr	RW 0x0	Outbound Post Head Pointer **NOTE:** Read Only from PCI side. This register is maintained by the CPU software.
31:20	QBAR	RO 0x0	Queue Base Address register

Table 489: Outbound Post Tail Pointer
Offset: PCI_0 0x1C7C, PCI_1 0x1CFC

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Reserved.
19:2	OutPTPtr	RW 0x0	Outbound Post Tail Pointer **NOTE:** Read only from the PCI side. This register is incremented by the MV64360/1/2 after the PCI read from the Outbound port.
31:20	QBAR	RO 0x0	Queue Base Address register

Appendix G. Gigabit Ethernet Controller Interface Registers

G.1 Gigabit Ethernet Controller Register Maps

Notes

- All interrupt cause registers are write ’0’ to clear, meaning that writing ‘1’ value has no effect, while writing ’0’ resets the relevant bit in the register.
- Register offset: Port 0 bits [11:10] = 00, Port 1 bits [11:10] = 10, Port 2 bits [11:10] = 1.
  There is an increment of 400 from port 0 to port 1 and from port 1 to port 2.
- Port 2 only applies to the MV64360 device.

Table 490: Serial Management Register Map
NOTE: Port 2 offsets only apply to the MV64360 device.

Description	Offset	Page Number
PHY Address Register	0x2000	Table 492, p.592
SMI	0x2004	Table 493, p.593
Ethernet Unit Default Address (EUDAR)	0x2008	Table 494, p.593
Ethernet Unit Default ID (EUDIDR)	0x200C	Table 495, p.594
Ethernet Unit Interrupt Cause (EUICR)	0x2080	Table 496, p.594
Ethernet Unit Interrupt Mask (EUIMR)	0x2084	Table 497, p.595
Ethernet Unit Error Address (EUEAR)	0x2094	Table 498, p.595
Ethernet Unit Internal Address Error (EUIAER)	0x2098	Table 499, p.596
Base Address Registers	BA0 0x2200, BA1 0x2208, BA2 0x2210, BA3 0x2218, BA4 0x2220, BA5 0x2228	Table 500, p.596
Size Registers (SR)	SR0 0x2204, SR1 0x220C, SR2 0x2214, SR3 0x221C, SR4 0x2224, SR5 0x222C	Table 501, p.598
Header Retarget Base (HRBR)	0x2230	Table 502, p.598

Table 490: Serial Management Register Map (Continued)
NOTE: Port 2 offsets only apply to the MV64360 device.

Description	Offset	Page Number
Headers Retarget Control (HRCR)	0x2234	Table 503, p.598
High Address Remap (HARR)	HARR0 0x2280, HARR1 0x2284, HARR2 0x2288, HARR3 0x228C	Table 504, p.599
Base Address Enable (BARER)	0x2290	Table 505, p.599
Ethernet Port Access Protect Registers (EPAPR)	Port0 0x2294, Port1 0x2298, Port2 0x229C	Table 506, p.600
MAC MIB Counters	Port0: 0x003000–0x00307C, Port1: 0x003080–0x0030FC, Port2: 0x003100–0x00317C	Description under G.5 “Port MIB Counter” on page 629.

Table 491: Gigabit Ethernet Port0/1/2 Register Map
NOTE: Port 2 offsets only apply to the MV64360 device.

Description	Offset	Page Number
Portx Configuration Registers (PxCR)	Port0 0x2400, Port1 0x2800, Port2 0x2C00	Table 507, p.600
Portx Configuration Extend Registers (PxCXR)	Port0 0x2404, Port1 0x2804, Port2 0x2C04	Table 508, p.602
MII Serial Parameters Registers	Port0 0x2408, Port1 0x2808, Port2 0x2C08	Table 509, p.602
GMII Serial Parameters	Port0 0x240C, Port1 0x280C, Port2 0x2C0C	Table 510, p.603
VLAN EtherType Registers (EVLANE)	Port0 0x2410, Port1 0x2810, Port2 0x2C10	Table 511, p.603
MAC Address Low Registers (MACAL)	Port0 0x2414, Port1 0x2814, Port2 0x2C14	Table 512, p.604
MAC Address High Registers (MACAH)	Port0 0x2418, Port1 0x2818, Port2 0x2C18	Table 513, p.604

Table 491: Gigabit Ethernet Port0/1/2 Register Map (Continued)
NOTE: Port 2 offsets only apply to the MV64360 device.

Description	Offset	Page Number
SDMA Configuration Registers (SDCR)	Port0 0x241C, Port1 0x281C, Port2 0x2C1C	Table 514, p.604
IP Differentiated Services CodePoint 0 to Priority Registers (DSCP0)	Port0 0x2420, Port1 0x2820, Port2 0x2C20	Table 515, p.606
IP Differentiated Services CodePoint 1 to Priority Registers (DSCP1)	Port0 0x2424, Port1 0x2824, Port2 0x2C24	Table 516, p.606
IP Differentiated Services CodePoint 2 to Priority Registers (DSCP2, DSCP3, DSCP4, DSCP5)	Port0 DSCP2 0x2428, DSCP3 0x242C, DSCP4 0x2430, DSCP5 0x2434 Port1 DSCP2 0x2828, DSCP3 0x282C, DSCP4 0x2830, DSCP5 0x2834 Port2 DSCP2 0x2C28, DSCP3 0x2C2C, DSCP4 0x2C30, DSCP5 0x2C34	Table 517, p.606
IP Differentiated Services CodePoint 6 to Priority Registers (DSCP6)	Port0 0x2438, Port1 0x2838, Port2 0x2C38	Table 518, p.606
Port Serial Control Registers (PSCR)	Port0 0x243C, Port1 0x283C, Port2 0x2C3C	Table 519, p.607
VLAN Priority Tag to Priority Registers (VPT2P)	Port0 0x2440, Port1 0x2840, Port2 0x2C40	Table 520, p.610
Ethernet Port Status Registers (PSR)	Port0 0x2444, Port1 0x2844, Port2 0x2C44	Table 521, p.611
Transmit Queue Command Registers (TQCR)	Port0 0x2448, Port1 0x2848, Port2 0x2C48	Table 522, p.613
Transmit Queue Fixed Priority Configuration Registers (TQFPC)	Port0 0x244C, Port1 0x284C, Port2 0x2C4C	Table 523, p.613
Port Transmit Token-Bucket Rate Configuration Registers (PTTBRC)	Port0 0x2450, Port1 0x2850, Port2 0x2C50	Table 524, p.614

Table 491: Gigabit Ethernet Port0/1/2 Register Map (Continued)
NOTE: Port 2 offsets only apply to the MV64360 device.

Description	Offset	Page Number
Reserved	0x2454, 2854, 2C54	-
Maximum Transmit Unit Registers (MTU)	Port0 0x2458, Port1 0x2858, Port2 0x2C58	Table 525, p.614
Port Maximum Token Bucket Size Registers (PMTBS)	Port0 0x245C, Port1 0x285C, Port2 0x2C5C	Table 526, p.614
Port Interrupt Cause Registers (ICR)	Port0 0x2460, Port1 0x2860, Port2 0x2C60	Table 527, p.615
Port Interrupt Cause Extend Registers (ICER)	Port0 0x2464, Port1 0x2864, Port2 0x2C64	Table 528, p.617
Port Interrupt Mask Registers (PIMR)	Port0 0x2468, Port1 0x2868, Port2 0x2C68	Table 529, p.619
Port Extend Interrupt Mask Registers (PEIMR)	Port0 0x246C, Port1 0x286C, Port2 0x2C6C	Table 530, p.619
Portx Rx FIFO Urgent Threshold Registers (PxRFUTR)	Port0 0x2470, Port1 0x2870, Port2 0x2C70	Table 531, p.620
Portx Tx FIFO Urgent Threshold Registers (PxTFUTR)	Port0 0x2474, Port1 0x2874, Port2 0x2C74	Table 532, p.620
Portx Rx Minimal Frame Size Registers (PxMFSR)	Port0 0x247C, Port1 0x287C, Port2 0x2C7C	Table 533, p.620
Portx Rx Discard Frame Counter Registers (PxDFC)	Port0 0x2484, Port1 0x2884, Port2 0x2C84	Table 534, p.621
Portx Overrun Frame Counter Registers (PxOFC)	Port0 0x2488, Port1 0x2888, Port2 0x2C88	Table 535, p.621
Port Internal Address Error Registers (EUIAER)	Port0 0x2494, Port1 0x2894, Port2 0x2C94	Table 536, p.621

Table 491: Gigabit Ethernet Port0/1/2 Register Map (Continued)
NOTE: Port 2 offsets only apply to the MV64360 device.

Description	Offset	Page Number
Ethernet Current Receive Descriptor Pointer Registers (CRDP)	Port0: Q0 0x260C, Q1 0x261C, Q2 0x262C, Q3 0x263C, Q4 0x264C, Q5 0x265C, Q6 0x266C, Q7 0x267C Port1: Q0 0x2A0C, Q1 0x2A1C, Q2 0x2A2C, Q3 0x2A3C, Q4 0x2A4C, Q5 0x2A5C, Q6 0x2A6C, Q7 0x2A7C Port2: Q0 0x2E0C, Q1 0x2E1C, Q2 0x2E2C, Q3 0x2E3C, Q4 0x2E4C, Q5 0x2E5C, Q6 0x2E6C, Q7 0x2E7C	Table 537, p.621
Receive Queue Command (RQCR)	Port0: 0x2680, Port1: 0x2A80, Port2: 0x2E80	Table 538, p.622
Transmit Current Served Descriptor Pointer Register	Port0 0x2684, Port1 0x2A84, Port2 0x2E84 (Read Only)	Table 539, p.623
Transmit Current Queue Descriptor Pointer (TCQDP)	Port0: Q0 0x26C0, Q1 0x26C4, Q2 0x26C8, Q3 0x26CC, Q4 0x26D0, Q5 0x26D4, Q6 0x26D8, Q7 0x26DC Port1: Q0 0x2AC0, Q1 0x2AC4, Q2 0x2AC8, Q3 0x2ACC, Q4 0x2AD0, Q5 0x2AD4, Q6 0x2AD8, Q7 0x2ADC Port2: Q0 0x2EC0, Q1 0x2EC4, Q2 0x2EC8, Q3 0x2ECC, Q4 0x2ED0, Q5 0x2ED4, Q6 0x2ED8, Q7 0x2EDC	Table 540, p.623

Table 491: Gigabit Ethernet Port0/1/2 Register Map (Continued)
NOTE: Port 2 offsets only apply to the MV64360 device.

Description	Offset	Page Number
Transmit Queue Token-Bucket Counter (TQxTBC)	Port0: Q0 0x2700, Q1 0x2710, Q2 0x2720, Q3 0x2730, Q4 0x2740, Q5 0x2750, Q6 0x2760, Q7 0x2770 Port1: Q0 0x2B00, Q1 0x2B10, Q2 0x2B20, Q3 0x2B30, Q4 0x2B40, Q5 0x2B50, Q6 0x2B60, Q7 0x2B70 Port2: Q0 0x2F00, Q1 0x2F10, Q2 0x2F20, Q3 0x2F30, Q4 0x2F40, Q5 0x2F50, Q6 0x2F60, Q7 0x2F70	Table 541, p.623
Transmit Queue Token Bucket Configuration (TQxTBC)	Port0: Q0 0x2704, Q1 0x2714, Q2 0x2724, Q3 0x2734, Q4 0x2744, Q5 0x2754, Q6 0x2764, Q7 0x2774 Port1: Q0 0x2B04, Q1 0x2B14, Q2 0x2B24, Q3 0x2B34, Q4 0x2B44, Q5 0x2B54, Q6 0x2B64, Q7 0x2B74 Port2: Q0 0x2F04, Q1 0x2F14, Q2 0x2F24, Q3 0x2F34, Q4 0x2F44, Q5 0x2F54, Q6 0x2F64, Q7 0x2F74	Table 542, p.624
Transmit Queue Arbiter Configuration (TQxAC)	Port0: Q0 0x2708, Q1 0x2718, Q2 0x2728, Q3 0x2738, Q4 0x2748, Q5 0x2758, Q6 0x2768, Q7 0x2778 Port1: Q0 0x2B08, Q1 0x2B18, Q2 0x2B28, Q3 0x2B38, Q4 0x2B48, Q5 0x2B58, Q6 0x2B68, Q7 0x2B78 Port2: Q0 0x2F08, Q1 0x2F18, Q2 0x2F28, Q3 0x2F38, Q4 0x2F48, Q5 0x2F58, Q6 0x2F68, Q7 0x2F78	Table 543, p.624

Table 491: Gigabit Ethernet Port0/1/2 Register Map (Continued)
NOTE: Port 2 offsets only apply to the MV64360 device.

Description	Offset	Page Number
Reserved	0x270C, 0x271C, 0x272C, 0x273C, 0x274C, 0x275C, 0x276C, 0x277C 0x2B0C, 0x2B1C, 0x2B2C, 0x2B3C, 0x2B4C, 0x2B5C, 0x2B6C, 0x2B7C 0x2F0C, 0x2F1C, 0x2F2C, 0x2F3C, 0x2F4C, 0x2F5C, 0x2F6C, 0x2F7C	-
Port Transmit Token-Bucket Counter (PTTBC)	Port0 0x2780, Port1 0x2B80, Port2 0x2F80	Table 544, p.625
Destination Address Filter Special Multicast Table (DFSMT)	Port0: 0x3400–0x34FC, Port1: 0x3800–0x38FC, Port2: 0x3C00–0x3CFC	Table 545, p.625
Destination Address Filter Other Multicast Table (DFOMT)	Port0: 0x3500–0x35FC, Port1: 0x3900–0x39FC, Port2: 0x3D00–0x3DFC	Table 546, p.626
Destination Address Filter Unicast Table (DFOMT)	Port0: 0x3600–0x360C, Port1: 0x3A00–0x3A0C, Port2: 0x3E00–0x3E0C	Table 547, p.628

G.2 Serial Management Interface Registers

Table 492: PHY Address Register
Offset: 0x2000

Bits	Field	Type/ Init Val	Function
4:0	PhyAd_0	0x8	PHY device address for port 0, R/W
9:5	PhyAd_1	0x9	PHY device address for port 1, R/W
14:10	PhyAD_2	0xA	PHY device address for port 2, R/W **NOTE:** Only applies to the MV64360. Reserved for the MV64361 and MV64362.
31:15	Reserved	RES 0x0	Read Only

Table 493: SMI
Offset: 0x2004

Bits	Field	Type/ Init Val	Function
15:0	Data	RW N/A	**Management for SMI READ operation:** Two transactions are required: (1) management write to the SMI register where Opcode = `1`, PhyAd, RegAd with the Data having any value. (2) management read from the SMI register. When reading back the SMI register, the Data is the addressed PHY register contents if the `ReadValid` bit [27] is '1'. The Data remains undefined as long as `ReadValid` is '0'. **Management for SMI WRITE operation:** One Management transaction is required: Management write to the SMI register with Opcode = `0`, PhyAd, RegAd with the Data to be written to the addressed PHY register.
20:16	PhyAd	RW 0x0	PHY device address
25:21	RegAd	RW 0x0	PHY device register address
26	Opcode	RW 0x1	0 = Write 1 = Read
27	ReadValid	RO 0x0	1 = Indicates that the Read operation for the addressed RegAd register has completed, and that the data is valid on the Data field.
28	Busy	RO 0x0	1 = Indicates that an operation is in progress and that the CPU should not write to the SMI register during this time.
31:29	N/A	RW 0x0	These bits should be driven 0x0 during any write to the SMI register.

G.3 Address Decoding Registers

Table 494: Ethernet Unit Default Address (EUDAR)
Offset: 0x2008

Bits	Field	Type/ Init Val	Function
31:0	DAR	RW 0x0	Specifies the Default Address to which the Ethernet unit directs no match, multiple address hits, and address protect violations. Occurrence of this event may be the result of programming errors of the descriptor pointers or buffer pointers.

Table 495: Ethernet Unit Default ID (EUDIDR)
Offset: 0x200C

Bits	Field	Type/ Init Val	Function
3:0	DIDR	RW 0x0 (DRAM)	Specifies the Default ID of the target unit to which the Ethernet unit directs no match and address protect violations. Identical to Base Address register's `Target` field encoding.
11:4	DATTR	RW 0xE	Specifies the Default Attribute of the target unit to which the Ethernet unit directs no match and address protect violations. Identical to Base Address register's `Attr` field encoding.
31:12	Reserved	RO 0x0	Read Only

Table 496: Ethernet Unit Interrupt Cause (EUICR)
Offset: 0x2080

NOTE: Write '0' to clear interrupt bits. Writing 1 does not effect the interrupt bits.

Bits	Field	Type/ Init Val	Function
0	EtherIntSum	RW 0x0	Ethernet Unit Interrupt Summary This bit is a logical OR of the (unmasked) bits [6:1] in the register.
1	Parity	RW 0x0	Parity Error Effect on Tx DMA operation: If a parity error occurs on the first descriptor fetch, the DMA stops and disables the queue. If it is on a non-first descriptor, the tx_dma, in addition, asserts the tx_error interrupt. If it is on a packet's data, the tx_dma continues with the transmission but does not ask to generate CRC at the end of the packet (even if `GC` is set in the first descriptor command). Effect on Rx DMA operation: If a parity error occurs on the first descriptor fetch, the DMA stops and disables the queue. If it is on a non-first descriptor, the rx_dma, in addition, asserts the rx_error interrupt.
2	AddressViola-tion	RW 0x0	During read operations, the tx_dma continues with the transmission but does not ask to generate CRC at the end of the packet (even if `GC` is set in the first descriptor command). During write operations, the DMA will not stop and will remain unaware of the violation, the transactions will be directed to the default address, as specified in the default address register.
3	Address-NoMatch	RW 0x0	During read operations, the tx_dma continues with the transmission but does not ask to generate CRC at the end of the packet (even if `GC` is set in the first descriptor command). During write operations, the DMA will not stop and will remain unaware of the violation, the transactions will be directed to the default address, as specified in the default address register.
4	SMIdone	RW 0x0	SMI Command Done Indicates the SMI completed a MII management command (either read or write) that was initiated by the CPU writing to the SMI register.

Table 496: Ethernet Unit Interrupt Cause (EUICR) (Continued)
Offset: 0x2080

NOTE: Write '0' to clear interrupt bits. Writing 1 does not effect the interrupt bits.

Bits	Field	Type/ Init Val	Function
5	Count_wa	RW 0x0	Counters Wrap Around Indication MIB Counter WrapAround Interrupt is set if one of the MIB counters wrapped around (passed 32 bits).
6	Reserved	RES 0x0	Reserved
7	Internal_addr_error	RW 0x0	Internal Address Error is set when there is an access to an illegal offset of the internal registers. When set, the Internal Address Error register locks the address that caused the error.
31:8	Reserved	RES 0x0	Reserved

Table 497: Ethernet Unit Interrupt Mask (EUIMR)
Offset: 0x2084

Bits	Field	Type/ Init Val	Function
7:0	Various	RW 0x0	Mask bits for Unit Interrupt Cause register. 0 = Mask 1 = Do not mask
31:8	Reserved	RES 0x0	Reserved

Table 498: Ethernet Unit Error Address (EUEAR)
Offset: 0x2094

Bits	Field	Type/ Init Val	Function
31:0	Error Address	RO	Locks the address, if there is an address violation of the DMA such as: Multiple Address window hit, No Hit, Access Violations The Address is locked until the register is read. (Used for software debug after address violation interrupt is raised) This field is read only.

Table 499: Ethernet Unit Internal Address Error (EUIAER)
Offset: 0x2098

Bits	Field	Type/ Init Val	Function
8:0	Internal Address	RO	Locks the relevant internal address bits (bit 12 and bits 9:2), if there is an address violation of unmapped access to the GbE unit top registers. The Address is locked until the register is read. (This field is used for software debugging after an address violation interrupt is raised.)
31:9	Reserved	RES 0x0	Reserved

Table 500: Base Address Registers
Offset: BA0 0x2200, BA1 0x2208, BA2 0x2210, BA3 0x2218, BA4 0x2220, BA5 0x2228

Bits	Field	Type/ Init Val	Function
3:0	Target	RW 0x0	Specifies the target resource associated with this window. 0x0 = SDRAM 0x1 = Device 0x2 = CPU/Integrated memory. 0x3 = PCI0 bus 0x4 = PCI1 bus 0x5 = Cunit 0x6 = Aunit 0x7 = Gunit 0x8 – 0xF = Reserved **NOTE:** 0x4 only applies to the MV64360 and MV64361. Reserved for the and MV64362.
7:4	Reserved	RES 0x0	Reserved.

Table 500: Base Address Registers (Continued)
Offset: BA0 0x2200, BA1 0x2208, BA2 0x2210, BA3 0x2218, BA4 0x2220, BA5 0x2228

Bits	Field	Type/ Init Val	Function
15:8	Attr	RW 0x0	Specifies target specific attributes **If the target interface is DRAM:** Bits[11:8] = Select DRAM bank: 0xE = CS[0]# 0xD = CS[1]# 0xB = CS[2]# 0x7 = CS[3]# All other values for bits [11:8] are Reserved. Bits[13:12] = Cache coherency support 0x0 = No cache coherency 0x1 = WT cache coherency 0x2 = WB cache coherency 0x3 = Reserved Bits[15:14] = Reserved **If the target interface is Device bus:** Bits[12:8] = Select Device bank: 0x1E = DevCS[0]# 0x1D = DevCS[1]# 0x1B = DevCS[2]# 0x17 = DevCS[3]# 0x0F = BootCS# All other values for bits [12:8] are Reserved. Bits[15:13] = Reserved **If the target interface is PCI0 or PCI1 (MV64360 and MV64361):** Bits[9:8] = Data swap 0x0 = Byte swap 0x1 = No swap 0x2 = Both byte and word swap 0x3 = Word swap Bit[10] = PCI-X No Snoop (NS) attribute 0x0 = NS attribute is not asserted 0x1 = NS attribute is asserted Bit[11] = Select PCI I/O or memory space 0x0 = I/O 0x1 = Memory Bit[12] = PCI REQ64# control (Reserved in the MV64361.) 0x0 = Force PCI master to always assert REQ64# 0x1 = Not forced, REQ64# is asserted according to the requested data size Bit[15:13] = Reserved **If the target interface is internal SRAM (MV64360 and MV64361) or 60x bus:** Bit[8] = Specify the internal SRAM block that is selected 0x0 = Block 0 0x1 = Block 1 Bits[10:9] and [15:12] = Reserved Bit[11] = Integrated SRAM/60x bus: Must be 0x0. 0x0 = Integrated SRAM 0x1 = 60x bus

Table 500: Base Address Registers (Continued)
Offset: BA0 0x2200, BA1 0x2208, BA2 0x2210, BA3 0x2218, BA4 0x2220, BA5 0x2228

Bits	Field	Type/ Init Val	Function
31:16	Base	RW 0x0	Base address. Used together with the size register to set the address window size and location within the range of 4 Gbyte space. An address driven by one of the Ethernet SDMAs is considered as a window hit if (`address | size) == (base | size`).

Table 501: Size Registers (SR)
Offset: SR0 0x2204, SR1 0x220C, SR2 0x2214, SR3 0x221C, SR4 0x2224, SR5 0x222C

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RES 0x0	Reserved.
31:16	Size	RW 0x0	Window size. Used together with the size register to set the address window size and location within the range of 4 Gbyte space. Must be programmed from LSB to MSB as sequence of 1's followed by sequence of 0's. The number of 1's specifies the size of the window in 64 Kbyte granularity (for example, a value of 0x00FF specifies 256x64k = 16 Mbyte). An address driven by one of the Ethernet MACs is considered as a window hit if (`address | size) == (base | size`).

Table 502: Header Retarget Base (HRBR)
Offset: 0x2230

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RES 0x0	Reserved.
31:1 6	Base	RW 0x0	Base address. Retarget is executed if the address matches Base.

Table 503: Headers Retarget Control (HRCR)
Offset: 0x2234

Bits	Field	Type/ Init Val	Function
0	En	RW 0x0	Headers retarget enable bit 0x0 = Disable 0x1 = Enable

Table 503: Headers Retarget Control (HRCR) (Continued)
Offset: 0x2234

Bits	Field	Type/ Init Val	Function
3:1	BSize	RW 0x0	Buffer Size 0x0 = 256 bytes 0x1 = 512 bytes 0x2 = 1 Kbyte 0x3 = 2 Kbyte 0x4 = 4 Kbyte 0x5 = 8 Kbyte 0x6 – 0x7 = Reserved
15:4	Reserved	RES 0x0	Reserved.
31:16	Mask1	RW 0x0	Defines the total space of the buffers to be manipulated, in 64 Kbyte granularity. Size must be set from LSB to MSB as a sequence of 1's, followed by sequence of 0's. For example, in order to retarget the headers of 1K buffers of 1 Kbyte size, which means 1 Mbyte of buffers space, Mask1 should be set to 0x000F. NOTE: The total address space of retargeted headers must not exceed the integrated SRAM size (256 Kbyte). The minimum buffers space to be manipulated is 64 Kbyte.

Table 504: High Address Remap (HARR)[1]
Offset: HARR0 0x2280, HARR1 0x2284, HARR2 0x2288, HARR3 0x228C

Bits	Field	Type/ Init Val	Function
31:0	Remap	RW 0x0	Remap address. Specifies address bits [63:32] to be driven to the target interface. Relevant only for target interfaces that supports more than 4 Gbyte of address space (for example, PCI bus).

1. Remap 0 corresponds to Base Address register 0, Remap 1 to Base Address register 1, Remap 2 to Base Address register 2 and Remap 3 to Base Address register 3.

Table 505: Base Address Enable (BARER)
Offset: 0x2290

Bits	Field	Type/ Init Val	Function
5:0	En	RW 0x3F	Address window enable. This is one bit per window. If it is set to 0, the corresponding address window is enabled. (bit [0] matches to the Window0, bit [1] to Window1, etc.) 0 = Enable 1 = Disable
31:6	Reserved	RES 0x0	Reserved.

Table 506: Ethernet Port Access Protect Registers (EPAPR)
Offset: Port0 0x2294, Port1 0x2298, Port2 0x229C

Bits	Field	Type/ Init Val	Function
1:0	Win0	RW 0x3	Window0 access control: 0x0 = No access allowed 0x1 = Read Only 0x2 = Reserved 0x3 = Full access (read or write) In case of access violation (for example, write data to a read only region), an interrupt is set, and the transaction is written or read from the default address, as specified in the default address register.
3:2	Win1	RW 0x3	Window1 access control (the same as Win0 access control).
5:4	Win2	RW 0x3	Window2 access control (the same as Win0 access control).
7:6	Win3	RW 0x3	Window3 access control (the same as Win0 access control).
9:8	Win4	RW 0x3	Window4 access control (the same as Win0 access control).
11:10	Win5	RW 0x3	Window5 access control (the same as Win0 access control).
31:12	Reserved	RES 0x0	Reserved

G.4 Port0/1/2 Control Registers

Note

Port 2 offsets only apply to the MV64360 device.

Table 507: Portx Configuration Registers (PxCR)
Offset: Port0 0x2400, Port1 0x2800, Port2 0x2C00

Bits	Field	Type/ Init Val	Function
0	UPM	RW 0x0	Unicast Promiscuous mode 0 = Normal mode. Unicast frames are received only if the destination address is found in the DA-filter table and DA is matched against the port DA MAC Address base. 1 = Promiscuous mode. Unicast unmatched frames are received to the Rx queue.

Table 507: Portx Configuration Registers (PxCR) (Continued)
Offset: Port0 0x2400, Port1 0x2800, Port2 0x2C00

Bits	Field	Type/ Init Val	Function
3:1	RXQ	RW 0x0	Default Rx Queue Is the Default Rx Queue for not matched Unicast frames when UPM bit is set. It is also the default Rx Queue for all MAC broadcast (except for ARP broadcast that has a different field for default queue) if receiving them is enabled.
6:4	RXQArp	RW 0x0	Default Rx Queue for ARP Broadcasts, if receiving them is enabled.
7	RB	RW 0x0	Reject mode of MAC Broadcasts that are not IP or ARP Broadcast. 0 = Receive to the RXQ queue. 1 = Reject.
8	RBIP	RW 0x0	Reject mode of MAC Broadcast s that are IP (Ethertype 0x800). 0 = Receive to the RXQ queue. 1 = Reject.
9	RBArp	RW 0x0	Reject mode of MAC Broadcasts that are ARP (Ethertype 0x806). 0 = Receive to RXQArp queue. 1 = Reject.
10	Reserved	RES 0x0	Reserved
11	Reserved	RES 0x0	Reserved
12	AMNoTxES	RW 0x0	Automatic mode not updating Error Summary in Tx descriptor When set, then for each Transmitted packet with `AM=1` on first descriptor, no status will be reported in the last descriptor. The advantage of using this bit is that it avoids another write to memory to update the error status.
13	Reserved	RES 0x0	Reserved. Must be set to zero.
14	TCP_CapEn	RW 0x0	Capture TCP frames to <TCPQ>. 1 = Enable 0 = Disable
15	UDP_CapEn	RW 0x0	Capture UDP frames to <UDPQ>. 1 = Enable 0 = Disable
18:16	TCPQ	RW 0x0	Captured TCP frames are directed to this Queue number.
21:19	UDPQ	RW 0x0	Captured UDP frames are directed to this Queue number.
24:22	BPDUQ	RW 0x7	Captured BPDU frames (if PCXR<Span> is set) are directed to this Queue number.

Table 507: Portx Configuration Registers (PxCR) (Continued)
Offset: Port0 0x2400, Port1 0x2800, Port2 0x2C00

Bits	Field	Type/ Init Val	Function
31:25	Reserved	RES 0x0	Reserved

Table 508: Portx Configuration Extend Registers (PxCXR)
Offset: Port0 0x2404, Port1 0x2804, Port2 0x2C04

Bits	Field	Type/ Init Val	Function
0	Reserved	RES 0x0	Reserved. (Read Only)
1	Span	RW 0x0	Spanning Tree packets capture enable. 0 = BPDU packets are treated as normal Multicast packets. 1 = BPDU packets are trapped and sent to the Port Configuration register BPDU queue.
2	ParEn	RW 0x0	Partition enable. When more than 61 collisions occur while transmitting, the port enters Partition mode. It waits for the first good packet from the wire, and then goes back to Normal mode. Under Partition mode it continues transmitting, but it does not receive. 0 = Disable. 1 = Enable.
31:3	Reserved	RO 0x0	Read Only

Table 509: MII Serial Parameters Registers
Offset: Port0 0x2408, Port1 0x2808, Port2 0x2C08

Bits	Field	Type/ Init Val	Function
1:0	JAM LENGTH	RW 11B (4K-bit times)	These two bits determine the JAM Length (in Back Pressure) as follows: 00 = 12K bit times 01 = 24K bit times 10 = 32K bit times 11 = 48K bit times NOTE: These bits can only be changed when `PortEn` bits are set to ‘0’ in all Port Control Registers (Port is disabled).
6:2	JAM-IPG	RW 01000B (32-bit times)	These five bits determine the JAM IPG. The step is 4-bit times. The `JAM IPG` varies between 4- and 124-bit times. NOTE: These bits can only be changed when `PortEn` bits are set to ‘0’ in all Port Control Registers (Port is disabled). The `JAM IPG` bit cannot be programmed to ’0’ or ’1’.

Table 509: MII Serial Parameters Registers (Continued)
Offset: Port0 0x2408, Port1 0x2808, Port2 0x2C08

Bits	Field	Type/ Init Val	Function
11:7	IPG-JAM_TO_DATA	RW 10000B (64-bit times)	These five bits determine the IPG JAM to DATA. The step is 4-bit times. The value may vary between 4- and 128-bit times. NOTE: These bits can only be changed when `PortEn` bits are set to '0' in all Port Control Registers (Port is disabled).
16:12	IPG-DATA	RW 11000B (96-bit times)	Inter-Packet Gap (IPG): The step is 4-bit times. The value may vary between 12- and 124-bit times. NOTE: These bits can only be changed when `PortEn` bits are set to '0' in all Port Control Registers (Port is disabled).
21:17	DataBlind	RW 10000B (64-bit times)	Data Blinder: The number of nibbles from the beginning of the IFG, in which the MV64360/1/2 restarts the IFG counter when detecting a carrier activity. Following this value, MV64360/1/2 enters the Data Blinder zone and does not reset the IFG counter. This ensures fair access to the medium. The value must be written in hexadecimal format. The default is 10 hex (64-bit times; 2/3 of the default IPG). The step is 4-bit times. Valid range is 3 to 1F hex nibbles. NOTE: These bits can only be changed when `PortEn` bits are set to '0' in all Port Control Registers (Port is disabled).
31:22	Reserved	RES 0x0	Reserved

Table 510: GMII Serial Parameters
Offset: Port0 0x240C, Port1 0x280C, Port2 0x2C0C

Bits	Field	Type/ Init Val	Function
2:0	IPG-DATA	RW 0x6 (96-bit times)	Inter-Packet Gap (IPG): The step is 16-bit times. The value may vary between 48- to 112-bit times. GMII is full-duplex only. **NOTE:** These bits may be changed only when `PortEn` bits are set to 0 in Port Control Register (Port is disabled).
31:3	Reserved	RES 0x0	Reserved

Table 511: VLAN EtherType Registers (EVLANE)
Offset: Port0 0x2410, Port1 0x2810, Port2 0x2C10

Bits	Field	Type/ Init Val	Function
15:0	VL_EtherType	RW 0x8100	The Ethertype for packets carrying the VLAN tag, for 802.1p priority field processing, and for continued parsing of the received frames Layer3/4 headers.
31:16	Reserved	RES 0x0	Reserved

Table 512: MAC Address Low Registers (MACAL)
Offset: Port0 0x2414, Port1 0x2814, Port2 0x2C14

Bits	Field	Type/ Init Val	Function
15:0	MAC[15:0]	RW 0x0	The least significant bits of the MAC Address. Used for both flow-control Pause frames as a source address, as well as for address filtering (see 15.5.1 “Parsing the Frames” on page 231). NOTE: MAC[47] is the Multicast/Unicast bit.
31:16	Reserved	RES 0x0	Read Only

Table 513: MAC Address High Registers (MACAH)
Offset: Port0 0x2418, Port1 0x2818, Port2 0x2C18

Bits	Field	Type/ Init Val	Function
31:0	MAC[47:16]	RW 0x0	The most significant bits of the MAC Address. Used for both flow-control Pause frames as source address, as well as for address filtering (see 15.5.1 “Parsing the Frames” on page 231). NOTE: MAC[47] is the Multicast/Unicast bit.

Table 514: SDMA Configuration Registers (SDCR)
Offset: Port0 0x241C, Port1 0x281C, Port2 0x2C1C

Bits	Field	Type/ Init Val	Function
0	RIFB	RW 0x0	Receive Interrupt on Frame Boundaries When set, the SDMA Rx generates interrupts only on frame boundaries (i.e. after writing the frame status to the descriptor). See also the IPG_Int_Rx field description (bits [21:8]) for further masking.
3:1	RxBSZ	RW 0x100	Rx Burst Size Sets the maximum burst size for Rx SDMA transactions: 000 = Burst is limited to 1 64-bit words. 001 = Burst is limited to 2 64-bit words. 010 = Burst is limited to 4 64-bit words. 011 = Burst is limited to 8 64-bit words. 100 = Burst is limited to 16 64-bit words. NOTE: This field effects only data-transfers. Descriptor fetch is done always with 4LW burst size. Should not be changed (other values degrade performance). However, a larger value is optimal for DRAM performance.
4	BLMR	RW 0x1	Big/Little Endian Receive Mode The DMA supports Big or Little Endian configurations per channel. The BLMR bit only affects data transfer to memory. 1 = No swap. 0 = Byte swap.

**Table 514: SDMA Configuration Registers (SDCR) (Continued)
Offset: Port0 0x241C, Port1 0x281C, Port2 0x2C1C**

Bits	Field	Type/ Init Val	Function
5	BLMT	RW 0x1	Big/Little Endian Transmit Mode The DMA supports Big or Little Endian configurations per channel. The BLMT bit only affects data transfer from memory. 1 = No swap. 0 = Byte swap.
6	SwapMode	RW 0x0	Swap mode The DMA supports swapping, for descriptors only, for both receive and transmit ports, on every access to memory space. 0 = No swapping. 1 = Byte swap: In every 64-bit word of the descriptor, the byte order is swapped such that byte 0 is placed in byte 7, byte 7 is placed in byte 0, byte 1 is placed in byte 6, byte 6 is placed in byte 1, byte 2 is placed in byte 5 etc.
7	Reserved	RES 0x0	
21:8	IPG_Int_Rx	RW 0x0	Rx frame IPG between interrupts counter and enable: This field provides a way to force a delay from the last ICR[RxBufferQueue] interrupt from any of the queues, to the next RxBufferQueue interrupt from any of the queues. The ICR bits still reflect the new interrupt, but this masking is reflected by potentially not propagating to the Global_Interrupt register in the PCI or CPU units. This provides a way for interrupt coalescing on receive packet events. The time is calculated in multiples of 64 clock cycles. Valid values are 0 (No delay between packets to CPU, the counter is effectively disabled.), through 0x3FFF (1,048,544 clock cycles). NOTE: See the detailed description in 15.7.1 “Interrupt Coalescing” on page 234.
24:22	TxBSZ	RW 0x100 NOTE:	Tx Burst Size Sets the maximum burst size for Tx SDMA transactions: 000 = Burst is limited to 1 64-bit words. 001 = Burst is limited to 2 64-bit words. 010 = Burst is limited to 4 64-bit words. 011 = Burst is limited to 8 64-bit words. 100 = Burst is limited to 16 64-bit words. NOTE: This field effects only data-transfers. Descriptor fetch is done always with 4LW burst size. Must not be changed (other values degrade performance). How-ever, a larger value is optimal for DRAM performance.
31:25	Reserved	RES 0x0	Read Only

Table 515: IP Differentiated Services CodePoint 0 to Priority Registers (DSCP0)
Offset: Port0 0x2420, Port1 0x2820, Port2 0x2C20

Bits	Field	Type/ Init Val	Function
29:0	TOS_Q[29:0]	RW 0x0	The Priority queue mapping of received frames with DSCP values 0 (corresponding to TOS_Q[2:0]) through 9 (corresponding to TOS_Q[29:27]). **NOTE:** The initial value means that TOS does not effect queue decisions.
31:30	Reserved	RES 0x0	Reserved

Table 516: IP Differentiated Services CodePoint 1 to Priority Registers (DSCP1)
Offset: Port0 0x2424, Port1 0x2824, Port2 0x2C24

Bits	Field	Type/ Init Val	Function
29:0	TOS_Q[59:30]	RW 0x0	The Priority queue mapping of received frames with DSCP values 10 (corresponding to TOS_Q[32:30]) through 19 (corresponding to TOS_Q[59:57]). **NOTE:** The initial value means that TOS does not effect queue decisions.
31:30	Reserved	RES 0x0	Reserved

Table 517: IP Differentiated Services CodePoint 2 to Priority Registers (DSCP2, DSCP3, DSCP4, DSCP5)
Offset: Port0 DSCP2 0x2428, DSCP3 0x242C, DSCP4 0x2430, DSCP5 0x2434
Port1 DSCP2 0x2828, DSCP3 0x282C, DSCP4 0x2830, DSCP5 0x2834
Port2 DSCP2 0x2C28, DSCP3 0x2C2C, DSCP4 0x2C30, DSCP5 0x2C34

Bits	Field	Type/ Init Val	Function
29:0	TOS_Q[89:60]	RW 0x0	The Priority queue mapping of received frames with DSCP values 20 (corresponding to TOS_Q[62:60]) through 29 (corresponding to TOS_Q[89:87]). **NOTE:** The initial value means that TOS does not effect queue decisions.
31:30	Reserved	RES 0x0	Reserved

Table 518: IP Differentiated Services CodePoint 6 to Priority Registers (DSCP6)
Offset: Port0 0x2438, Port1 0x2838, Port2 0x2C38

Bits	Field	Type/ Init Val	Function
11:0	TOS_Q[191:180]	RW 0x0	The Priority queue mapping of received frames with DSCP values 60 (corresponding to TOS_Q[2:0]) through 63 (corresponding to TOS_Q[191:180]). **NOTE:** The initial value means that TOS does not effect queue decisions.
31:12	Reserved	RES 0x0	Reserved

When changing the value of the Port Serial Control register, the following steps must be taken:

- If any of the Tx DMAs were enabled before, they must be disabled through their command registers. The CPU must verify that they are disabled by polling the TxQ Command register, then the CPU must poll each port's Port Status register to verify that its Tx FIFOs is empty and that there is not transmission in progress.
- Read the Port Serial Control register.
- Disable the Serial port by writing a '0' to bit 0 (`PortEn`) of the Port Serial Control register.
- Set the desired bits in the Port Serial Control register.
- Enable the Serial port by writing a '1' to bit 0 (`PortEn`) of the Port Serial Control register.
- Re-enable the Tx DMAs, according to their initialization sequence, as necessary.

Table 519: Port Serial Control Registers (PSCR)
Offset: Port0 0x243C, Port1 0x283C, Port2 0x2C3C

Bits	Field	Type/ Init Val	Function
0	PortEn	RW 0x0	Serial Port Enable. No frames will be received of transmitted while serial port is disabled. 0 = Serial Port is disabled 1 = Serial Port is enabled NOTE: Disabling the port should not happen during Tx DMA operation. See guidelines above.
Link			
1	Force_Link_Pass	RW 0x0	Force Link status on port to Link UP state 1 = Force Link pass 0 = Do NOT Force Link Pass
Duplex			
2	AN_Duplex	RW 0x1 in 10-bit interface 0x0 in GMII interface	Enable Auto-Negotiation for duplex mode. 0 = Enable 1 = Disable In 10-bit Interface, this field must set to 0x1. NOTE: Half-Duplex is not supported in 1000 Mbps mode.
802.3x Flow Control and Backpressure			
3	AN_FC	RW 0x1	Enable Auto-Negotiation for Flow Control. 0 = Enable 1 = Disable When enabled, the MV64360/1/2 can either advertise no flow control or symmetric flow-control according to the Port Serial Control Register's `Pause_Adv` bit. Asymmetric flow-control advertisement is not supported.

Table 519: Port Serial Control Registers (PSCR) (Continued)
Offset: Port0 0x243C, Port1 0x283C, Port2 0x2C3C

Bits	Field	Type/ Init Val	Function
4	Pause_Adv	RW 0x1	Flow control advertise: 0 = Advertise no flow control. 1 = Advertise symmetric flow control support in Auto-Negotiation The MV64360/1/2 does not modify this bit as result of the Auto-Negotiation process (unlike the Port Status Register's `En_FC` bit which may be modified based on Auto-Negotiation results).
6:5	Force_FC_Mode	RW 0x0	The port will transmit Pause enable and disable frames depending on the CPU writing 00 and 01 values, conditioned with the flow-control operation enable as reflected in the PSR `En_Fc` bit being set in the following way: 00 = No Pause disable frames are sent. However, when the value of the field is changed by the CPU from 01 to 00, the port will send a single Pause enable packet (timer=0x0000) to enable the other side to transmit. 01 = When this field is set to 01 value, *and* Flow-control is enabled (The PSR `En_Fc` bit is set.) then Pause disable (timer=0xFFFF) are retransmitted at least every 4.2 msec (in GMII/10-bit mode) 42 msec (in MII mode at 100 MB) or 420 msec (in MII mode at 10 MB). Other values - Reserved NOTE: Only one mode is supported for enabling flow-control. When the link falls, this field goes to disabled 0x00 and must be reprogrammed only after the link is up.
8:7	Force_BP_ Mode	RW 0x0	When this bit is set, the port will start transmitting JAM on the line (Back-pressure) in Half-Duplex (which is only supported in 10/100 Mbps) according to the following settings: 00 = No JAM (no backpressure) 01 = JAM is transmitted continuously, on next frame boundary. 10 = Reserved 11 = Reserved NOTE: When the link falls, this field goes to disabled 0x00 and must be reprogrammed only after the link is up.
9	Reserved	RES 0x1	NOTE: This field must be set to '1'. Read/Write.
10	Force_Link_ Fail	RW 0x0	Force Link status on port to Link DOWN state. 0 = Force Link Fail 1 = Do NOT Force Link Fail
11	Retr_Forever	RW 0x0	Control the maximum number of retransmission attempts of a packet after collision. 0 = Retransmission attempts after collision on transmit are limited to 16 (as IEEE standard recommends). 1 = Retransmit forever (Required to enable Partition to take effect after 61 collisions on the same frame).

Table 519: Port Serial Control Registers (PSCR) (Continued)
Offset: Port0 0x243C, Port1 0x283C, Port2 0x2C3C

Bits	Field	Type/ Init Val	Function
12	Reserved	RES 0x0	Reserved. This field is Read/Write.
13	AN_Speed	RW 0x0 in GMII, 0x1 in 10-bit interface	Enable Auto-negotiation of interface speed in GMII mode. 0 = Enable update 1 = Disable update Auto-Negotiation on speed must not be enabled in 10-bit interface
14	DTE_Advert	RW 0x0	DTE advertise. The value of this bit is written to bit 9.10 of the 1000BaseT PHY device after power up or detection of a link failure.
15	AN_bypass_en	RW 0x0	This bit is valid in 10-bit interface only: Auto-Negotiation bypass enable. When enabled, if link partner does not respond to Auto-Negotiation process link is established by bypassing the Auto-Negotiation procedure. The default after bypassing Auto-Negotiation is Full-Duplex with no Flow-Control. 0 = Disable 1 = Enable
16	Restart_AN	RW 0x0	This bit is valid in 10-bit interface only. Setting this bit restarts Auto-Negotiation. This bit is reset by the MV64360/1/2 immediately after restarting the Auto-Negotiation process. 0 = No change. 1 = Restart.
19:17	MRU	RW 0x1	The Maximal Receive Packet Size: 0 = Accept packets up to 1518 bytes in length 1 = Accept packets up to 1522 bytes in length 2 = Accept packets up to 1552 bytes in length 3 = Accept packets up to 9022 bytes in length 4 = Accept packets up to 9192 bytes in length 5 = Accept packets up to 9700 bytes in length 6-7 = Reserved NOTE: Modes 3-5 are supported only when operating in 1000 Mbps. Receiving 9700 byte frames is supported *only* during 1000 Mbps operation. *Receiving frames over 2 KB during 100 Mbps operation may result in overrun/underrun in some cases*.
Loopback			
20	Reserved	RES 0x0	This field is reserved. Read/Write.
21	Set_FullDx	RW 0x1	Half/Full Duplex Mode 0 = Port works in Half Duplex 1 = Port works in Full Duplex NOTE: This bit is meaningless when the PSCR's `AN_Duplex` bit is set to enable. For 10-bit interface this bit must be set to disable.

**Table 519: Port Serial Control Registers (PSCR) (Continued)
Offset: Port0 0x243C, Port1 0x283C, Port2 0x2C3C**

Bits	Field	Type/ Init Val	Function
22	Set_FC_En	RW 0x1	Enable receiving and transmitting of 802.3x Flow Control frames in full duplex, Or enabling of backpressure in half duplex. 0 = Disabled 1 = Enabled NOTE: This bit is meaningless when this PSCR<AN_FC> is set to enable.
23	Set_GMII_ Speed	RW 0x1	Meaningful when the PSR's `Interface_Mode` field is GMII/MII only. 0 = Port works at 10/100 Mbps. 1 = Port works at 1000 Mbps. NOTE: This bit is meaningless when <AN_Speed> is set to enable. For 10-bit interface this bit must be set to disable.
24	Set_MII_Speed	RW 0x1	If Speed Auto-Negotiation is disabled <AN_Speed> = 0 and <Set_GMII_Speed> = 0, then this bit should set the speed of the MII interface. 0 = Port works in 10 Mbps 1 = Port works in 100 Mbps NOTE: This bit is meaningless when <AN_Speed> is enabled. For 10-bit interface this bit must be disabled.
31:25	Reserved	RO 0x0	Read Only

**Table 520: VLAN Priority Tag to Priority Registers (VPT2P)
Offset: Port0 0x2440, Port1 0x2840, Port2 0x2C40**

Bits	Field	Type/ Init Val	Function
23:0	Priority[23:0]	RW 0x0	The Priority queue mapping of received frames with 802.1p priority field values 0 (corresponding to Priority[2:0]) through 7 (corresponding to Priority[23:21]). **NOTE:** The initial value means that Priority does not effect the queue decisions.
31:24	Reserved	RES 0x0	Reserved

Table 521: Ethernet Port Status Registers (PSR)
Offset: Port0 0x2444, Port1 0x2844, Port2 0x2C44

Bits	Field	Type/ Init Val	Function
0	Interface_Mode	RO Sampled at reset, on pin TxD0[0] (for Port 0), and pin TxD1[0] (for Port 1), and TxD2[0] (for Port 2).	Interface mode 0 = GMII/MII 1 = 10-Bit Interface
Link			
1	Link_Up	RO 0x0	The Link Status 0 = Link is down 1 = Link is up This bit is set forced to ‘1’ when `Force_Link_Pass` = 1 This bit is set forced to ‘0’ when `Force_Link_Fail` = 0
Duplex			
2	FullDx	RW Determined by Auto-Negotiation for duplex mode, if the Port Control register’s `AN_Duplex` bit is enabled.	Half-/Full-Duplex mode. 0 = Port works in half-duplex mode 1 = Port works in full-duplex mode This bit may change in any time when the `AN_Duplex` bit is set to enable. When AN_Duplex in clear (disabled), this bit is set by the management in PSCR `Set_FullDx` Read Only. NOTE: Half-Duplex is not supported in 1000 Mbps.

Table 521: Ethernet Port Status Registers (PSR) (Continued)
Offset: Port0 0x2444, Port1 0x2844, Port2 0x2C44

Bits	Field	Type/ Init Val	Function
802.3x Flow Control and Backpressure			
3	En_FC	RO Set by Flow_Control Auto-Negotiation if PSCR<AN_FC> is enabled.	Enables receiving 802.3x Flow_Control frames in full-duplex mode: 0 = Disabled 1 = Enabled. If Port Control register's `An_Fc` bit is enabled, then each time that Auto-Negotiation is performed on the GMII/MII or TBI interface, the value in the `En_Fc` bit may change.
4	GMII_Speed	RO Determined by Auto-Negotiation for speed mode when AN_Speed is enabled.	Meaningful when `Interface_Mode` =GMII/MII only 0 = Port works in 10/100 Mbps 1 = Port works in 1000 Mbps If the `AN_Speed` bit is enabled, then each time that Auto-Negotiation is performed, the value in the `GMII_Speed` field may change. When this bit is 0, the `MII_Speed` defines whether it is 10 Mbps or 100 Mbps When the `AN_Speed` bit is disabled, this bit is set by the management in the PSCR `Set_GMII_Speed`.
5	MII_Speed	RO Auto-Negotiation for duplex mode when AN_Speed is enabled.	This field is meaningful only when `Interface_Mode` =GMII/MII and `GMII_Speed` = 10/100. 0 = Port works at 10 Mbps 1 = Port works at 100 Mbps If `AN_Speed` is enabled, then each time that Auto-Negotiation is performed, the value in the MII_Speed may change. When `AN_Speed` is disabled, and `Interface_Mode` is not set to 10-bit interface, this bit is set by the management in PSCR `Set_MII_Speed`.
7	TxInProg	RO 0x0	Transmit in Progress. Indicates that the port's transmitter is in an active transmission state.
8	Bypass_ Activated	RO 0x0	Link was established through the Auto-Negotiation bypass. Significant only in TBI mode and Auto-Negotiation enabled.
9	Partition	RO 0x0	Set when the port enters partition state.
10	TxFIFOEmp	RO 0x0	Set when the port Transmit FIFO is empty.
31:11	Reserved	RO 0x0	Read Only.

**Table 522: Transmit Queue Command Registers (TQCR)
Offset: Port0 0x2448, Port1 0x2848, Port2 0x2C48**

Bits	Field	Type/ Init Val	Function
7:0	ENQ	RW 0x0	Enable Queue [7:0] One bit per queue. Writing these bits set to ’1’ enables the queue. The transmit DMA will fetch the first descriptor programmed to the FDP register for that queue and start the transmit process. Writing ’1’ to the ENQ bit resets the matching DISQ bit. Writing ’1’ to the ENQ bit of a DMA that is already in enable state, has not effect. Writing ’0’ to the ENQ bit has no effect. When transmit DMA encounters queue end either by a null terminated descriptor pointer or a descriptor with a parity error or a CPU owned descriptor, The DMA will clear the ENQ bit for that queue. Thus reading these bits reports the active enable status for each queue. NOTE: The ENQ bits will be cleared on link down. After link is up, the CPU has to restart the DMA (set ENQ bits).
15:8	DISQ	RW 0x0	Disable Queue [7:0] One bit per each queue. Writing these bits set to ’1’ disables the queue. The transmit DMA will stop the transmit process from this queue on next packet boundary. Writing ’1’ to the DISQ bit resets the matching ENQ bit (when the DMA is finished with that specific queue and if the current active queue has been disabled). Writing ’0’ to the DISQ bit has no effect. When transmit DMA encounters queue end either by a null terminated descriptor pointer or by a CPU owned descriptor or by a descriptor with a parity error, the DMA will disable the queue but not set the DISQ bit for that queue, thus reading DISQ and ENQ bits discriminates between queues disables by the CPU and those stopped by DMA due to null pointer or CPU owned descriptor or parity error on descriptor. NOTE: The DISQ bits will be cleared on link down.
31:16	Reserved	RES 0x0	Read Only

**Table 523: Transmit Queue Fixed Priority Configuration Registers (TQFPC)
Offset: Port0 0x244C, Port1 0x284C, Port2 0x2C4C**

Bits	Field	Type/ Init Val	Function
7:0	FIXPR	RW 0xFF	Fixed priority queue [7:0]. Setting this bit to ’1’ configures the transmit queue as fixed priority. A ’0’ value means the queue is subject to the WRR algorithm (see 15.8.2 “Fixed Priority Mode” on page 235).
31:8	Reserved	RES 0x0	Reserved.

Table 524: Port Transmit Token-Bucket Rate Configuration Registers (PTTBRC)
Offset: Port0 0x2450, Port1 0x2850, Port2 0x2C50

Bits	Field	Type/ Init Val	Function
9:0	PTKNRT	RW 0x1023	Port Token Rate Configurable value 0–1023, in units of 1/64 byte, is added to queues Token-Bucket every 8 system clock cycles. The port-Token-Bucket is used to limit the port transmit bandwidth (see 15.8.3 “Weighted Round-Robin Priority Mode” on page 235 and 15.8.4 “Transmit Queue Bandwidth Limitation” on page 236). **NOTE:** The initial value actually means that there is no bandwidth limitation on the port.
31:10	Reserved	RES 0x0	Reserved.

Table 525: Maximum Transmit Unit Registers (MTU)
Offset: Port0 0x2458, Port1 0x2858, Port2 0x2C58

Bits	Field	Type/ Init Val	Function
5:0	MTU	RW 9 KB/256 = 36 (=0x24)	Maximum Transmit Unit Configurable value in 256-byte units, up to 16 Kbyte, used to implement the Token-Bucket bandwidth limitation. For port/queue that TKN-BKT>MTU, the port/queue bandwidth limitation is OFF (transmit enabled).
31:6	Reserved	RES 0x0	Reserved.

Table 526: Port Maximum Token Bucket Size Registers (PMTBS)
Offset: Port0 0x245C, Port1 0x285C, Port2 0x2C5C

Bits	Field	Type/ Init Val	Function
15:0	PMTBS	RW 0xFFFF	Port Maximum Token Bucket Size Configurable value in 256-byte units, used to implement the port Token-Bucket bandwidth limitation. The PTKNBKT is incremented by PTKNRT up to PMTBS*256*64. (Maximum accumulated transmit credit.)
31:16	Reserved	RES 0x0	Reserved.

Table 527: Port Interrupt Cause Registers (ICR)
Offset: Port0 0x2460, Port1 0x2860, Port2 0x2C60

NOTE: Write '0' to clear interrupt bits. Writing 1 does not effect the interrupt bits.

Bits	Field	Type/ Init Val	Function
0	RxBuffer	RW 0x0	Rx Buffer Return Indicates a Rx buffer returned to CPU ownership or that the port finished reception of a Rx frame in either priority queues. NOTE: To get a Rx Buffer return per priority queue, use bits [9:2]. These bits are set upon closing any Rx descriptor which has its `EI` bit set. To limit the interrupts to frame (rather than buffer) boundaries, the user should set the SDCR<RIFB> bit. When RIFB is set, an interrupt will be generated only upon closing the first descriptor of a received packet if this descriptor has its `EI` bit set.
1	Extend	RW 0x0	Interrupt Cause Extend register (ICERx) of this port has a bit set.
2	RxBuffer-Queue[0]	RW 0x0	Rx Buffer Return in Priority Queue[0] indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[0].
3	RxBuffer-Queue[1]	RW 0x0	Rx Buffer Return in Priority Queue[1] indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[1].
4	RxBuffer-Queue[2]	RW 0x0	Rx Buffer Return in Priority Queue[2] indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[2].
5	RxBuffer-Queue[3]	RW 0x0	Rx Buffer Return in Priority Queue[3] indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[3].
6	RxBuffer-Queue[4]	RW 0x0	Rx Buffer Return in Priority Queue[4] indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[4].
7	RxBuffer-Queue[5]	RW 0x0	Rx Buffer Return in Priority Queue[5] indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[5].
8	RxBuffer-Queue[6]	RW 0x0	Rx Buffer Return in Priority Queue[6] indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[6].
9	RxBuffer-Queue[7]	RW 0x0	Rx Buffer Return in Priority Queue[7] indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[7].
10	RxError	RW 0x0	Rx Resource Error indicates a Rx resource error event in either Rx priority queues. To get a Rx Resource Error Indication per priority queue, use bits [18:11].

Table 527: Port Interrupt Cause Registers (ICR) (Continued)
Offset: Port0 0x2460, Port1 0x2860, Port2 0x2C60

NOTE: Write ‘0’ to clear interrupt bits. Writing 1 does not effect the interrupt bits.

Bits	Field	Type/ Init Val	Function
11	RxError-Queue[0]	RW 0x0	Rx Resource Error in Priority Queue[0], indicates a Rx resource error event in receive priority queue[0].
12	RxError-Queue[1]	RW 0x0	Rx Resource Error in Priority Queue[1] indicates a Rx resource error event in receive priority queue[1].
13	RxError-Queue[2]	RW 0x0	Rx Resource Error in Priority Queue[2] indicates a Rx resource error event in receive priority queue[2].
14	RxError-Queue[3]	RW 0x0	Rx Resource Error in Priority Queue[3] indicates a Rx resource error event in receive priority queue[3].
15	RxError-Queue[4]	RW 0x0	Rx Resource Error in Priority Queue[0] indicates a Rx resource error event in receive priority queue[4].
16	RxError-Queue[5]	RW 0x0	Rx Resource Error in Priority Queue[1] indicates a Rx resource error event in receive priority queue[5].
17	RxError-Queue[6]	RW 0x0	Rx Resource Error in Priority Queue[2] indicates a Rx resource error event in receive priority queue[6].
18	RxError-Queue[7]	RW 0x0	Rx Resource Error in Priority Queue[3] indicates a Rx resource error event in receive priority queue[7].
19	TxEnd0	RW 0x0	Tx End for Priority Queue 0 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
20	TxEnd1	RW 0x0	Tx End for Priority Queue 1 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
21	TxEnd2	RW 0x0	Tx End for Priority Queue 2 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
22	TxEnd3	RW 0x0	Tx End for Priority Queue 3 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
23	TxEnd4	RW 0x0	Tx End for Priority Queue 4 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
24	TxEnd5	RW 0x0	Tx End for Priority Queue 5 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
25	TxEnd6	RW 0x0	Tx End for Priority Queue 6 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).

Table 527: Port Interrupt Cause Registers (ICR) (Continued)
Offset: Port0 0x2460, Port1 0x2860, Port2 0x2C60

NOTE: Write ‘0’ to clear interrupt bits. Writing 1 does not effect the interrupt bits.

Bits	Field	Type/ Init Val	Function
26	TxEnd7	RW 0x0	Tx End for Priority Queue 7 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
30:27	Reserved	RES 0x0	Reserved
31	EtherIntSum	RW 0x0	Ethernet Interrupt Summary This bit is a logical OR of the (unmasked) bits [30:0] in the Interrupt Cause register of the port.

Table 528: Port Interrupt Cause Extend Registers (ICER)
Offset: Port0 0x2464, Port1 0x2864, Port2 0x2C64

NOTE: Write ‘0’ To Clear interrupt bits. Writing 1 does not effect the interrupt bits.

Bits	Field	Type/ Init Val	Function
0	TxBuffer[0]	RW 0x0	Tx Buffer for Queue 0 Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame. NOTE: This bit is set upon closing any Tx descriptor which has its EI bit set. In order to limit the interrupts to frame (rather than buffer) boundaries, the user should set EI only in the last descriptor.
1	TxBuffer[1]	RW 0x0	Tx Buffer for Queue 1 Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame.
2	TxBuffer[2]	RW 0x0	Tx Buffer for Queue 2 Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame.
3	TxBuffer[3]	RW 0x0	Tx Buffer for Queue 3 Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame.
4	TxBuffer[4]	RW 0x0	Tx Buffer for Queue 4 Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame.
5	TxBuffer[5]	RW 0x0	Tx Buffer for Queue 5 Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame.
6	TxBuffer[6]	RW 0x0	Tx Buffer for Queue 6 Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame.

Table 528: Port Interrupt Cause Extend Registers (ICER) (Continued)
Offset: Port0 0x2464, Port1 0x2864, Port2 0x2C64
NOTE: Write '0' To Clear interrupt bits. Writing 1 does not effect the interrupt bits.

Bits	Field	Type/ Init Val	Function
7	TxBuffer[7]	RW 0x0	Tx Buffer for Queue 7 Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame.
8	TxError0	RW 0x0	Tx Resource Error for Priority Queue 0 Indicates a Tx resource error event during packet transmission from the queue.
9	TxError1	RW 0x0	Tx Resource Error for Priority Queue 1 Indicates a Tx resource error event during packet transmission from the queue.
10	TxError2	RW 0x0	Tx Resource Error for Priority Queue 2 Indicates a Tx resource error event during packet transmission from the queue.
11	TxError3	RW 0x0	Tx Resource Error for Priority Queue 3 Indicates a Tx resource error event during packet transmission from the queue.
12	TxError4	RW 0x0	Tx Resource Error for Priority Queue 4 Indicates a Tx resource error event during packet transmission from the queue.
13	TxError5	RW 0x0	Tx Resource Error for Priority Queue 5 Indicates a Tx resource error event during packet transmission from the queue.
14	TxError6	RW 0x0	Tx Resource Error for Priority Queue 6 Indicates a Tx resource error event during packet transmission from the queue.
15	TxError7	RW 0x0	Tx Resource Error for Priority Queue 7 Indicates a Tx resource error event during packet transmission from the queue.
16	PhySTC	RW 0x0	PHY Status Change Indicates a status change reported by the PHY connected to this port. If there is any change in the link, speed, duplex mode, or flow control capability as it is detected by the MDIO interface with the PHY, this interrupt will be set. This interrupt is set regardless of the actual changes in the status register, since auto-negotiation might be set to disabled on some of the parameters.
17	Reserved	RES 0x0	Reserved
18	RxOVR	RW 0x0	Rx Overrun Indicates an overrun event that occurred during reception of a packet.

Table 528: Port Interrupt Cause Extend Registers (ICER) (Continued)
Offset: Port0 0x2464, Port1 0x2864, Port2 0x2C64

NOTE: Write '0' To Clear interrupt bits. Writing 1 does not effect the interrupt bits.

Bits	Field	Type/ Init Val	Function
19	TxUdr	RW 0x0	Tx Underrun Indicates an underrun event that occurred during transmission of packet from either queue.
20	LinkChange	RW 0x0	Link State Change This bit is set by upon a change in the link state (down->up, up->down).
21	Partition	RW 0x0	Partition When set, indicates that the port entered partition state.
22	AutonegDone	RW 0x0	Auto-negotiation Done. Set when the port completed the auto-negotiation process, in 10-bit interface (Ten Bit Interface — TBI) mode *only*
23	Internal_addr_error	RW 0x0	Internal Address Error is set when there is an access to an illegal offset of the internal registers. When set, the Internal Address Error register locks the address that caused the error.
30:24	Reserved	RES 0x0	Reserved.
31	EtherIntSum	RW 0x0	Ethernet Interrupt Extend Summary This bit is a logical OR of the (unmasked) bits [30:0] in the Interrupt Cause Extend register.

Table 529: Port Interrupt Mask Registers (PIMR)
Offset: Port0 0x2468, Port1 0x2868, Port2 0x2C68

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Mask bits for Interrupt Cause register 0 = Mask 1 = Do not mask

Table 530: Port Extend Interrupt Mask Registers (PEIMR)
Offset: Port0 0x246C, Port1 0x286C, Port2 0x2C6C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Mask bits for Port Extend Interrupt Cause register 0 = Mask 1 = Do not mask

**Table 531: Portx Rx FIFO Urgent Threshold Registers (PxRFUTR)
Offset: Port0 0x2470, Port1 0x2870, Port2 0x2C70**

Bits	Field	Type/ Init Val	Function
3:0	RxUThreshold	RW 0x8	Contain the Rx FIFO Threshold to start an Urgent indication. 0x0 = Disable Urgent. 0x1-0xF = Threshold for Urgent is 8–120 entries (64–960 byte)
31:4	Reserved	RES 0x0	Read Only

**Table 532: Portx Tx FIFO Urgent Threshold Registers (PxTFUTR)
Offset: Port0 0x2474, Port1 0x2874, Port2 0x2C74**

Bits	Field	Type/ Init Val	Function
3:0	Reserved	RES 0x0	Must be 0.
17:4	IPG_Int_Tx	RW 0x0	Tx frame IPG between interrupt counter and enable. This field provides a way to force a delay from the last ICRE[TxBuffer] interrupt, from any of the queues, to the next TxBuffer interrupt, from any of the queues. The ICRE bits still reflect the new interrupt, but this masking is reflected by potentially not propagating to the Global_Interrupt register in the PCI or CPU units. This provides a way for interrupt coalescing on receive packet events. The time is calculated in multiples of 64 clock cycles. Valid values are 0 (no delay between packets to CPU, the counter is effectively disabled), through 0x3FFF (1,048,544 clock cycles). See the detailed description in 15.7.1 “Interrupt Coalescing” on page 234.
31:18	Reserved	RO 0x0	Read Only

**Table 533: Portx Rx Minimal Frame Size Registers (PxMFSR)
Offset: Port0 0x247C, Port1 0x287C, Port2 0x2C7C**

Bits	Field	Type/ Init Val	Function
1:0	RxMFS[1:0]	RO 0x0	Read Only.
6:2	RxMFS[6:2]	RW 0x10 (Corresponding to 64 bytes)	Contain the Receive Minimal Frame Size in bytes. Valid Range of RxMFS [6:2] is 0xA–0x10. (RxMFS = 40,44,48,52,56,60,64 bytes)
31:7	Reserved	RO 0x0	Read Only

Table 534: Portx Rx Discard Frame Counter Registers (PxDFC)
Offset: Port0 0x2484, Port1 0x2884, Port2 0x2C84

Bits	Field	Type/ Init Val	Function
31:0	Rx Discard[31:0]	RO 0x0	Number of frames that were discarded because of address filtering or resource error. This register is reset every time the CPU reads from it.

Table 535: Portx Overrun Frame Counter Registers (PxOFC)
Offset: Port0 0x2488, Port1 0x2888, Port2 0x2C88

Bits	Field	Type/ Init Val	Function
31:0	Rx Overrun[31:0]	RO 0x0	Number of frames that were received and overrun This register is reset every time the CPU reads from it.

Table 536: Port Internal Address Error Registers (EUIAER)
Offset: Port0 0x2494, Port1 0x2894, Port2 0x2C94

Bits	Field	Type/ Init Val	Function
8:0	InternalAddress	RO	Locks the relevant internal address bits (bit 12 and bits 9:2), if there is an address violation of unmapped access to the port registers. The Address is locked until the register is read. (This field is used for software debugging after an address violation interrupt is raised.)
31:9	Reserved	RES 0x0	Reserved

Table 537: Ethernet Current Receive Descriptor Pointer Registers (CRDP)
Offset: Port0: Q0 0x260C, Q1 0x261C, Q2 0x262C, Q3 0x263C, Q4 0x264C, Q5 0x265C, Q6 0x266C, Q7 0x267C
Port1: Q0 0x2A0C, Q1 0x2A1C, Q2 0x2A2C, Q3 0x2A3C, Q4 0x2A4C, Q5 0x2A5C, Q6 0x2A6C, Q7 0x2A7C
Port2: Q0 0x2E0C, Q1 0x2E1C, Q2 0x2E2C, Q3 0x2E3C, Q4 0x2E4C, Q5 0x2E5C, Q6 0x2E6C, Q7 0x2E7C

Bits	Field	Type/ Init Val	Function
31:0	RxCDP	RW 0x0	Receive Current Queue Descriptor Pointer. **NOTE:** See J.2 “IDMA Descriptor Registers” on page 662.

Table 538: Receive Queue Command (RQCR)
Offset: Port0: 0x2680, Port1: 0x2A80, Port2: 0x2E80

Bits	Field	Type/ Init Val	Function
7:0	ENQ	RW 0x0	Enable Queue [7:0] One bit per each queue. Writing these bits set to '1' enables the queue. The Receive DMA will fetch the first descriptor programmed to the RxCDP register for that queue and start the Receive process. Writing '1' to ENQ bit resets the matching DISQ bit. Writing '1' to ENQ bit of a DMA that is already in enable state, has not effect. Writing '0' to ENQ bit has no effect. When the receive DMA encounters a queue ended by a null terminated descriptor pointer or a descriptor with a parity error, the DMA will clear the ENQ bit for that queue. Thus reading these bits reports the active enable status for each queue. NOTE: Reaching CPU owned descriptor, null terminated descriptor or descriptor that is read with a parity error in the middle of a packet will result in closing the status of the packet with a resource error condition. Reaching CPU owned descriptor either in the middle or on start of new packet will not result in disabling the DMA, and DMA will continue to read the descriptor it is using, when a new packet arrives to this queue until it gets ownership of it. For these cases – not owned descriptor, null terminated descriptor, or a parity error on descriptor – the DMA will assert the resource RxErrorQueue interrupt.
15:8	DISQ	RW 0x0	Disable Queue [7:0] One bit per each queue.Writing these bits set to '1' disables the queue. The transmit DMA will stop the Receive process to this queue, on the next packet boundary. Writing '1' to DISQ bit resets the matching ENQ bit after the RxDMA finished processing the queue, if the ENQ bit was used while the CPU wrote the DISQ for it. Writing '0' to DISQ bit has no effect. When transmit DMA encounters a queue ended either by a null terminated descriptor pointer or a descriptor with a parity error, the DMA will disable the queue but not set the DISQ bit for that queue, thus reading DISQ and ENQ bits discriminates between queues disables by CPU and those stopped by DMA due to a null pointer or a parity error on descriptor.
31:16	Reserved	RO 0x0	Read only.

Table 539: Transmit Current Served Descriptor Pointer Register
Offset: Port0 0x2684, Port1 0x2A84, Port2 0x2E84

Bits	Field	Type/ Init Val	Function
31:0	TxCDP	RO 0x0	Transmit Current Descriptor Pointer. (See the Tx descriptor restrictions in Section 18. “IDMA Controller” on page 301.)

Table 540: Transmit Current Queue Descriptor Pointer (TCQDP)
Offset: Port0: Q0 0x26C0, Q1 0x26C4, Q2 0x26C8, Q3 0x26CC, Q4 0x26D0, Q5 0x26D4, Q6 0x26D8, Q7 0x26DC
Port1: Q0 0x2AC0, Q1 0x2AC4, Q2 0x2AC8, Q3 0x2ACC, Q4 0x2AD0, Q5 0x2AD4, Q6 0x2AD8, Q7 0x2ADC
Port2: Q0 0x2EC0, Q1 0x2EC4, Q2 0x2EC8, Q3 0x2ECC, Q4 0x2ED0, Q5 0x2ED4, Q6 0x2ED8, Q7 0x2EDC

Bits	Field	Type/ Init Val	Function
31:0	TxCDP	RW 0x0	Transmit Current Queue Descriptor Pointer. (See J.2 “IDMA Descriptor Registers” on page 662.)

Table 541: Transmit Queue Token-Bucket Counter (TQxTBC)
Offset: Port0: Q0 0x2700, Q1 0x2710, Q2 0x2720, Q3 0x2730, Q4 0x2740, Q5 0x2750, Q6 0x2760, Q7 0x2770
Port1: Q0 0x2B00, Q1 0x2B10, Q2 0x2B20, Q3 0x2B30, Q4 0x2B40, Q5 0x2B50, Q6 0x2B60, Q7 0x2B70
Port2: Q0 0x2F00, Q1 0x2F10, Q2 0x2F20, Q3 0x2F30, Q4 0x2F40, Q5 0x2F50, Q6 0x2F60, Q7 0x2F70

Bits	Field	Type/ Init Val	Function
29:0	TKNBKT	RW Undefined. Must be initialized. See 15.9 “Token Rate Configuration” on page 237.	Token Bucket Byte counter, in units of 1/64 byte, incremented by TFNRT and decremented by transmitted bytes x 64 from this queue. The CPU may write to this counter to override with Token-bucket operation.
31:30	Reserved	RO 0x0	Reserved.

Table 542: Transmit Queue Token Bucket Configuration (TQxTBC)
Offset: Port0: Q0 0x2704, Q1 0x2714, Q2 0x2724, Q3 0x2734, Q4 0x2744, Q5 0x2754, Q6 0x2764, Q7 0x2774
Port1: Q0 0x2B04, Q1 0x2B14, Q2 0x2B24, Q3 0x2B34, Q4 0x2B44, Q5 0x2B54, Q6 0x2B64, Q7 0x2B74
Port2: Q0 0x2F04, Q1 0x2F14, Q2 0x2F24, Q3 0x2F34, Q4 0x2F44, Q5 0x2F54, Q6 0x2F64, Q7 0x2F74

Bits	Field	Type/ Init Val	Function
9:0	TKNRT	RW Undefined Must be initialized. See Table 524 on page 614.	Token Rate Legal values are between 0--1023. The Token Rate, in units of 1/64 byte, is added to queues Token-Bucket every 8 system clock cycles. The Token-Bucket is used to limit the transmit bandwidth per queue (see 15.8.3 “Weighted Round-Robin Priority Mode” on page 235 and 15.8.4 “Transmit Queue Bandwidth Limitation” on page 236).
25:10	MTBS	RW Undefined Must be initialized.	Maximum Token Bucket Size Configurable value in 256-byte units, used to implement the Token-Bucket bandwidth limitation. The TKNBKT is incremented by TKNRT up to MTBS*256*64. (Maximum accumulated transmit credit.)
31:26	Reserved	RO 0x0	Reserved.

Table 543: Transmit Queue Arbiter Configuration (TQxAC)
Offset: Port0: Q0 0x2708, Q1 0x2718, Q2 0x2728, Q3 0x2738, Q4 0x2748, Q5 0x2758, Q6 0x2768, Q7 0x2778
Port1: Q0 0x2B08, Q1 0x2B18, Q2 0x2B28, Q3 0x2B38, Q4 0x2B48, Q5 0x2B58, Q6 0x2B68, Q7 0x2B78
Port2: Q0 0x2F08, Q1 0x2F18, Q2 0x2F28, Q3 0x2F38, Q4 0x2F48, Q5 0x2F58, Q6 0x2F68, Q7 0x2F78

Bits	Field	Type/ Init Val	Function
7:0	WRRWGT	RW Undefined. Must be initialized. See 15.8 “Transmit Weighted Round-Robin Arbitration” on page 234.	Transmit Weighted-Round-Robin Weight The configurable value of WRR weight is 256 bytes per unit, spans 0–64K byte in 256-bytes increments. The transmit bandwidth assigned to a queue is equal to WRRWGT/(Sum of all "non fixed priority" plus "maximum limited" WRRWGTs) part of remaining bandwidth (not consumed by fixed priority). See 15.8.3 “Weighted Round-Robin Priority Mode” on page 235.

Table 543: Transmit Queue Arbiter Configuration (TQxAC) (Continued)
Offset: Port0: Q0 0x2708, Q1 0x2718, Q2 0x2728, Q3 0x2738, Q4 0x2748, Q5 0x2758, Q6 0x2768, Q7 0x2778
Port1: Q0 0x2B08, Q1 0x2B18, Q2 0x2B28, Q3 0x2B38, Q4 0x2B48, Q5 0x2B58, Q6 0x2B68, Q7 0x2B78
Port2: Q0 0x2F08, Q1 0x2F18, Q2 0x2F28, Q3 0x2F38, Q4 0x2F48, Q5 0x2F58, Q6 0x2F68, Q7 0x2F78

Bits	Field	Type/ Init Val	Function
24:8	WRR_BC[16:0]	RW Initialized by the CPU to ’0’.	Internal counter for implementing the WRR algorithm The CPU should not modify this field after writing the initialization value and enabling the port.
31:25	Reserved	RES 0x0	Reserved.

Table 544: Port Transmit Token-Bucket Counter (PTTBC)
Offset: Port0 0x2780, Port1 0x2B80, Port2 0x2F80

Bits	Field	Type/ Init Val	Function
29:0	PTKNBKT	RW 0x3FFFFFFF	Port Token Bucket Byte counter, in units of 1/64 byte, incremented by PTKNRT (up to PMTBS*256*64) and decremented by transmitted bytes x 64 from this port. The CPU may write to this counter to override with Token-bucket operation. NOTE: The initial value is set to its maximum value so the port can start transmission immediately.
31:30	Reserved	RO 0x0	Reserved.

Table 545: Destination Address Filter Special Multicast Table (DFSMT)
Offset: Port0: 0x3400–0x34FC, Port1: 0x3800–0x38FC, Port2: 0x3C00–0x3CFC
NOTE: Every register holds four entries. A total of 64 registers appear in the table in consecutive order.

Bits	Field	Type/ Init Val	Function
0	Pass[0]	RW N/A	Determines whether to filter or accept, for pointer index ’0’. 0 = Reject (filter) frame 1 = Accepts frame
3:1	Queue[0]	RW N/A	For pointer index ’0’: Determines the Queue number if Pass[0]=1.
4	Reserved[0]	RES N/A	Reserved. Must be set to zero.
7:5	Unused[0]	RES N/A	Reserved.

Table 545: Destination Address Filter Special Multicast Table (DFSMT) (Continued)
Offset: Port0: 0x3400–0x34FC, Port1: 0x3800–0x38FC, Port2: 0x3C00–0x3CFC

NOTE: Every register holds four entries. A total of 64 registers appear in the table in consecutive order.

Bits	Field	Type/ Init Val	Function
8	Pass[1]	RW N/A	Determines whether to filter or accept, for pointer index ’1’. 0 = Reject (filter) frame 1 = Accepts frame
11:9	Queue[1]	RW N/A	For pointer index ’1’: Determines the Queue number if Pass[1]=1.
12	Reserved[1]	RES N/A	Reserved. Must be set to zero.
15:13	Unused[1]	RES N/A	Reserved.
16	Pass[2]	RW N/A	Determines whether to filter or accept, for pointer index ’2’. 0 = Reject (filter) frame 1 = Accepts frame
19:17	Queue[2]	RW N/A	For pointer index ’2’: Determines the Queue number if Pass[2]=1.
20	Reserved[2]	RES N/A	Reserved. Must be set to zero.
23:21	Unused[2]	RES N/A	Reserved.
24	Pass[3]	RW N/A	Determines whether to filter or accept, for pointer index ’3’. 0 = Reject (filter) frame 1 = Accept frame
27:25	Queue[3]	RW N/A	For pointer index ’0’: Determines the Queue number if Pass[3]=1.
28	Reserved[3]	RES N/A	Reserved. Must be set to zero.
31:29	Unused[3]	RES N/A	Reserved.

Table 546: Destination Address Filter Other Multicast Table (DFOMT)
Offset: Port0: 0x3500–0x35FC, Port1: 0x3900–0x39FC, Port2: 0x3D00–0x3DFC

NOTE: Every register holds four entries. A total of 64 registers appear in this table in consecutive order.

Bits	Field	Type/ Init Val	Function
0	Pass[0]	RW N/A	Determines whether to filter or accept, for pointer index ’0’. 0 = Reject (filter) frame 1 = Accepts frame

Table 546: Destination Address Filter Other Multicast Table (DFOMT) (Continued)
Offset: Port0: 0x3500–0x35FC, Port1: 0x3900–0x39FC, Port2: 0x3D00–0x3DFC
NOTE: Every register holds four entries. A total of 64 registers appear in this table in consecutive order.

Bits	Field	Type/ Init Val	Function
3:1	Queue[0]	RW N/A	For pointer index ’0’: Determines the Queue number if Pass[0]=1.
4	Reserved[0]	RES N/A	Reserved. Must be set to zero.
7:5	Unused[0]	RES N/A	Reserved.
8	Pass[1]	RW N/A	Determines whether to filter or accept, for pointer index ’1’. 0 = Reject (filter) frame 1 = Accepts frame
11:9	Queue[1]	RW N/A	For pointer index ’1’: Determines the Queue number if Pass[1]=1.
12	Reserved[1]	RES N/A	Reserved. Must be set to zero.
15:13	Unused[1]	RES N/A	Reserved.
16	Pass[2]	RW N/A	Determines whether to filter or accept, for pointer index 2. 0 = Reject (filter) frame 1 = Accepts frame
19:17	Queue[2]	RW N/A	For pointer index ’2’: Determines the Queue number if Pass[2]=1.
20	Reserved[2]	RES N/A	Reserved. Must be set to zero.
23:21	Unused[2]	RES N/A	Reserved.
24	Pass[3]	RW N/A	Determines whether to filter or accept, for pointer index ’3’. 0 = Reject (filter) frame 1 = Accepts frame
27:25	Queue[3]	RW N/A	For pointer index ’3’: Determines the Queue number if Pass[3]=1.
28	Reserved[3]	RES N/A	Reserved. Must be set to zero.
31:29	Unused[3]	RES N/A	Reserved.

Table 547: Destination Address Filter Unicast Table (DFOMT)
Offset: Port0: 0x3600–0x360C, Port1: 0x3A00–0x3A0C, Port2: 0x3E00–0x3E0C

NOTE: Every register holds four entries. A total of four registers appear in this table in consecutive order.

Bits	Field	Type/ Init Val	Function
0	Pass[0]	RW N/A	Determines whether to filter or accept, for pointer index ’0’. 0 = Reject (filter) frame 1 = Accepts frame
3:1	Queue[0]	RW N/A	For pointer index ’0’: Determines the Queue number if Pass[0]=1.
4	Reserved[0]	RW N/A	Reserved. Must be set to zero.
7:5	Unused[0]	RW N/A	Reserved.
8	Pass[1]	RW N/A	Determines whether to filter or accept, for pointer index ’1’. 0 = Reject (filter) frame 1 = Accepts frame
11:9	Queue[1]	RW N/A	For pointer index ’1’: Determines the Queue number if Pass[1]=1.
12	Reserved[1]	RW N/A	Reserved. Must be set to zero.
15:13	Unused[1]	RES N/A	Reserved.
16	Pass[2]	RW N/A	Determines whether to filter or accept, for pointer index 2. 0 = Reject (filter) frame 1 = Accepts frame
19:17	Queue[2]	RW N/A	For pointer index ’2’: Determines the Queue number if Pass[2]=1.
20	Reserved[2]	RES N/A	Reserved. Must be set to zero.
23:21	Unused[2]	RES N/A	Reserved.
24	Pass[3]	RW N/A	Determines whether to filter or accept, for pointer index ’3’. 0 = Reject (filter) frame 1 = Accepts frame
27:25	Queue[3]	RW N/A	For pointer index ’3’: Determines the Queue number if Pass[3]=1.
28	Reserved[3]	RES N/A	Reserved. Must be set to zero.
31:29	Unused[3]	RES N/A	Reserved.

G.5 Port MIB Counter

Table 548: MAC MIB Counters
Offset: Port0: 0x003000–0x00307C, Port1: 0x003080–0x0030FC, Port2: 0x003100–0x00317C

Index	Offset	Width	Counter Name	Description
0	0x0	64	GoodOctetsReceived	The sum of lengths of all good Ethernet frames received: frames that are not Bad frames NOR MAC Control frames. NOTE: This does *not* include 802.3x pause messages, but, does include bridge control packets like LCAP and BPDU.
1	0x8	32	BadOctetsReceived	The sum of lengths of all bad Ethernet frames received
2	0x10	32	GoodFramesReceived	The number of Ethernet frames received that are not Bad Ethernet frames or MAC Control packets. NOTE: This does include Bridge Control packets.
3	0x14	32	BadFramesReceived	The number of bad Ethernet frames received
4	0x18	32	BroadcastFrames Received	The number of good frames received that had a Broadcast destination MAC address
5	0x1C	32	MulticastFramesReceived	The number of good frames received that had a Multicast destination MAC address NOTE: This does *not* include 802.3 Flow Control messages as they are considered MAC Control messages.
6	0x20	32	Frames64Octets	The total number of received and transmitted, Good and Bad frames that are 64 bytes in size or are between the minimum-size (as specified in RxMFS in the MFSR register) and 64 bytes. NOTE: This does *not* include MAC Control frames.
7	0x24	32	Frames65to127Octets	The total number of received and transmitted, Good and Bad frames that are 65 to 127 bytes in size. NOTE: This does *not* include MAC Control frames.
8	0x28	32	Frames128to255Octets	The total number of received and transmitted, Good and Bad frames that are 128 to 255 bytes in size. NOTE: This does *not* include MAC Control frames.
9	0x2C	32	Frames256to511Octets	The total number of received and transmitted, Good and Bad frames that are 256 to 511 bytes in size. NOTE: This does *not* include MAC Control frames.
10	0x30	32	Frames512to1023Octets	The total number of received and transmitted, Good and Bad frames that are 512 to 1023 bytes in size. NOTE: This does *not* include MAC Control frames.
11	0x34	32	Frames1024toMaxOctets	The total number of received and transmitted, Good and Bad frames that are more than 1023 bytes in size and less than the MRU. NOTE: This does not include MAC Control frames.

Table 548: MAC MIB Counters (Continued)
Offset: Port0: 0x003000–0x00307C, Port1: 0x003080–0x0030FC, Port2: 0x003100–0x00317C

Index	Offset	Width	Counter Name	Description
12	0x38	64	GoodOctetsSent	The sum of lengths of all good Ethernet frames sent from this MAC. This does not include 802.3 Flow Control frames NOR packets dropped due to excessive collision NOR packets with an Tx Error Event.
13	0x40	32	GoodFramesSent	The number of Ethernet frames sent from this MAC. This does not include 802.3 Flow Control frames NOR packets dropped due to excessive collision NOR packets with an Tx Error Event.
14	0x48	32	MulticastFramesSent	The number of good frames sent that had a Multicast destination MAC address. NOTE: This does NOT include 802.3 Flow Control messages, as they are considered MAC Control messages, NOR does it include packets with an Tx Error Event. This counter counts frames of all sizes, including frames smaller than the Minimal Frame Size and frames larger than the MRU.
15	0x4C	32	BroadcastFramesSent	The number of good frames sent that had a Broadcast destination MAC address. This does not include 802.3 Flow Control frames NOR packets dropped due to excessive collision NOR packets with an Tx Error Event. NOTE: This counter counts frames of all sizes, including frames smaller than the Minimal Frame Size and frames larger than the MRU.
16	0x50	32	UnrecogMACControl Received	The number of received MAC Control frames that have an opcode different than 00-01.
17	0x58	32	GoodFCReceived	The number of good flow control messages received
18	0x5C	32	BadFCReceived	The number of bad flow control frames received
19	0x60	32	Undersize	The number of undersize packets received
20	0x64	32	Fragments	The number of fragments received
21	0x68	32	Oversize	The number of oversize packets received
22	0x6C	32	Jabber	The number of jabber packets received
23	0x70	32	MACRcvError	The number of Rx Error events seen by the receive side of the MAC
24	0x74	32	BadCRC	The number CRC error events
25	0x78	32	Collisions	The number of collision events seen by the MAC
26	0x7C	32	Late Collision	The number of late collisions seen by the MAC

Note
The 802dot3SingleCollisionFrames and 802dot3MultipleCollisionFrames are not implemented.

Appendix H. Communication Unit Interface Registers

H.1 Communication Unit Register Maps

Table 549: Address Decoding Register Map

Register	Offsets	Page
Base Address 0-3	Base Address 0 0xF200, Base Address 1 0xF208, Base Address 2 0xF210, Base Address 3 0xF218	Table 558, p.635
Size Register 0-3	Size Register 0 0xF204, Size Register 1 0xF20C, Size Register 2 0xF214, Size Register 3 0xF21C	Table 559, p.637
High Address Remap 0-1	High Address Remap 0 0xF240, High Address Remap 1 0xF244	Table 560, p.637
Base Address Enable	0xF250	Table 561, p.637
MPSC0/1 Access Protect	MPSC0 0xF254, MPSC1 0xF258	Table 562, p.638
Internal Space Base Address	0xF25C	Table 563, p.638

Table 550: Cunit Control Register Map

Register	Offset	Page
Cunit Arbiter Control	0xF300	Table 564, p.639
Cunit Configuration (CUACR)	0xB40C	Table 565, p.639
Cunit Crossbar Time out	0xF304	Table 566, p.640

Table 551: Error Report Register Map

Register	Offset	Page
Cunit Interrupt Cause	0xF310	Table 567, p.640
Cunit Interrupt Mask	0xF314	Table 568, p.641
Cunit Error Address	0xF318	Table 569, p.641

Table 552: MPSCs Clocks Routing Register Map

Register	Offset	Page
MPSC Routing (MRR)	0xB400	Table 570, p.642
Rx Clock Routing (RCR)	0xB404	Table 571, p.642
Tx Clock Routing (TCR)	0xB408	Table 572, p.643

Table 553: MPSCs Interrupts Register Map

Register	Offsets	Page
MPSC Cause	MPSC0 0xB804, MPSC1 0xB80C	Table 573, p.643
MPSC Mask	MPSC0 0xB884, MPSC1 0xB88C	Table 574, p.644

Table 554: MPSCx Main Configuration Register Map

Register	Offsets	Page
MPSCx Main Configuration (Low) (MMCRLx)	MPSC0 0x8000, MPSC1 0x9000	Table 575, p.646
MPSCx Main Configuration (High) (MMCRHx)	MPSC0 0x8004, MPSC1 0x9004	Table 576, p.650

Table 555: MPSCx Protocol Configuration Register Map

Register	Offsets	Page
MPSCx Protocol Configuration (MPCx) for HDLC	MPSC0 0x8008, MPSC1 0x9008	Table 72, p.257
MPSCx Protocol Configuration (MPCRx) for BISYNC		Table 85, p.267
MPSCx Protocol Configuration (MPCRx) for UART Mode		Table 92, p.277

Table 555: MPSCx Protocol Configuration Register Map (Continued)

Register	Offsets	Page
Channel0 Register1 (CH0R1)	MPSC0 0X800C, MPSC1 0X900C	The functionality of each CHxRx is protocol dependent. Detailed descriptions of the CHRs are given in the following sections: • 16.6 “HDLC Mode” on page 256 • 16.7 “BISYNC Mode” on page 263 • 16.8 “UART Mode” on page 275 • 16.9 “Transparent Mode” on page 284
Channel0 Register2 (CH0R2)	MPSC0 0x8010, MPSC1 0x9010	
Channel0 Register3 (CH0R3)	MPSC0 0x8014, MPSC1 0x9014	
Channel0 Register4 (CH0R4)	MPSC0 0x8018, MPSC1 0x9018	
Channel0 Register5 (CH0R5)	MPSC0 0X801C, MPSC1 0X901C	
Channel0 Register6 (CH0R6)	MPSC0 0x8020, MPSC1 0x9020	
Channel0 Register7 (CH0R7)	MPSC0 0x8024, MPSC1 0x9024	
Channel0 Register8 (CH0R8)	MPSC0 0x8028, MPSC1 0x9028	
Channel0 Register9 (CH0R9)	MPSC0 0X802C, MPSC1 0X902C	
Channel0 Register10 (CH0R10)	MPSC0 0x8030, MPSC1 0x9030	

Table 556: SDMA Register Map

Register	Offsets	Page
SDMA Configuration Register (SDCx)	Channel0 0x4000, Channel1 0x6000	Table 577, p.652
SDMA Command (SDCMx)	Channel0 0x4008, Channel1 0x6008	Table 578, p.654
Channel0 Rx Descriptor	Channel0 0x4800 - 0x480f, Channel1 0x6800 - 0x680f	Not to be accessed during normal operation.
Channelx Current Rx Descriptor Pointer (SCRDPx)	Channel0 0x4810, Channel1 0x6810	16.15.1 “SDMA Current Receive Descriptor Pointer (SCRDP)” on page 294
Channel0 Tx Descriptor	Channel0 0x4c00 - 0x4c0f, Channel1 0x6c00 - 0x6c0f	Not to be accessed during normal operation.
Channelx Current Tx Descriptor Pointer (SCTDPx)	Channel0 0x4c10, Channel1 0x6c10	16.15.2 “SDMA Current Transmit Descriptor Pointer (SCTDP)” on page 294

Table 556: SDMA Register Map (Continued)

Register	Offsets	Page
Channelx First Tx Descriptor Pointer (SFTDPx)	Channel0 0x4c14, Channel1 0x6c14	16.15.3 “SDMA First Transmit Descriptor Pointer (SFTDP)” on page 294

Table 557: SDMA Interrupts Register Map

Register	Offset	Page
SDMA Cause	0xB800	Table 579, p.655
SDMA Mask	0xB880	Table 580, p.656

H.2 Address Decoding Registers

Table 558: Base Address 0-3
Offset: Base Address 0 0xF200, Base Address 1 0xF208, Base Address 2 0xF210, Base Address 3 0xF218

Bits	Field	Type/ Init Val	Function
3:0	Target	RW 0x0	Specifies the target interface associated with this window: 0x0 = DRAM 0x1 = Device 0x2 = CPU bus (60x bus only) or integrated SRAM 0x3 = PCI_0 0x4 = PCI_1 0x5 – 0xF = Reserved **NOTE:** 0x2 and 0x4 are only valid for the MV64360 and MV64361. Reserved in the MV64362 device.
7:4	Reserved	RES 0x0	Reserved.

Table 558: Base Address 0-3 (Continued)
Offset: Base Address 0 0xF200, Base Address 1 0xF208, Base Address 2 0xF210, Base Address 3 0xF218

Bits	Field	Type/ Init Val	Function
15:8	Attr	RW 0x0	Specifies target specific attributes depending on the target interface. **DRAM Target Interface** Bits[11:8] select the DRAM bank: 0xe = CS[0]# 0xd = CS[1]# 0xb = CS[2]# 0x7 = CS[3]# **NOTE:** All other bits [11:8] values are Reserved. Bits[13:12] specify the cache coherency support: 0x0 = No cache coherency 0x1 = WT cache coherency 0x2 = WB cache coherency 0x3 = Reserved **NOTE:** Bits[15:14] - Reserved **Device Bus Target Interface** Bits[12:8] select the Device bank: 0x1e = DevCS[0]# 0x1d = DevCS[1]# 0x1b = DevCS[2]# 0x17 = DevCS[3]# 0x0f = BootCS# **NOTE:** All other bits[12:8] values are Reserved. Bits[15:13] - Reserved **PCI0/1 Target Interface** (PCI_1 only applies to the MV64360 and MV64361.) Bits[9:8] specifies the Data swap type: 0x0 = Byte swap 0x1 = No swap 0x2 = Both byte and Word swap 0x3 = Word swap Bit[10] determines the PCI-X No Snoop (NS) attribute: 0x0 = NS attribute is not asserted 0x1 = NS attribute is asserted Bit[11] selects PCI I/O or memory space: 0x0 = I/O 0x1 = Memory Bit[12] selects the PCI REQ64# control: (Only applies to the MV64360 and MV64362.) 0x0 = Forces the PCI master to always assert REQ64#. 0x1 = Asserts REQ64# according to the requested data size. Bit[15:13] - Reserved If the target interface is integrated SRAM or 60x bus: Bits[10:8,15:12] - Reserved Bit[11] - Integrated SRAM or 60x bus select: (Integrated SRAM only applies to the MV64360 and MV64361.) 0x0 = Integrated SRAM 0x1 = 60x bus

Table 558: Base Address 0-3 (Continued)
Offset: Base Address 0 0xF200, Base Address 1 0xF208, Base Address 2 0xF210, Base Address 3 0xF218

Bits	Field	Type/ Init Val	Function
31:16	Base	RW 0x0	Base Address Used with the size register to set the address window size and location within the range of 4 GB space. An address driven by one of the Ethernet MACs is considered as window hit if (address | size) == (base | size).

Table 559: Size Register 0-3
Offset: Size Register 0 0xF204, Size Register 1 0xF20C, Size Register 2 0xF214, Size Register 3 0xF21C

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RES 0x0	Reserved.
31:16	Size	RW 0x0	Window Size Used with the size register to set the address window size and location within the range of 4 GB space. Must be programed from LSB to MSB as sequence of 1's followed by sequence of 0's. The number of 1's specifies the size of the window in 64 KB granularity (e.g. a value of 0x00ff specifies 256x64k = 16 MB). An address driven by one of the Ethernet MACs is considered as window hit if (address | size) == (base | size).

Table 560: High Address Remap 0-1[1]
Offset: High Address Remap 0 0xF240, High Address Remap 1 0xF244

Bits	Field	Type/ Init Val	Function
31:0	Remap	RW 0x0	Remap Address Specifies address bits[63:32] to be driven to the target interface. Relevant only for target interfaces that supports more than 4 GB address space (e.g. PCI bus).

1. High Address Remap 0 corresponds to Based Address register 0, High Address Remap 1 corresponds to Base Address register 1.

Table 561: Base Address Enable
Offset: 0xF250

Bits	Field	Type/ Init Val	Function
3:0	En	RW 0xF	Address Window Enable Bit per window. If set to '0', the corresponding address window is enabled.

Table 561: Base Address Enable
Offset: 0xF250

Bits	Field	Type/ Init Val	Function
31:4	Reserved	RES 0x0	Reserved.

Table 562: MPSC0/1 Access Protect
Offset: MPSC0 0xF254, MPSC1 0xF258

Bits	Field	Type/ Init Val	Function
1:0	Win0	RW 0x3	Window0 Access Control 0x0 = No access allowed 0x1 = Read Only 0x2 = Reserved 0x3 = Full access (read or write) In case of an access violation (e.g. write data to a read only region): • An interrupt is set. • The transaction is not driven to the target interface.
3:2	Win1	RW 0x3	Window1 Access Control
5:4	Win2	RW 0x3	Window2 Access Control
7:6	Win3	RW 0x3	Window3 Access Control
31:8	Reserved	RES 0x0	Reserved.

Table 563: Internal Space Base Address
Offset: 0xF25C

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RES 0x0	Reserved.
31:16	Base	RW Specified at reset by DevAD[5]. 0x1400 or 0xF100	Base address.

Note

The TWSI serial ROM initialization logic is implemented as part of the.MV64360/1/2 Cunit. This logic is sharing the Cunit four address windows. However, this logic is using a fifth window defined by Internal

Space Base Address register, for accesses to the MV64360/1/2 internal registers. The MPSCs SDMAs do not have access to the MV64360/1/2 internal registers.

H.3 Cunit Control Registers

Table 564: Cunit Arbiter Control
Offset: 0xF300

Bits	Field	Type/ Init Val	Function
9:0	Reserved	RES 0x0	Reserved.
10	GPPInt	RW 0x0	Selects either GPP edge or level interrupts. 0 = Edge trigger interrupt (default setting) 1 = Level sensitive interrupt
15:11	Reserved	RES 0x0	Reserved.
31:16	Reserved	RES 0x11FF	Reserved.

Table 565: Cunit Configuration (CUACR)
Offset: 0xB40C

Bits	Field	Type/ Init Val	Function
30:0	Reserved	RES 0x0	Reserved.
31	DescEnd	RW 0x1	Descriptors Endianess Byte swap during descriptor fetch and close. 0 = Byte swap 1 = No byte swap

Table 566: Cunit Crossbar Time out
Offset: 0xF304

Bits	Field	Type/ Init Val	Function
7:0	Timeout	RW 0xFF	CrossBar Arbiter Timeout Preset Value
15:8	Reserved	RES 0x0	Reserved.
16	TimeoutEn	RW 0x1	CrossBar Arbiter Timer Enable 0 = Enable 1 = Disable
31:17	Reserved	RES 0x0	Reserved.

H.4 Cunit Error Report Registers

Table 567: Cunit Interrupt Cause
Offset: 0xF310

NOTE: All cause bits are "Clear Only". They are set to '1' upon the interrupt event and cleared when the software writes a value of '0'. Writing '1' has no affect.

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RES 0x0	Reserved.
16	S0AddrMiss	RW 0x0	MPSC 0 Address Miss Failed address decoding.
17	S0AccProt	RW 0x0	MPSC 0 Access Protect Violation
18	S0WrProt	RW 0x0	MPSC 0 Write Protect Violation
23:19	Reserved	RES 0x0	Reserved.
24	S1AddrMiss	RW 0x0	MPSC 1 Address Miss Failed address decoding.
25	S1AccProt	RW 0x0	MPSC 1 Access Protect Violation
26	S1WrProt	RW 0x0	MPSC 1 Write Protect Violation
31:27	Reserved	RES 0x0	Reserved.

Table 568: Cunit Interrupt Mask
Offset: 0xF314

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RES 0x0	Reserved.
16	S0AddrMiss	RW 0x0	S0AddrMiss 0 = Disabled 1 = Enabled
17	S0AccProt	RW 0x0	S0AccProt Interrupt 0 = Disabled 1 = Enabled
18	S0WrProt	RW 0x0	S0WrProt Interrupt 0 = Disabled 1 = Enabled
23:19	Reserved	RES 0x0	Reserved.
24	S1AddrMiss	RW 0x0	S1AddrMiss Interrupt 0 = Disabled 1 = Enabled
25	S1AccProt	RW 0x0	S1AccProt Interrupt 0 = Disabled 1 = Enabled
26	S1WrProt	RW 0x0	S1WrProt Interrupt 0 = Disabled 1 = Enabled
31:27	Sel	RW 0x0	Specifies the error event currently being reported in the Error Address register: 0x10 = S0AddrMiss 0x11 = S0AccProt 0x12 = S0WrProt 0x18 = S1AddrMiss 0x19 = S1AccProt 0x1a = S1WrProt All other values are reserved.

Table 569: Cunit Error Address
Offset: 0xF318

Bits	Field	Type/ Init Val	Function
31:0	ErrAddr	RW 0x0	Bits[31:0] of Error Address Latched upon any of the interrupt events. No new address can be latched (due to additional error condition) until the register is being read.

H.5 MPSCs Clocks Routing Registers

Table 570: MPSC Routing (MRR)
Offset: 0xB400

Bits	Field	Type/ Init Val	Function
2:0	MR0	RW 0x7	MPSC0 Routing 0x0 = Connected 0x1 - 0x7 = Unconnected
5:3	Reserved	RES 0x7	Must be 0.
8:6	MR1	RW 0x7	MPSC1 Routing 0x0 = Connected 0x1 - 0x7 = Unconnected
31:9	Reserved	RES 0x0	Must be 0.

Table 571: Rx Clock Routing (RCR)
Offset: 0xB404

Bits	Field	Type/ Init Val	Function
3:0	CRR0	RW 0x0	MPSC0 Rx Clock Routing 0x0 = BRG0 0x1 = BRG1 0x2 - 0x7 = Reserved 0x8 = SCLK0 0x9 - 0xF = Reserved
7:4	Reserved	RES 0x0	Reserved.
11:8	CRR1	RW 0x0	MPSC1 Rx Clock Routing 0x0 = BRG0 0x1 = BRG1 0x2 - 0x7 = Reserved 0x8 = SCLK1 0x9 - 0xF = Reserved
31:12	Reserved	RES 0x0	Reserved.

Table 572: Tx Clock Routing (TCR)
Offset: 0xB408

Bits	Field	Type/ Init Val	Function
3:0	CRT0	RW 0x0	MPSC0 Tx Clock Routing 0x0 = BRG0 0x1 = BRG1 0x2 - 0x7 = Reserved 0x8 = SCLK0 0x9 = TSCLK0 0xa - 0xF = Reserved
7:4	Reserved	RES 0x0	Reserved.
11:8	CRT1	RW 0x0	MPSC1 Tx Clock Routing 0x0 = BRG0 0x1 = BRG1 0x2 - 0x7 = Reserved 0x8 = SCLK1 0x9 = TSCLK1 0xA - 0xF = Reserved
31:12	Reserved	RES 0x0	Reserved.

H.6 MPSC Cause and Mask Registers

Table 573: MPSC Cause
Offset: MPSC0 0xB804, MPSC1 0xB80C

Bits	Field	Type/ Init Val	Description
0	Mpsc0Rx	RO 0x0	MPSC0 Normal Rx Interrupt Summary Logical OR of (unmasked) bits 4-7 below. This bit is read only.
1	Mpsc0RxErr	RO 0x0	MPSC0 Rx Error Interrupt Summary Logical OR of (unmasked) bits 8-11 below. This bit is read only.
2	Reserved	RES 0x0	Reserved.
3	Mpsc0TxErr	RO 0x0	MPSC0 Tx Error Interrupt Summary Logical OR of (unmasked) bits 13-15 below. This bit is read only.
4	Mpsc0RLSC	RW 0x0	MPSC0 Rx Line Status Change (from to IDLE)

Table 573: MPSC Cause (Continued)
Offset: MPSC0 0xB804, MPSC1 0xB80C

Bits	Field	Type/ Init Val	Description
5	Mpsc0RHNT	RW 0x0	MPSC0 Rx Entered HUNT State
6	Mpsc0RFSC/ Mpsc0RCC	RW 0x0	MPSC0 Rx Flag Status Change (HDLC mode) MPSC0 Received Control Character (Bisync, UART modes)
7	Mpsc0RCSC	RW 0x0	MPSC0 Rx Carrier Sense Change (DPLL decoded carriers sense)
8	Mpsc0ROVR	RW 0x0	MPSC0 Rx Overrun
9	Mpsc0RCDL	RW 0x0	MPSC0 Rx Carrier Detect Loss
10	Mpsc0RCKG	RW 0x0	MPSC0 Rx Clock Glitch
11	MPsc0BPER	RW 0x0	MPSC0 Bisync Protocol Error (valid only in Bisync mode)
12	Mpsc0TEIDL	RW 0x0	MPSC0 Tx Entered IDLE State
13	Mpsc0TUDR	RW 0x0	MPSC0 Tx Underrun
14	Mpsc0TCTSL	RW 0x0	MPSC0 Tx Clear To Send Loss
15	Mpsc0TCKG	RW 0x0	MPSC0 Tx Clock Glitch
31:16	Reserved	RES 0x0	Reserved.

Table 574: MPSC Mask
Offset: MPSC0 0xB884, MPSC1 0xB88C

Bits	Field	Type/ Init Val	Description
0	Mpsc0Rx	RW 0x0	Mpsc0Rx Interrupt 0 = Disabled 1 = Enabled
1	Mpsc0RxErr	RW 0x0	Mpsc0RxErr interrupt 0 = Disabled 1 = Enabled
2	Reserved	RES 0x0	Reserved

Table 574: MPSC Mask (Continued)
Offset: MPSC0 0xB884, MPSC1 0xB88C

Bits	Field	Type/ Init Val	Description
3	Mpsc0TxErr	RW 0x0	Mpsc0TxErr Interrupt 0 = Disabled 1 = Enabled
4	Mpsc0Rx	RW 0x0	Mpsc0RLSC Interrupt 0 = Disabled 1 = Enabled
5	Mpsc0RHNT	RW 0x0	Mpsc0RHNT Interrupt 0 = Disabled 1 = Enabled
6	Mpsc0RFSC	RW 0x0	Mpsc0RFSC Interrupt 0 = Disabled 1 = Enabled
7	Mpsc0RCSC	RW 0x0	Mpsc0RCSC Interrupt 0 = Disabled 1 = Enabled
8	Mpsc0ROVR	RW 0x0	Mpsc0ROVR Interrupt 0 = Disabled 1 = Enabled
9	Mpsc0RCDL	RW 0x0	Mpsc0RCDL Interrupt 0 = Disabled 1 = Enabled
10	Mpsc0RCKG	RW 0x0	Mpsc0RCKG Interrupt 0 = Disabled 1 = Enabled
11	MPsc0BPER	RW 0x0	MPsc0BPER interrupt 0 = Disabled 1 = Enabled
12	Mpsc0TEIDL	RW 0x0	Mpsc0TEIDL Interrupt 0 = Disabled 1 = Enabled
13	Mpsc0TUDR	RW 0x0	Mpsc0TUDR Interrupt 0 = Disabled 1 = Enabled
14	Mpsc0TCTSL	RW 0x0	Mpsc0TCTSL Interrupt 0 = Disabled 1 = Enabled
15	Mpsc0TCKG	RW 0x0	Mpsc0TCKG Interrupt 0 = Disabled 1 = Enabled

Table 574: **MPSC Mask** (Continued)
Offset: MPSC0 0xB884, MPSC1 0xB88C

Bits	Field	Type/ Init Val	Description
31:16	Reserved	RES 0x0	Reserved

H.7 MPSCx Main Configuration Registers

Unless otherwise specified:

- ‘1’ means set
- ‘0’ means not set
- ‘0’ is the default value after reset.

Table 575: MPSCx Main Configuration (Low) (MMCRLx)
Offset: MPSC0 0x8000, MPSC1 0x9000

Bits	Field	Type/ Init Val	Function
2:0	MODE	RW 0x000	Mode 000 = HDLC 001 = Reserved 010 = Reserved 011 = Reserved 100 = UART 101 = BISYNC 110 = Reserved 111 = Reserved
3	TTX	RW 0x0	Transparent Transmitter 0 = Normal Mode 1 = Transparent Mode **NOTE:** Transparent Mode overrides the program mode in MODE bits.
4	TRX	RW 0x0	Transparent Receiver 0 = Normal Mode 1 = Transparent Mode **NOTE:** Transparent Mode overrides the program mode in MODE bits.
5	Reserved	RES 0x0	Reserved.
6	ET	RW 0x0	Enable Transmit 0 = Disabled The Tx channel is in Low Power Mode. 1 = Enable The Tx controller is ready for data. When the SDMA has data to transmit, it loads the data to the Tx controller that transmits the data in the selected protocol.

Table 575: MPSCx Main Configuration (Low) (MMCRLx) (Continued)
Offset: MPSC0 0x8000, MPSC1 0x9000

Bits	Field	Type/ Init Val	Function
7	ER	RW 0x0	Enable Receive 0 = Disabled The Rx channel is in Low Power Mode. 1 = Enable The Rx controller is ready to receive data.
9:8	LPBK	RW 0x00	Loop Back (for diagnostic) mode 00 = Normal Operation, no loop back (Default) 01 = Loopback 10 = Echo 11 = Loop Back + Echo In loop back mode, which is only for diagnostic purposes, the transmitted data on TxD is also fed into RxD. In this mode, the same clock source should be used for both Rx and Tx. Echo mode re-transmits received data on RxD (with one clock delay) on TxD. If CD# is asserted, the receiver also receives the incoming data.
10	NLM	RW 0x0	Null Modem 0 = Normal operation The MPSC uses the CD# and CTS# inputs to control the data flow 1 = Null Modem The MPSC CD# and RTS# internal signals are always asserted. The external pin status can still be read from the Event Register. For information about the behavior of the Event Register in different modes, see: • Section 16.6.1 “HDLC Receive/Transmit Operation” on page 256. • Section 16.7.5.6 “CHR10 - BISYNC Event Status Register (ES)” on page 274. • Section 16.8.5.7 “CHR10 - UART Event Status Register (ESR)” on page 284. • Section 16.9.2.3 “CHR10 - Transparent Event Status Register (ESR)” on page 289.
11	Reserved	RES 0x0	Reserved.
12	TSYN	RW 0x0	Transmitter Synchronize to Receiver Setting this bit synchronizes the transmitter to receiver byte boundaries. This is particularly important in the x.21 protocol. 0 = No synchronization assumed. 1 = Transmit bit stream is synchronized to the receive bit stream. **This bit only affects a transparent transmitter.** The transmitter starts transmission nx8 bit period after the receive data arrives. If CTS# is already asserted, the transparent transmitter starts transmitting eight clocks after the receiver starts to receive data. **NOTE:** Only enable this bit when transmit and receive clocks are equal and TCDV and RCDV are set to ‘00’.

Table 575: MPSCx Main Configuration (Low) (MMCRLx) (Continued)
Offset: MPSC0 0x8000, MPSC1 0x9000

Bits	Field	Type/ Init Val	Function
13	Reserved	RES 0x0	Reserved.
15:14	TSNS	RW 0x0	Transmit Sense. Defines the number of bit times the internal sense signal will stay active after last transition on the RXD line occurs. It is useful for AppleTalk protocol to avoid the spurious CD# change interrupt that would otherwise occur during the frame synchronization sequence that precedes the opening flag. The delay is a function of RCDV (clock divider) setting. 00 (RCDV = 0) = Infinite (Carrier Sense is always active - default) 00 (RCDV ≠ 0) = Infinite (Carrier Sense is always active - default) 01 (RCDV = 0) = 14 bit times 01 (RCDV ≠ 0) = 6.5 bit times 10 (RCDV = 0) = 4 bit times (normal AppleTalk) 10 (RCDV ≠0) = 2.5 bit times (normal AppleTalk) 11 (RCDV = 0) = 3 bit times 11 (RCDV ≠ 0) = 1 bit time
16	TIDL	RW 0x0	Transmit Idles 0 = TxD is encoded during data transmission (including preamble and flags/sync patterns). (Default.) TxD is in MARK during idle. 1 = TxD is encoded all the time, even when idles are transmitted, see Table 70 on page 255.
17	RTSM	RW 0x0	RTS# Mode This bit may be changed on the fly. 0 = Send IDLE between frames. RTS# negated between frames and the IDLE pattern is defined by the protocol and TIDL bit. 1 = Send flags/syncs between frames according to the protocol. RTS# is always asserted. Refer to Table 70 on page 255.
18	Reserved	RES 0x0	Reserved.
19	CTSS	RW 0x0	CTS# Sampling Mode 0 = Asynchronous CTS# (Default) CTS# is synchronized inside the MV64360/1/2. Transmission starts after synchronization is achieved with a few cycles delay to the external CTS#. 1 = Synchronous CTS# CTS# is synchronized to the Rx clock. Synchronous CTS# must be used for ISDN D channels.
20	CDS	RW 0x0	CD# Sampling mode 0 = Asynchronous CD# (Default) CD# is synchronized internally in the MV64360/1/2 and then data is received. 1 = Synchronous CD#. CD# is synchronized to the Rx clock.

Table 575: MPSCx Main Configuration (Low) (MMCRLx) (Continued)
Offset: MPSC0 0x8000, MPSC1 0x9000

Bits	Field	Type/ Init Val	Function
21	CTSM	RW 0x0	CTS# Operating Mode 0 = Normal mode (Envelop Mode) CTS# must envelop the frame. De-assertion of CTS# during transmission causes a CTS lost error. 1 = Pulse Mode Once CTS# is sampled low, synchronization has been achieved. Further transitions of CTS# have no effect. CTS# synchronization is lost when RTS# is de-asserted.
22	CDM	RW 0x0	CD# Operating Mode 0 = Normal mode (Envelop Mode) CD# must envelop the frame. De-assertion of CD# during reception causes a CD lost error. 1 = Pulse Mode Once CD# is sampled low, synchronization has been achieved. Further transitions of CD# have no effect.
25:23	CRCM	RW 0x010	CRC Mode 000 = CRC16-CCITT (HDLC based protocols, e.g. X.25) (Default) 001 = CRC-16 (BISYNC) 010 = CRC32-CCITT (HDLC based protocols, e.g. LAP-D. Identical to the Ethernet CRC) 011 = Reserved 1XX = Reserved
27:26	Reserved	RES 0x0	Reserved.
28	TRVD	RW 0x0	Transmit Reverse Data 0 = Normal Mode 1 = Reverse Data Mode The MSB is shifted out first.
29	RRVD	RW 0x0	Receive Reverse Data 0 = Normal Mode 1 = Reverse Data Mode The MSB is shifted in first.
30	Reserved	RES 0x0	Reserved.
31	GDE	RW 0x0	Glitch Detect Enable 0 = Normal mode. No glitch detect. 1 = When glitch is detect, a maskable interrupt is generated. When this bit is set, the MPSC looks for glitches in the external receive and transmit clocks. The MV64360/1/2 tries to clean the input clocks by receiving them via a Schmitt trigger input buffer.

Table 576: MPSCx Main Configuration (High) (MMCRHx)
Offset: MPSC0 0x8004, MPSC1 0x9004

Bits	Field	Type/ Init Val	Function
0	TCI	RW 0x0	Transmit Clock Invert 0 = Normal operation (Default.) Data is shifted out on the falling edge. 1 =The internal transmit clock is inverted by the MPSC before it is used. This allows the MPSC to clock data out half a cycle earlier on the rising edge of the clock.
1	TINV	RW 0x0	Transmit Bit Stream Inversion 0 = No invert 1 = Invert the data before it is sent to the DPLL Setting TINV to ‘1’ generates FM1 from FM0, NRZI mark from NRZI space, etc. It also inverts the bit stream in NRZ mode.
4:2	TPL	RW 0x0	Transmit Preamble Length Determines the number of preamble bytes the transmitter sends before it starts to transmit data. The send pattern is defined by the TPPT bits. 000 = No Preamble (Default) 001 = 1 byte 010 = 2 bytes 011 = 4 bytes 100 = 6 bytes 101 = 8 bytes 110 = 16 bytes 111 = Reserved
8:5	TPPT	RW 0x0	Transmit Preamble Pattern Defines a character sent as a preamble sequence. Two TPPT characters form a preamble byte. The number of preamble bytes sent is defined by the TPL field. The receiving DPLL uses the preamble pattern to lock on the receiving signal.
10:9	TCDV	RW 0x0	Transmit Clock Divider Defines the transmit clock divider. The transmit bit rate is the rate of the clock entering the MPSC Tx machine (from external pin or a BRG) divided by the TCDV field. For FM0, FM1, Manchester, and Differential Manchester, one of the 8x, 16x, or 32x options must be set. 00 = 1x clock mode (Default. For NRZ and NRZI only.) 01 = 8x clock mode 10 = 16x clock mode 11 = 32x clock mode

Table 576: MPSCx Main Configuration (High) (MMCRHx) (Continued)
Offset: MPSC0 0x8004, MPSC1 0x9004

Bits	Field	Type/ Init Val	Function
13:11	TDEC	RW 0x0	Transmit Encoder Specifies the encoding method for the dedicated Tx channel DPLL. 000 = NRZ (default) 001 = NRZI (mark, can be set to Space by setting TINV bit) 010 = FM0 (can be set to FM1 by setting the TINV bit) 011 = Reserved 100 = Manchester 101 = Reserved 110 = Differential Manchester 111 = Reserved
15:14	Reserved	RES 0x0	Reserved.
16	RINV	RW 0x0	Receive Bit Stream Inversion. 0 = No invert 1 = Inverts the data before it is sent from the DPLL to the MPSC data path. Setting RINV to '1' decodes FM1 and NRZI mark when the RENC field is programed to FM0 and NRZI space etc. It also inverts the received bit stream in NRZ mode.
20:17	GDW	RW 0x0	Clock Glitch Width When the GDE bit is set, the MPSC considers Tx/Rx clock pulses that are narrower than GDW system clocks as a glitch.
21	Reserved	RES 0x0	Reserved.
22	RDW	RW 0x0	Receive Data Width 0 = Normal mode The MPSC data path is 16-bits wide. Upon receiving 16-bits, the data is transferred into the SDMA FIFOs. The buffers must be 64-bit word aligned and DMA bursts enabled. **NOTE:** Normal Mode must be used for HDLC based protocols. 1 = Low latency operation Data is transferred to the FIFOs after 8-bits are received. Logical FIFO width is one byte. **NOTE:** This mode allows byte aligned buffers and must be chosen for BISYNC and UART modes. DMA bursts are disabled. The SDMA writes one byte per DRAM access. Setting RDW also bypasses the receive FIFO threshold. The SDMA arbitrates for DMA access as soon as the FIFO has one byte in it.
24:23	RSYL	RW 0x0	Receive Sync Length (BISYNC and Transparent Modes) 00 = External sync (CD# assertion) 01 = 4-bit sync 10 = 8-bit sync (MonoSYNC) 11 = 16-bit sync (BISYNC)

Table 576: MPSCx Main Configuration (High) (MMCRHx) (Continued)
Offset: MPSC0 0x8004, MPSC1 0x9004

Bits	Field	Type/ Init Val	Function
26:25	RCDV	RW 0x0	Receive Clock Divider Defines the receive clock divider. The receive bit rate is the rate of the clock entering the MPSC Rx machine (from external pin or a BRG) divided by the RCDV field. For FM0, FM1, Manchester, and Differential Manchester, one of the 8x, 16x, or 32x options must be set. 00 = 1x clock mode (Default. For NRZ and NRZI only.) 01 = 8x clock mode 10 = 16x clock mode 11 = 32x clock mode
29:27	RENC	RW 0x0	Receive Encoder Specifies the encoding method for the dedicated Rx channel DPLL. 000 = NRZ (default) 001 = NRZI (Mark, can be set to Space by setting RINV bit) 010 = FM0 (can be set to FM1 by setting the RINV bit) 011 = Reserved 100 = Manchester 101 = Reserved 110 = Differential Manchester 111 = Reserved
31:30	SEDG	RW 0x0	Synchronization Clock Edge The clock edge used by the DPLL for adjusting the receive sample point due to drift in the receive signal. 00 = Both rising and falling edges. (Default.) 01 = Rising edge 10 = Falling edge 11 = No adjustment

H.8 SDMA Registers

Table 577: SDMA Configuration Register (SDCx)
Offset: Channel0 0x4000, Channel1 0x6000

Bits	Field	Type/ Init Val	Function
0	RFT	RW 0x0	Receive FIFO Threshold 0 = 8 bytes 1 = Half FIFO (128 bytes) **NOTE:** When working with an 8-bit data path, the threshold is always one byte regardless of the RFT value. It is recommended that RFT bit be set to '0' in this case. When RFT is set to '0', the SDMA will not burst. It will transfer one word (64 bits) on each transfer.

Table 577: SDMA Configuration Register (SDCx) (Continued)
Offset: Channel0 0x4000, Channel1 0x6000

Bits	Field	Type/ Init Val	Function
1	SFM	RW 0x0	Single Frame Mode 0 = Multi frame mode The MV64360/1/2 reads as many frames as needed into the FIFO to keep the transmit FIFO full. The FIFO can handle more than one frame at a time. 1 = Single frame mode The first descriptor is not fetched before the current frame's last descriptor is closed. **NOTE:** The SFM bit must be set to '1' for HDLC Collision mode, and BISYNC protocols. It is also recommended for UART. When the SFM bit is set to '0', CTS Lost cannot be reported in the correct descriptor/frame. In LAN HDLC mode SFM must be set for proper operation.
5:2	RC	RW 0xF	Retransmit Count In collision modes (LAP-D), after executing a backoff procedure RC times, the Tx SDMA closes the buffer with a Retransmit Limit (RL) error, a maskable interrupt is generated, and the SDMA goes to the OFF state. A new Transmit Demand command must be issued to start a new transmission process. When RC field is 0000, the MV64360/1/2 tries to retransmit forever. The CPU needs to issue an abort command in order to stop the retransmit process.
6	BLMR	RW 0x1	Rx Big Little/Endian Receive Mode The MV64360/1/2 supports big or little endian configuration per channel for maximum system flexibility. The BLMR bit only affects data movements. 0 = Big endian convention 1 = Little endian convention
7	BLMT	RW 0x1	Tx Big/Little Endian Transmit Mode The MV64360/1/2 supports big or little endian configuration per channel for maximum system flexibility. The BLMT bit only affects data movements. 0 = Big endian convention 1 = Little endian convention
8	POVR	RW 0x0	PCI Override When set, causes the SDMA to direct all its accesses in PCI_0 direction, overriding normal address decoding process.
9	RIFB	RW 0x0	Receive Interrupt on Frame Boundaries When set, the SDMA Rx generates interrupts only on frame boundaries (i.e. after writing the frame status to the descriptor).
11:10	Reserved	RES 0x0	Reserved.

Table 577: SDMA Configuration Register (SDCx) (Continued)
Offset: Channel0 0x4000, Channel1 0x6000

Bits	Field	Type/ Init Val	Function
13:12	BSZ	0x0	Burst Size Sets the maximum burst size for SDMA transactions: 00 = Burst is limited to 1 64bit words 01 = Burst is limited to 2 64bit words 10 = Burst is limited to 4 64bit words 11 = Burst is limited to 8 64bit words
31:14	Reserved	RES 0x0	Reserved.

Table 578: SDMA Command (SDCMx)
Offset: Channel0 0x4008, Channel1 0x6008

Bits	Field	Type/ Init Val	Function
6:0	Reserved	RES 0x0	Reserved.
7	ERD	RW 0x0	Enable Rx DMA When set to ‘1’, the Rx SDMA will fetch the first descriptor and will be ready for a receive frame. The MV64360/1/2 clears ERD when the MV64360/1/2 receive SDMA has a resource error or when the CPU issues an abort command.
14:8	Reserved	RES 0x0	Reserved.
15	AR	RW 0x0	Abort Receive The CPU sets the AR bit when it needs to abort a receive SDMA channel operation. When the AR bit is set, the SDMA aborts its operation and goes to IDLE state. No descriptor is closed. The MV64360/1/2 clears both the AR and ERD bits when entering IDLE state. The CPU must poll bit 15. When it is ‘0’, the MV64360/1/2 has completed the abort sequence. After an abort the CPU should write the first descriptor address and then set ERD bit to ‘1’.
16	STD	RW 0x0	Stop Tx The SDMA stops transmission at the end of frame (i.e. at the end of buffer with L bit set to ‘1’). After transmitting the last buffer, the transmit SDMA goes to IDLE state. The MV64360/1/2 clears the TxD bit when entering IDLE state. After the SDMA stops, the CPU must write the first descriptor address and than set the TxD bit to ‘1’. The MV64360/1/2 signals the CPU with interrupt when the stop procedure is accomplished.
22:17	Reserved	RES 0x0	Reserved.

Table 578: SDMA Command (SDCMx) (Continued)
Offset: Channel0 0x4008, Channel1 0x6008

Bits	Field	Type/ Init Val	Function
23	TxD	RW 0x0	Tx Demand When this bit is set to '1', the Tx DMA will fetch the first descriptor and will start the transmission process. The MV64360/1/2 clears TxD when it successfully ends an SDMA transmit process. It also clears TxD when a resource error occurs, when the transmit process is halted due to channel error (i.e. CTS# lost), or when the CPU issues an abort command.
30:24	Reserved	RES 0x0	Reserved.
31	AT	0x0	Abort Transmit The CPU sets the TRD bit to '1' when it needs to abort a transmit SDMA channel operation. When the TRD bit is set, the SDMA aborts its operation and goes to IDLE state. No descriptor is closed. The MV64360/1/2 clears both the TRD and TxD bits when entering IDLE state. The CPU must poll bit 31. When it is '0', the MV64360/1/2 has completed the abort sequence. After an abort, the CPU must write the first descriptor address and than set TxD bit to '1'.

H.9 SDMA Cause and Mask Registers

Table 579: SDMA Cause
Offset: 0xB800

Bits	Field	Type/ Init Val	Function
0	Sdma0RxBuf	RW 0x0	SDMA Channel 0 Rx Buffer Return Indicates that SDMA0 Rx closed a descriptor and returned the associated buffer to CPU ownership.
1	Sdma0RxErr	RW 0x0	SDMA Channel 0 Rx Error Indicates that a Rx resource error occurred.
2	Sdma0TxBuf	RW 0x0	SDMA Channel 0 Tx Buffer Return Indicates that SDMA0 Tx closed a descriptor and returned the associated buffer to CPU ownership.
3	Sdma0TxEnd	RW 0x0	SDMA Channel 0 Tx End Indicates that a Tx resource error occurred or that the Tx DMA moved to IDLE after a stop command. Also set when Tx retransmit limit is reached in HDLC mode.
7:4	Reserved	RES 0x0	Reserved.

Table 579: SDMA Cause (Continued)
Offset: 0xB800

Bits	Field	Type/ Init Val	Function
8	Sdma1RxBuf	RW 0x0	SDMA Channel 1 Rx Buffer Return Indicates that SDMA1 Rx closed a descriptor and returned the associated buffer to CPU ownership.
9	Sdma1RxErr	RW 0x0	SDMA Channel 1 Rx Error Indicates that a Rx resource error occurred.
10	Sdma1TxBuf	RW 0x0	SDMA Channel 1 Tx Buffer Return Indicates that SDMA1 Tx closed a descriptor and returned the associated buffer to CPU ownership.
11	Sdma1TxEnd	RW 0x0	SDMA Channel 1 Tx End Indicates that a Tx resource error occurred or that the Tx DMA moved to IDLE after a stop command. Also, set when Tx retransmit limit is reached in HDLC mode.
31:12	Reserved	RES 0x0	Reserved.

Table 580: SDMA Mask
Offset: 0xB880

Bits	Field	Type/ Init Val	Function
0	Sdma0RxBuf	0x0	Sdma0RxBuf Interrupt 0 = Disabled 1 = Enabled
1	Sdma0RxErr	0x0	Sdma0RxErr Interrupt 0 = Disabled 1 = Enabled
2	Sdma0TxBuf	0x0	Sdma0TxBuf Interrupt 0 = Disabled 1 = Enabled
3	Sdma0TxEnd	0x0	Sdma0TxEnd Interrupt 0 = Disabled 1 = Enabled
7:4	Reserved	RES 0x0	Reserved.
8	Sdma1RxBuf	0x0	Sdma1RxBuf Interrupt 0 = Disabled 1 = Enabled
9	Sdma1RxErr	0x0	Sdma1RxErr Interrupt 0 = Disabled 1 = Enabled

**Table 580: SDMA Mask (Continued)
Offset: 0xB880**

Bits	Field	Type/ Init Val	Function
10	Sdma1TxBuf	0x0	Sdma1TxBuf Interrupt 0 = Disabled 1 = Enabled
11	Sdma1TxEnd	0x0	Sdma1TxEnd Interrupt 0 = Disabled 1 = Enabled
31:12	Reserved	RES 0x0	Reserved.

Appendix I. Baud Rate Generators (BRG) Registers

I.1 BRG Register Maps

Table 581: BRG Configuration and Baud Timing Registers Map

Register Name	Offsets	Page
BRGx Configuration Register (BCR)	BRG0 0xB200, BRG1 0xB208	Table 583, p.658
BRGx Baud Tuning register (BTR)	BRG0 0xB204, BRG1 0xB20C	Table 584, p.659

Table 582: BRG Interrupt Registers Map

Register Name	Offset	Page
BRG Cause Register	0xB834	Table 585, p.659
BRG Mask Register	0xB8B4	Table 586, p.660

I.2 BRG Registers

Table 583: BRGx Configuration Register (BCR)
Offset: BRG0 0xB200, BRG1 0xB208

Bits	Field	Type/ Init Val	Function
15:0	CDV	RW 0x0	Count Down Value The user programs the CDV field to define the baud rate that the BRG generates. CDV is loaded into the BRG counter every time it reaches 0. The actual baud rate is: $\text{BaudRate} = \frac{\text{InputClockRate}}{(\text{CDV}+1)\ \text{¥}\ 2}$ When CDV is 0x0000, the generated baud rate is equal to the input clock rate.
16	En	RW 0x0	Enable BRG 0 = Disabled (Output clock is clamped to 0.) 1 = Enabled.
17	RST	RW 0x0	Reset BRG 0 = No Op. 1 = Reset BRG counter to 0.

**Table 583: BRGx Configuration Register (BCR) (Continued)
Offset: BRG0 0xB200, BRG1 0xB208**

Bits	Field	Type/ Init Val	Function
21:18	CLKS	RW 0x2	Clock Source (input clock to the BRG) 0x0 = BclkIn 0x1 = Reserved 0x2 = SCLK0 (from MPSC0 port) 0x3 = TSCLK0 (from MPSC0 port) 0x4,0x5 = Reserved 0x6 = SCLK1 (from MPSC11 port) 0x7 = TSCLK1 (from MPSC1 port) 0x8 – 0xF = SysClk
24:22	Reserved	RES 0x1	Reserved.
25	BTEn	RW 0x0	Baud Tunning Enabled 0 = Disabled 1 = Enabled
31:26	Reserved	RES 0x0	Reserved.

**Table 584: BRGx Baud Tuning register (BTR)
Offset: BRG0 0xB204, BRG1 0xB20C**

NOTE: If the BRG is written for a clock source that is inactive, this register cannot be accessed

Bits	Field	Type/ Init Val	Function
15:0	CUV	RO 0x0	Count Up Value
31:16	Reserved	RES 0x0	Reserved.

**Table 585: BRG Cause Register
Offset: 0xB834**

Bits	Field	Type/ Init Val	Function
0	BTR0	RW 0x0	Baud Tuning 0 interrupt
1	BTR1	RW 0x0	Baud Tuning 1 interrupt
31:2	Reserved	RES 0x1	Reserved.

Table 586: **BRG Mask Register**
Offset: 0xB8B4

Bits	Field	Type/ Init Val	Function
0	BTR0	0x0	BTR0 Interrupt 0 = Disabled 1 = Enabled
1	BTR1	0x0	BTR1 Interrupt 0 = Disabled 1 = Enabled
31:2	Reserved	RES 0x0	Reserved.

Appendix J. IDMA Controller Interface Registers

J.1 IDMA Register Maps

Note

There is aliasing of IDMA registers offsets. Address bits [12:10, 8] are not decoded. This means for example, that offset 0x900 is aliased to offset 0x800 (Channel0 Byte Count register).

Table 587: IDMA Descriptor Register Map

Register	Offsets	Page
Channel DMA Byte Count	Channel 0 0x800, Channel 1 0x804, Channel 2 0x808, Channel 3 0x80C	Table 591, p.662
Channel DMA Source Address	Channel 0 0x810, Channel 1 0x814, Channel 2 0x818, Channel 3 0x81C	Table 592, p.663
Channel DMA Destination Address	Channel 0 0x820, Channel 1 0x824, Channel 2 0x828, Channel 3 0x82C	Table 593, p.663
Channel Next Descriptor Pointer	Channel 0 0x830, Channel 1 0x834, Channel 2 0x838, Channel 3 0x83C	Table 594, p.663
Channel Current Descriptor Pointer	Channel 0 0x870, Channel 1 0x874, Channel 2 0x878, Channel 3 0x87C	Table 595, p.663

Table 588: IDMA Address Decoding Register Map

Register	Offset(s)	Page
Base Address Register x	BAR0 0xA00, BAR1 0xA08, BAR2 0xA10, BAR3 0xA18, BAR4 0xA20, BAR5 0xA28, BAR6 0xA30, BAR7 0xA38	Table 596, p.664
Size Register x	SR0 0xA04, SR1 0xA0C, SR2 0xA14, SR3 0xA1C, SR4 0xA24, SR5 0xA2C, SR6 0xA34, SR7 0xA3C	Table 597, p.666
High Address Remap x	Register 0 0xA60, Register 1 0xA64, Register 2 0xA68, Register 3 0xA6C	Table 598, p.666
Base Address Enable	0xA80	Table 599, p.667
Channelx Access Protect	Channel 0 0xA70, Channel 1 0xA74, Channel 2 0xA78, Channel 3 0xA7C	Table 600, p.667

Table 588: IDMA Address Decoding Register Map (Continued)

Register	Offset(s)	Page
Headers Retarget Control	0xA84	Table 601, p.668
Header Retarget Base	0xa88	Table 602, p.668

Table 589: IDMA Control Register Map

Register	Offset	Page
Channel Control (Low)	Channel 0 0x840, Channel 1 0x844, Channel 2 0x848, Channel 3 0x84C	Table 603, p.669
Channel Control (High)	Channel 0 0x880, Channel 1 0x884, Channel 2 0x888, Channel 3 0x88C	Table 604, p.671
Arbiter Control	0x860	Table 605, p.672
Crossbar Timeout	0x8D0	Table 606, p.673

Table 590: IDMA Interrupt Register Map

Register	Offset	Page
Interrupt Cause	0x8C0	Table 607, p.673
Interrupt Mask	0x8C4	Table 608, p.674
Error Address	0x8C8	Table 609, p.675
Error Select	0x8CC	Table 610, p.676

J.2 IDMA Descriptor Registers

Table 591: Channel DMA Byte Count[1]
Offset: Channel 0 0x800, Channel 1 0x804, Channel 2 0x808, Channel 3 0x80C

Bits	Field	Type/ Init Val	Function
23:0	ByteCnt	RW 0x0	Number of bytes left for the DMA to transfer. When running in 64K descriptor mode, the byte count is 16-bit only (bits[15:0]).
29:24	Reserved	RES 0x0	Reserved.

Table 591: Channel DMA Byte Count[1] (Continued)
Offset: Channel 0 0x800, Channel 1 0x804, Channel 2 0x808, Channel 3 0x80C

Bits	Field	Type/ Init Val	Function
30	BCLeft	RW 0x0	Left Byte Count When running in 16M descriptor mode and when closing a descriptor, indicates whether the whole byte count was completely transferred. 0 = The whole byte count transferred. 1 = Transfer terminated before the whole byte count was transferred.
31	Own	RW 0x0	Ownership Bit When running in 16M descriptor mode, this bit indicates whether the descriptor is owned by the DMA engine (0) or the CPU (1). 0 = DMA engine owned. 1 = CPU owned.

1. When running in 64K descriptor mode and when closing the descriptor, the DMA writes to bits[31:16] the left byte count to be transferred.

Table 592: Channel DMA Source Address
Offset: Channel 0 0x810, Channel 1 0x814, Channel 2 0x818, Channel 3 0x81C

Bits	Field	Type/ Init Val	Function
31:0	SrcAdd	RW 0x0	Bits[31:0] of the DMA source address.

Table 593: Channel DMA Destination Address
Offset: Channel 0 0x820, Channel 1 0x824, Channel 2 0x828, Channel 3 0x82C

Bits	Field	Type/ Init Val	Function
31:0	DestAdd	RW 0x0	Bits[31:0] of the DMA destination address.

Table 594: Channel Next Descriptor Pointer
Offset: Channel 0 0x830, Channel 1 0x834, Channel 2 0x838, Channel 3 0x83C

Bits	Field	Type/ Init Val	Function
31:0	NextDescPtr	RW 0x0	Bits[31:0] of the DMA next descriptor address. The address must be 32-byte aligned (bits[3:0] must be 0x0).

Table 595: Channel Current Descriptor Pointer
Offset: Channel 0 0x870, Channel 1 0x874, Channel 2 0x878, Channel 3 0x87C

Bits	Field	Type/ Init Val	Function
31:0	CDPTR0/1/2/3	RW 0x0	Bits[31:0] of the address from which the current descriptor was fetched.

J.3 IDMA Address Decoding Registers

Table 596: Base Address Register x
Offset: BAR0 0xA00, BAR1 0xA08, BAR2 0xA10, BAR3 0xA18, BAR4 0xA20, BAR5 0xA28, BAR6 0xA30, BAR7 0xA38

Bits	Field	Type/ Init Val	Function
3:0	Target	RW 0x0	Specifies the target interface associated with this window: 0x0 = DRAM 0x1 = Device 0x2 = CPU bus (60x bus only) or integrated SRAM 0x3 = PCI_0 0x4 = PCI_1 0x5 – 0xF = Reserved **NOTE:** 0x2 and 0x4 are only valid for the MV64360 and MV64361. Reserved in the MV64362 device.
7:4	Reserved	RES 0x0	Reserved.

Table 596: Base Address Register x (Continued)
Offset: BAR0 0xA00, BAR1 0xA08, BAR2 0xA10, BAR3 0xA18, BAR4 0xA20, BAR5 0xA28, BAR6 0xA30, BAR7 0xA38

Bits	Field	Type/ Init Val	Function
15:8	Attr	RW 0x0	Specifies target specific attributes depending on the target interface. **DRAM Target Interface** Bits[11:8] select the DRAM bank: 0xe = CS[0]# 0xd = CS[1]# 0xb = CS[2]# 0x7 = CS[3]# **NOTE:** All other bits [11:8] values are Reserved. Bits[13:12] specify the cache coherency support: 0x0 = No cache coherency 0x1 = WT cache coherency 0x2 = WB cache coherency 0x3 = Reserved **NOTE:** Bits[15:14] - Reserved **Device Bus Target Interface** Bits[12:8] select the Device bank: 0x1e = DevCS[0]# 0x1d = DevCS[1]# 0x1b = DevCS[2]# 0x17 = DevCS[3]# 0x0f = BootCS# **NOTE:** All other bits[12:8] values are Reserved. Bits[15:13] - Reserved **PCI0/1 Target Interface** (PCI_1 only applies to the MV64360 and MV64361.) Bits[9:8] specifies the Data swap type: 0x0 = Byte swap 0x1 = No swap 0x2 = Both byte and Word swap 0x3 = Word swap Bit[10] determines the PCI-X No Snoop (NS) attribute: 0x0 = NS attribute is not asserted 0x1 = NS attribute is asserted Bit[11] selects PCI I/O or memory space: 0x0 = I/O 0x1 = Memory Bit[12] selects the PCI REQ64# control: (Only applies to the MV64360 and MV64362.) 0x0 = Forces the PCI master to always assert REQ64#. 0x1 = Asserts REQ64# according to the requested data size. Bit[15:13] - Reserved If the target interface is integrated SRAM or 60x bus: (Integrated SRAM only applies to the MV64360 and MV64361.) Bits[10:8,15:12] - Reserved Bit[11] - Integrated SRAM/60x bus select: 0x0 = Integrated SRAM 0x1 = 60x bus

Table 596: Base Address Register x (Continued)
Offset: BAR0 0xA00, BAR1 0xA08, BAR2 0xA10, BAR3 0xA18, BAR4 0xA20, BAR5 0xA28, BAR6 0xA30, BAR7 0xA38

Bits	Field	Type/ Init Val	Function
31:16	Base	RW 0x0	Base Address Used with the size register to set the address window size and location within the range of 4 GB space. An address driven by one of the Ethernet MACs is considered as window hit if: (address | size) == (base | size).

Table 597: Size Register x
Offset: SR0 0xA04, SR1 0xA0C, SR2 0xA14, SR3 0xA1C, SR4 0xA24, SR5 0xA2C, SR6 0xA34, SR7 0xA3C

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RO 0x0	Read only.
31:16	Size	RW 0x0	Window Size Used with the size register to set the address window size and location within the range of 4 GB space. Must be programed from LSB to MSB as sequence of 1's followed by sequence of 0's. The number of 1's specifies the size of the window in 64 KB granularity (e.g. a value of 0x00ff specifies 256x64k = 16 MB). An address driven by one of the Ethernet MACs is considered as window hit if: (address | size) == (base | size).

Table 598: High Address Remap x[1]
Offset: Register 0 0xA60, Register 1 0xA64, Register 2 0xA68, Register 3 0xA6C

Bits	Field	Type/ Init Val	Function
31:0	Remap	RW 0x0	Remap Address Specifies address bits[63:32] to be driven to the target interface. Only relevant for target interfaces that supports more than 4 GB address space (e.g. PCI bus).

1. Remap 0 corresponds to Base Address register 0, Remap 1 to Base Address register 1, Remap 2 to Base Address register 2 and Remap 3 to Base Address register 3.

Table 599: Base Address Enable
Offset: 0xA80

Bits	Field	Type/ Init Val	Function
7:0	En	RW 0xFF	Address Window Enable Bit per window. If set to '0', the corresponding address window is enabled.
31:8	Reserved	RO 0x0	Read only.

Table 600: Channelx Access Protect
Offset: Channel 0 0xA70, Channel 1 0xA74, Channel 2 0xA78, Channel 3 0xA7C

Bits	Field	Type/ Init Val	Function
1:0	Win0	RW 0x3	Window0 Access control: 0x0 = No access allowed 0x1 = Read Only 0x2 = Reserved 0x3 = Full access (read or write) In case of access violation (e.g. write data to a read only region), an interrupt is set, and the transaction is not driven to the target interface
3:2	Win1	RW 0x3	Window1 access control.
5:4	Win2	RW 0x3	Window2 access control.
7:6	Win3	RW 0x3	Window3 access control.
9:8	Win4	RW 0x3	Window4 access control.
11:10	Win5	RW 0x3	Window5 access control.
13:12	Win6	RW 0x3	Window6 access control.
15:14	Win7	RW 0x3	Window7 access control.
31:16	Reserved	RO 0x0	Read only.

Table 601: Headers Retarget Control
Offset: 0xA84

Bits	Field	Type/ Init Val	Function
0	En	RW 0x0	Headers retarget enable bit 0x0 = Disable 0x1 = Enable
3:1	BSize	RW 0x0	Buffer Size 0x0 = 256 bytes 0x1 = 512 bytes 0x2 = 1 KB 0x3 = 2 KB 0x4 = 4 KB 0x5 = 8 KB 0x6 – 0x7= Reserved
15:4	Reserved	RO 0x0	Read only
31:16	Mask1	RW 0x0	Defines the total space of the buffers to be manipulated, in 64 KB granularity. Size must be set from LSB to MSB as a sequence of 1's, followed by sequence of 0's. For example, in order to retarget the headers of 1K buffers of 1 KB size, which means 1 MB of buffers space, Mask1 should be set to 0x000f **NOTE:** The total address space of retargeted headers must not exceed integrated SRAM size (256 KB). The minimum buffers space to be manipulated is 64 KB

Table 602: Header Retarget Base
Offset: 0xa88

Bits	Field	Type/ Init Val	Function
15:0	Reserved	RO 0x0	Read only.
31:16	Base	RW 0x0	Base address Retarget is executed if address matches base.

J.4 IDMA Channel Control Registers

Table 603: Channel Control (Low)
Offset: Channel 0 0x840, Channel 1 0x844, Channel 2 0x848, Channel 3 0x84C

Bits	Field	Type/ Init Val	Function
2:0	DstBurstLimit	RW 0x0	000 = 8 Bytes 001 = 16 Bytes 010 = Reserved 011 = 32 Bytes 100 = 128 Bytes 101 = Reserved 110 = Reserved 111 = 64 Bytes
3	SrcHold	RW 0x0	Source Hold 0 = Increment source address. 1 = Hold in the same value.
4	DMAAck_Width	RW 0x0	0 = Asserted for one SysClk cycle 1 = Asserted for two SysClk cycles
5	DestHold	RW 0x0	Destination Hold 0 = Increment destination address. 1 = Hold in the same value.
8:6	SrcBurstLimit	RW 0x0	Burst Limit in Each DMA Access 000 = 8 Bytes 001 = 16 Bytes 010 = reserved 011 = 32 Bytes 100 = 128 Bytes 101 = Reserved 110 = Reserved 111 = 64 Bytes
9	ChainMode	RW 0x0	Chained Mode 0 = Chained mode 1 = Non-Chained mode
10	IntMode	RW 0x0	Interrupt Mode 0 = Interrupt asserted every time the DMA byte count reaches ‘0’. 1 = Interrupt asserted when the Next Descriptor pointer is NULL and the DMA byte count reaches ‘0’. **NOTE:** IntMode is only relevant in chain mode.
11	DemandMode	RW 0x0	Demand Mode Enable 0 = Demand mode 1 = Block mode
12	ChanEn	RW 0x0	Channel Enable 0 = The channel is suspended. 1 = The channel is activated. Re-setting the bit to ‘1’, allows the channel to continue the DMA transfer.

Table 603: Channel Control (Low) (Continued)
Offset: Channel 0 0x840, Channel 1 0x844, Channel 2 0x848, Channel 3 0x84C

Bits	Field	Type/ Init Val	Function
13	FetchND	RWC 0x0	Fetch Next Descriptor If set to '1', forces a fetch of the next descriptor. Cleared after the fetch is completed. **NOTE:** FetchND is only relevant in chain mode
14	ChanAct	RO 0x0	DMA Channel Active Read only. 0 = Channel is not active. 1 = Channel is active.
15	DMAReqDir	RW 0x0	DMAReq Direction 0 = DMAReq# generated by the source. 1 = DMAReq# generated by the destination.
16	DMAReqMode	RW 0x0	DMAReq# Mode 0 = DMAReq# is level input. 1 = DMAReq# is edge triggered input.
17	CDEn	RW 0x0	Close Descriptor Enable If enabled, the DMA writes the upper byte(s) of the byte count field back to memory. In 64K descriptor mode, it writes the remainder byte count into bits[31:16] of the byte count field. In n16M descriptor mode, it writes the ownership and status bits into bits[31:24] of byte count field. 0 = Disable 1 = Enable **NOTE:** Enable in chain mode only. Disable when a new chain is begun by directly programming the first descriptor of the chain into the channel registers instead of fetching the descriptor from memory using the FetchND bit [13].
18	EOTEn	RW 0x0	End Of Transfer Enable If enabled, a DMA transfer can be stopped in the middle of the transfer using EOT signal. 0 = Disable 1 = Enable
19	EOTMode	RW 0x0	End of Transfer Affect 0 = Fetch next descriptor 1 = Channel halt
20	Abr	RW 0x0	Channel Abort When the software sets this bit to '1', the DMA aborts in the middle. The bit is cleared by the DMA hardware.
22:21	SAddrOvr	RW 0x0	Override Source Address 00 = No override. 01 = Source interface and attributes are taken from BAR 1 10 = Source interface and attributes are taken from BAR 2 11 = Source interface and attributes are taken from BAR 3

Table 603: Channel Control (Low) (Continued)
Offset: Channel 0 0x840, Channel 1 0x844, Channel 2 0x848, Channel 3 0x84C

Bits	Field	Type/ Init Val	Function
24:23	DAddrOvr	RW 0x0	Override Destination Address 00 = No override. 01 = Destination interface and attributes are taken from BAR 1 10 = Destination interface and attributes are taken from BAR 2 11 = Destination interface and attributes are taken from BAR 3
26:25	NAddrOvr	RW 0x0	Override Next Descriptor Address 00 = No override. 01 = Next descriptor interface and attributes are taken from BAR 1 10 = Next descriptor interface and attributes are taken from BAR 2 11 = Next descriptor interface and attributes are taken from BAR 3
27	DMAAckMode	RW 0x0	DMA Acknowledge Mode 0 = Asserted for one SysClk when the DMA engine issues the transaction. 1 = Asserted only with the actual transaction driven on the device bus (same timing as CSTiming signal).
28	TimerReq	RW 0x0	Timer DMA Request Enable 0 = DMA requests taken from the DMAReq# pin. 1 = DMA requests taken from the timer/counter.
30:29	DMAAckDir	RW 0x0	DMA Acknowledge Direction 00 = Reserved. 01 = Asserted with accesses to destination. 10 = Asserted with accesses to source. 11 = Reserved. **NOTE:** Must match the DMAReqDir setting.
31	DescMode	RW 0x0	Descriptor Mode 0 = 64K descriptor mode 1 = 16M descriptor mode

Table 604: Channel Control (High)
Offset: Channel 0 0x880, Channel 1 0x884, Channel 2 0x888, Channel 3 0x88C

Bits	Field	Type/ Init Val	Function
0	Endianess	RW 0x0	Big/Little Endian Data 0 = Big 1 = Little
1	DescBS	RW 0x0	Descriptors Byte Swap If enabled, DMA swap the bytes of 64-bit dword during descriptor fetch and close. 0 = Enable 1 = Disable
7:2	Reserved	RES 0x0	Reserved

Table 604: Channel Control (High)
Offset: Channel 0 0x880, Channel 1 0x884, Channel 2 0x888, Channel 3 0x88C

Bits	Field	Type/ Init Val	Function
31:8	Reserved	RO 0x0	Read Only

Table 605: Arbiter Control
Offset: 0x860

Bits	Field	Type/ Init Val	Function
1:0	Arb0	RW 0x0	Slice 0 of “pizza arbiter”. 00 = Channel0 01 = Channel1 10 = Channel2 11 = Channel3
3:2	Arb1	RW 0x1	Slice 1 of “pizza arbiter”.
5:4	Arb2	RW 0x2	Slice 2 of “pizza arbiter”.
7:6	Arb3	RW 0x3	Slice 3 of “pizza arbiter”.
9:8	Arb4	RW 0x0	Slice 4 of “pizza arbiter”.
11:10	Arb5	RW 0x1	Slice 5 of “pizza arbiter”.
13:12	Arb6	RW 0x2	Slice 6 of “pizza arbiter”.
15:14	Arb7	RW 0x3	Slice 7 of “pizza arbiter”.
17:16	Arb8	RW 0x0	Slice 8 of “pizza arbiter”.
19:18	Arb9	RW 0x1	Slice 9 of “pizza arbiter”.
21:20	Arb10	RW 0x2	Slice 10 of “pizza arbiter”.
23:22	Arb11	RW 0x3	Slice 11 of “pizza arbiter”.
25:24	Arb12	RW 0x0	Slice 12 of “pizza arbiter”.
27:26	Arb13	RW 0x1	Slice 13 of “pizza arbiter”.

Table 605: Arbiter Control (Continued)
Offset: 0x860

Bits	Field	Type/ Init Val	Function
29:28	Arb14	RW 0x2	Slice 14 of “pizza arbiter”.
31:30	Arb15	RW 0x3	Slice 15 of “pizza arbiter”.

Table 606: Crossbar Timeout
Offset: 0x8D0

Bits	Field	Type/ Init Val	Function
7:0	Timeout	RW 0xFF	CrossBar Arbiter Timeout Preset Value
15:8	Reserved	RO 0x0	Read only.
16	TimeoutEn	RW 0x1	CrossBar Arbiter Timer Enable 0 = Enable 1 = Disable
31:17	Reserved	RO 0x0	Read only.

J.5 IDMA Interrupt Registers

Table 607: Interrupt Cause
Offset: 0x8C0[1]

Bits	Field	Type/ Init Val	Function
0	Comp	RW 0x0	Channel0 DMA Completion.
1	AddrMiss	RW 0x0	Channel0 Address Miss Failed address decoding.
2	AccProt	RW 0x0	Channel0 Access Protect Violation
3	WrProt	RW 0x0	Channel0 Write Protect
4	Own	RW 0x0	Channel0 Descriptor Ownership Violation Attempt to access the descriptor owned by the CPU.

**Table 607: Interrupt Cause
Offset: 0x8C0[1] (Continued)**

Bits	Field	Type/ Init Val	Function
7:5	Reserved	RES 0x0	Reserved.
12:8	Various	RW 0x0	Same as channel0 cause bits.
15:13	Reserved	RES 0x0	Reserved
20:16	Various	RW 0x0	Same as channel0 cause bits.
23:21	Reserved	RES 0x0	Reserved.
28:24	Various	RW 0x0	Same as channel0 cause bits.
31:29	Reserved	RES 0x0	Reserved.

1. All cause bits are clear only. They are set to ‘1’ upon an interrupt event and cleared when the software writes a value of ‘0’. Writing ‘1’ has no affect.

**Table 608: Interrupt Mask
Offset: 0x8C4**

Bits	Field	Type/ Init Val	Function
0	Comp	RW 0x0	Comp Interrupt 0 = Disable 1 = Enable
1	AddrMiss	RW 0x0	AddrMiss Interrupt 0 = Disable 1 = Enable
2	AccProt	RW 0x0	AccProt Interrupt 0 = Disable 1 = Enable
3	WrProt	RW 0x0	WrProt Interrupt 0 = Disable 1 = Enable
4	Own	RW 0x0	Own Interrupt 0 = Disable 1 = Enable
7:5	Reserved	RES 0x0	Reserved.

Table 608: Interrupt Mask
Offset: 0x8C4 (Continued)

Bits	Field	Type/ Init Val	Function
12:8	Various	RW 0x0	Same as channel0 mask bits.
15:13	Reserved	RES 0x0	Reserved.
20:16	Various	RW 0x0	Same as channel0 mask bits.
23:21	Reserved	RES 0x0	Reserved.
28:24	Various	RW 0x0	Same as channel0 mask bits.
31:29	Reserved	RES 0x0	Reserved.

Table 609: Error Address
Offset: 0x8C8

Bits	Field	Type/ Init Val	Function
31:0	ErrAddr	RW 0x0	Bits[31:0] of Error Address Latched upon any of the error events interrupts (address miss, access protection, write protection, ownership violation). Once the address is latched, no new address is latched until the register is read. **NOTE:** If the erroneous transaction was retargeted, the address reported in this field is the retargeted address, not the original address.

Table 610: Error Select
Offset: 0x8CC

Bits	Field	Type/ Init Val	Function
4:0	Sel	RW 0x0	Specifies the error event currently reported in the Error Address register: 0x0 = Reserved 0x1 = AddrMiss0 0x2 = AccProt0 0x3 = WrProt0 0x4 = Own0 0x5 - 0x7 = Reserved 0x8 = Reserved 0x9 = AddrMiss1 0xA = AccProt1 0xB = WrProt1 0xC = Own1 0xD - 0xF = Reserved 0x10 = Reserved 0x11 = AddrMiss2 0x12 = AccProt2 0x13 = WrProt2 0x14 = Own2 0x15 - 0x17 = Reserved 0x18 = Reserved 0x19 = AddrMiss3 0x1A = AccProt3 0x1B = WrProt3 0x1C = Own3 0x1D – 0x1F = Reserved Read Only.
31:5	Reserved	RO 0x0	Read only.

Appendix K. Timer/Counters Registers

K.1 Timers/Counters Register Map

Note

There is aliasing of Timers/Counters registers offsets. Address bits [12:10, 8] are not decoded. This means for example, that offset 0x950 is aliased to offset 0x850 (Timer/Counter0 register).

Table 611: IDMA Descriptor Register Map

Register	Offset	Page
Timer/Counter 0	0x850	Table 612, p.677
Timer/Counter 1	0x854	Table 613, p.677
Timer/Counter 2	0x858	Table 614, p.678
Timer/Counter 3	0x85C	Table 615, p.678
Timer/Counter Control	0x864	Table 616, p.678
Timer/Counter Interrupt Cause	0x868	Table 617, p.680
Timer/Counter Interrupt Mask	0x86C	Table 618, p.681

K.2 Timer/Counters Registers

Table 612: Timer/Counter 0
Offset: 0x850

Bits	Field	Type/ Init Val	Function
31:0	TC0	RW 0x0	Timer/Counter 0 Value

Table 613: Timer/Counter 1
Offset: 0x854

Bits	Field	Type/ Init Val	Function
31:0	TC1	RW 0x0	Timer/Counter 1 value.

**Table 614: Timer/Counter 2
Offset: 0x858**

Bits	Field	Type/ Init Val	Function
31:0	TC2	RW 0x0	Timer/Counter 2 value.

**Table 615: Timer/Counter 3
Offset: 0x85C**

Bits	Field	Type/ Init Val	Function
31:0	TC3	RW 0x0	Timer/Counter 3 value.

**Table 616: Timer/Counter Control
Offset: 0x864**

Bits	Field name	Type/ Init Val	Function
0	TC0En	RW 0x0	Timer/Counter Enable 0 = Disable 1 = Enable **NOTE:** When configured to counter, new count starts only with new write of '1' to the `TC0En` bit. In timer mode, the count continues as long as `TC0En` is set to '1'. Counting starts two cycles after `TC0En` assertion.
1	TC0Mode	RW 0x0	Timer/Counter Mode 0 = Counter 1 = Timer
2	TC0Trig	RW 0x0	Timer/Counter Trigger 0 = No external trigger Starts counting as soon as TC0En is set to '1'. 1 = External trigger. Starts counting as soon as TC0En is set to '1' AND the external TC0En input is asserted.
3	TCnt0_Width	RW 0x0	0 = TCTcnt asserted for one SysClk cycle. 1 = TCTcnt asserted for two SysClk cycles.
7:4	Reserved	RES 0x0	Reserved.

Table 616: Timer/Counter Control (Continued)
Offset: 0x864

Bits	Field name	Type/ Init Val	Function
8	TC1En	RW 0x0	Timer/Counter Enable 0 = Disable 1 = Enable **NOTE:** When configured to counter, new count starts only with new write of '1' to the `TC1En` bit. In timer mode, the count continues as long as `TC1En` is set to '1'. Counting starts two cycles after `TC1En` assertion.
9	TC1Mode	RW 0x0	Timer/Counter Mode 0 = Counter 1 = Timer
10	TC1Trig	RW 0x0	Timer/Counter Trigger 0 = No external trigger Starts counting as soon as `TC1En` is set to '1'. 1 = External trigger Starts counting as soon as `TC1En` is set to '1' AND the external `TC1En` input is asserted.
11	TCnt1_Width	RW 0x0	0 = TCTcnt asserted for one SysClk cycle. 1 = TCTcnt asserted for two SysClk cycles.
15:12	Reserved	RES 0x0	Reserved.
16	TC2En	RW 0x0	Timer/Counter Enable 0 = Disable 1 = Enable **NOTE:** When configured to counter, new count starts only with new write of '1' to the `TC2En` bit. In timer mode, the count continues as long as `TC2En` is set to '1'. Counting starts two cycles after `TC2En` assertion.
17	TC2Mode	RW 0x0	Timer/Counter Mode 0 = Counter 1 = Timer
18	TC2Trig	RW 0x0	Timer/Counter Trigger 0 = No external trigger. Starts counting as soon as `TC2En` is set to '1'. 1 = External trigger. Starts counting as soon as `TC2En` is set to '1' AND the external `TC2En` input is asserted.
19	TCnt2_Width	RW 0x0	0 = TCTcnt asserted for one SysClk cycle. 1 = TCTcnt asserted for two SysClk cycles.
23:20	Reserved	RES 0x0	Reserved.

Table 616: Timer/Counter Control (Continued)
Offset: 0x864

Bits	Field name	Type/ Init Val	Function
24	TC3En	RW 0x0	Timer/Counter Enable 0 = Disable 1 = Enable **NOTE:** When configured to counter, new count starts only with new write of '1' to the TcEn bit. In timer mode, the count continues as long as TcEn is set to '1'. Counting starts two cycles after TCEn assertion.
25	TC3Mode	RW 0x0	Timer/Counter Mode 0 = Counter 1 = Timer
26	TC3Trig	RW 0x0	Timer/Counter Trigger 0 = No external trigger Starts counting as soon as TC3En is set to '1'. 1 = External trigger Starts counting as soon as TC3En is set to '1' AND external TC3En input is asserted.
27	TCnt3_Width	RW 0x0	0 = TCTcnt asserted for one SysClk cycle. 1 = TCTcnt asserted for two SysClk cycles.
31:28	Reserved	RES 0x0	Reserved

Table 617: Timer/Counter Interrupt Cause
Offset: 0x868

NOTE: All cause bits are clear only. They are set to '1' upon timer terminal count. They are cleared by writing a value of '0'. Writing a value of '1' has no affect.

Bits	Field	Type/ Init Val	Function
0	TC0	RW 0x0	Timer/Counter 0 terminal count.
1	TC1	RW 0x0	Timer/Counter 1 terminal count.
2	TC2	RW 0x0	Timer/Counter 2 terminal count.
3	TC3	RW 0x0	Timer/Counter 3 terminal count.
30:4	Reserved	RES 0x0	Reserved.
31	Sum	RO 0x0	Summary of all cause bits. Read Only

Table 618: Timer/Counter Interrupt Mask
Offset: 0x86C

Bits	Field	Type/ Init Val	Function
0	TC0	0x0	TC0 Interrupt 0 = Disabled 1 = Enabled
1	TC1	RW 0x0	TC1 Interrupt 0 = Disabled 1 = Enabled
2	TC2	0x0	TC2 Interrupt 0 = Disabled 1 = Enabled
3	TC3	0x0	TC3 Interrupt 0 = Disabled 1 = Enabled
31:4	Reserved	RES 0x0	Reserved.

Appendix L. Watchdog Timer

L.1 Watchdog Registers

Table 619: Watchdog Configuration Register (WDC)
Offset: 0xB410

Bits	Field	Type/ Init Val	Description
23:0	Preset_VAL	RW 0xFF.FFFF	This field holds the 24 most significant bits which the watchdog counter loads each time it is enabled or serviced. After reset, this field is set to 0xFF.FFFF. The preset value is equal to {0xPreset_VAL,FF}.
25:24	CTL1	RW 0x0	A write sequence of ‘01’ followed by ‘10’ into CTL1 disables/enables the watchdog.
27:26	CTL2	RW 0x0	A write sequence of ‘01’ followed by ‘10’ to CTL2 services the watchdog timer.
28	Reserved	RES 0x0	Reserved.
29	WDNMI	RO 0x1	Non-Maskable Interrupt When the watchdog counter reaches a value equal to NMI_VAL, this bit is asserted. This pin can be used to drive the processor’s NMI# pin.
30	WDE	RO 0x1	Watchdog Expiration When the watchdog counter expires, this bit is asserted. The WDE# pin can be used to reset the entire system.
31	EN	RO 0x0	Enable 0 = Watchdog is disabled, counter is loaded with Preset_VAL. WDNMI and WDE are set to ‘1’. 1 = Watchdog is enabled.

Table 620: Watchdog Value Register (WDV)
Offset: 0xB414

Bits	Field	Type/ Init Val	Description
23:0	NMI_VAL	RW 0x0	NMI_VAL are the 24 least significant bits of a 32-bit value. The upper 8 bits are always ‘00’. When the Watchdog counter reaches a value equal to the NMI value WDNMI# pin is asserted. The actual NMI value is a 32-bit number equal to {0x00,NMI_VAL}.
31:24	Reserved	RES 0x0	Reserved.

Appendix M. TWSI Interface Registers

M.1 TWSI Interface Register Map

Table 621: TWSI Interface Register Map

Register	Offset	Page
TWSI Slave Address	0xC000	Table 622, p.683
TWSI Extended Slave Address	0xC010	Table 623, p.683
TWSI Data	0xC004	Table 624, p.684
TWSI Control	0xC008	Table 625, p.684
TWSI Status	0xC00C	Table 626, p.685
TWSI Baud Rate	0xC00C	Table 627, p.685
TWSI Soft Reset	0xC01C	Table 628, p.685

M.2 TWSI Registers

Table 622: TWSI Slave Address
Offset: 0xC000

Bits	Field	Type/ Init Val	Function
0	GCE	RW 0x0	General Call Enable If set to ‘1’, the TWSI slave interface responds to general call accesses.
7:1	SAddr	RW 0x0	Slave address For a 7-bit slave address, bits[7:1] are the slave address. For a 10-bit address, SAddr[7:3] must be set to ‘11110’ and SAddr[2:1] stands for the two MSB (bits[9:8]) of the 10-bit address.
31:8	Reserved	RES 0x0	Reserved.

Table 623: TWSI Extended Slave Address
Offset: 0xC010

Bits	Field	Type/ Init Val	Function
7:0	SAddr	RW 0x0	Bits[7:0] of the 10-bit slave address.

Table 623: **TWSI Extended Slave Address**
Offset: 0xC010

Bits	Field	Type/ Init Val	Function
31:8	Reserved	RES 0x0	Reserved.

Table 624: **TWSI Data**
Offset: 0xC004

Bits	Field	Type/ Init Val	Function
7:0	Data	RW 0x0	Data/Address byte to be transmitted by the TWSI master or slave, or data byte received.
31:8	Reserved	RES 0x0	Reserved.

Table 625: **TWSI Control**
Offset: 0xC008

Bits	Field	Type/ Init Val	Function
1:0	Reserved	RES 0x0	Read only.
2	ACK	RW 0x0	Acknowledge When set to ‘1’, the TWSI master drives the acknowledge bit in response to received read data and to the TWSI slave in response to received address or write data.
3	IFlg	RW 0x0	Interrupt Flag If any of the status codes other than 0xF8 are set, the TWSI hard-ware sets the bit to ‘1’. Cleared by a CPU write of ‘0’.
4	Stop	RW 0x0	Stop When set to ‘1’, the MV64360/1/2 drives a stop condition on the bus. Cleared by the TWSI hardware.
5	Start	RW 0x0	Start When set to ‘1’, the MV64360/1/2 drives a start condition as soon as the bus is free. Cleared by the TWSI hardware.
6	TWSIEn	RW 0x0	If set to ‘0’, the SDA and SCL inputs are not sampled and the TWSI slave interface does not respond to any address on the bus.
7	IntEn	RW 0x0	Interrupt Enable When set to ‘1’, the interrupt is generated each time the interrupt flag is set.

Table 625: TWSI Control
Offset: 0xC008

Bits	Field	Type/ Init Val	Function
31:8	Reserved	RES 0x0	Reserved.

Note

Status and Baud Rate registers share the same offset. When being read, this register functions as Status register. When written, it acts as Baud Rate register.

Table 626: TWSI Status
Offset: 0xC00C

Bits	Field	Type/ Init Val	Function
7:0	Stat	RO 0xF8	TWSI Status See exact status code in the TWSI section. Read only.
31:8	Reserved	RES 0x0	Reserved.

Table 627: TWSI Baud Rate
Offset: 0xC00C

Bits	Field	Type/ Init Val	Function
2:0	N	WO 0x4	See exact frequency calculation in the TWSI section. Write only.
6:3	M	WO 0x4	See exact frequency calculation in the TWSI section. Write only.
31:7	Reserved	RES 0x0	Reserved.

Table 628: TWSI Soft Reset
Offset: 0xC01C

Bits	Field	Type/ Init Val	Function
31:0	Rst	WO 0x0	Write Only Write to this register resets the TWSI logic and sets all TWSI registers to their reset values.

Appendix N. General Purpose Port Interface Registers

N.1 General Purpose Port Register Map

Table 629: GPP Register Map

Register	Offset	Page
GPP I/O Control	0xF100	Table 630, p.686
GPP Level Control	0xF110	Table 631, p.686
GPP Value	0xF104	Table 632, p.687
GPP Interrupt Cause	0xF108	Table 633, p.687
GPP Interrupt Mask0	GPP Interrupt Mask0	Table 634, p.687
GPP Interrupt Mask1	0xF114	Table 635, p.687
GPP Value Set	0xF118	Table 636, p.688
GPP Value Clear	0xF11C	Table 637, p.688

N.2 General Purpose Port Registers

Table 630: GPP I/O Control
Offset: 0xF100

Bits	Field	Type/ Init Val	Function
31:0	GPP I/O	RW 0x0	GPP Input/Output Select, bit per each GPP pin 0 = Input 1 = Output

Table 631: GPP Level Control
Offset: 0xF110

Bits	Field	Type/ Init Val	Function
31:0	GPP Level	RW 0x0	GPP Input Level Select, bit per each GPP pin 0 = Active high 1 = Active low

Table 632: GPP Value
Offset: 0xF104

Bits	Field	Type/ Init Val	Function
31:0	GPP Value	RW 0x0	GPP Pins Values, bit per each GPP pin If the GPP pin is programed as an input pin, it's associated bit is a Read Only bit containing the GPP pin value (or negated value in case of an active low pin). If programed as an output pin, it is a read/write bit. It's programed value is driven on the GPP pin.

Table 633: GPP Interrupt Cause
Offset: 0xF108

Bits	Field	Type/ Init Val	Function
31:0	Cause	RW 0x0	GPP Interrupt Cause Bits Set to '1' upon GPP input pin assertion. Only cleared by the CPU or PCI writing '0'. **NOTE:** If using level sensitive GPP interrupts, interrupt is generated according to the GPP Value register value, rather than this register value, and interrupt clear is performed directly on the originating device.

Table 634: GPP Interrupt Mask0
Offset: 0xF10C

Bits	Field	Type/ Init Val	Function
31:0	Mask	RW 0x0	GPP Interrupts Mask If a bit is set to '1', it's associated GPP interrupt is enabled. **NOTE:** If using edge sensitive GPP interrupts, it masks GPP Interrupt Cause register bits. If using level sensitive GPP interrupts, it masks GPP Value register bits.

Table 635: GPP Interrupt Mask1
Offset: 0xF114

Bits	Field	Type/ Init Val	Function
31:0	Mask	RW 0x0	Same as GPP Interrupt Mask0

Table 636: GPP Value Set
Offset: 0xF118

Bits	Field	Type/ Init Val	Function
31:0	Set	0x0	Write ’1’ to a bit sets the corresponding bit in the GPP Value register. Write an ’0’ has no affect. Read from this register returns ’0’.

Table 637: GPP Value Clear
Offset: 0xF11C

Bits	Field	Type/ Init Val	Function
31:0	Clear	RW 0x0	Write ’1’ to a bit clears the corresponding bit in the GPP Value register. Write an ’0’ has no affect. Read from this register returns ’0’.

Appendix O. Interrupt Controller Interface Registers

O.1 Interrupt Controller Register Map

Table 638: Register Map

Register	Offset	Page
Main Interrupt Cause (Low)	0x004	Table 639, p.689
Main Interrupt Cause (High)	0x00C	Table 640, p.691
CPUInt[0]# Mask (Low)	0x014	Table 641, p.693
CPUInt[0]# Mask (High)	0x01C	Table 642, p.693
CPUInt[0]# Select Cause	0x024	Table 643, p.693
CPUInt[1]# Mask (Low)	0x034	Table 644, p.694
CPUInt[1]# Mask (High)	0x03C	Table 645, p.694
CPUInt[1]# Select Cause	0x044	Table 646, p.694
INT0# Mask (Low)	0x054	Table 647, p.694
INT0# Mask (High)	0x05C	Table 648, p.695
INT0# Select Cause	0x064	Table 649, p.695
IINT1# Mask (Low)	0x074	Table 650, p.695
INT1# Mask (High)	0x07C	Table 651, p.695
INT1# Select Cause	0x084	Table 652, p.695

O.2 Interrupt Controller Registers

Table 639: Main Interrupt Cause (Low)
Offset: 0x004[1]

Bits	Field	Type/ Init Val	Function
0	Reserved	RES 0x0	Reserved.
1	DevErr	RW 0x0	Device bus error (parity, Ready# timer, burst violation)

Table 639: Main Interrupt Cause (Low) (Continued)
Offset: 0x004[1]

Bits	Field	Type/ Init Val	Function
2	DMAErr	RW 0x0	DMA error (address decoding and protection)
3	CPUErr	RW 0x0	CPU error (parity, address decoding and protection)
4	IDMA0	RW 0x0	IDMA Channel0 completion
5	IDMA1	RW 0x0	IDMA Channel1 completion
6	IDMA2	RW 0x0	IDMA Channel2 completion
7	IDMA3	RW 0x0	IDMA Channel3 completion
8	Timer0	RW 0x0	Timer 0
9	Timer1	RW 0x0	Timer 1
10	Timer2	RW 0x0	Timer 2
11	Timer3	RW 0x0	Timer 3
12	PCI0	RW 0x0	PCI0 (summary of PCI0 Cause register)
13	SRAMErr	RW 0x0	SRAM parity error
14	GbEErr	RW 0x0	GbE error (address decoding and protection)
15	CErr	RW 0x0	Serial ports error (address decoding and protection)
16	PCI1	RW 0x0	PCI1 (summary of PCI1 Cause register) **NOTE:** Only applies to the MV64360 and MV64361. Reserved in the MV64362 device.
17	DRAMErr	RW 0x0	DRAM ECC error
18	WDNMI	RW 0x0	WD reached its NMI threshold
19	WDE	RW 0x0	WD reached terminal count

Table 639: Main Interrupt Cause (Low) (Continued)
Offset: 0x004[1]

Bits	Field	Type/ Init Val	Function
20	PCI0In	RW 0x0	PCI_0 Inbound (summary of the PCI_0 Inbound Cause register).
21	PCI0Out	RW 0x0	PCI_0 Outbound (summary of the PCI_0 Outbound Cause register)
22	PCI1In	RW 0x0	PCI_1 Inbound (summary of the PCI_1 Inbound Cause register) **NOTE:** Only applies to the MV64360 and MV64361. Reserved in the MV64362 device.
23	PCI1Out	RW 0x0	PCI_1 Outbound (summary of the PCI_1 Outbound Cause register). **NOTE:** Only applies to the MV64360 and MV64361. Reserved in the MV64362 device.
24	P1_GPP0_7	RW 0x0	CPU1 GPP[7:0] Interrupt
25	P1_GPP8_15	RW 0x0	CPU1 GPP[15:8] Interrupt
26	P1_GPP16_23	RW 0x0	CPU1 GPP[23:16] Interrupt
27	P1_GPP24_31	RW 0x0	CPU1 GPP[31:24] Interrupt
28	P1_CPU_DB	RW 0x0	Summary of CPU1 Doorbell Cause register
31:29	Reserved	RES 0x0	Reserved.

1. All bits are read only. To clear an interrupt, the software must access the Local Interrupt Cause registers.

Table 640: Main Interrupt Cause (High)
Offset: 0x00C

Bits	Field	Type/ Init Val	Function
0	GbE0	RW 0x0	GbE0 summary
1	GbE1	RW 0x0	GbE1 summary **NOTE:** Only valid for the MV64360 and MV64361. Reserved in the MV64362 device.
2	GbE2	RW 0x0	GbE2 summary **NOTE:** Only valid for the MV64360. Reserved in the MV64361 and MV64362 device.
3	Reserved	RES 0x0	Reserved.

Table 640: Main Interrupt Cause (High) (Continued)
Offset: 0x00C

Bits	Field	Type/ Init Val	Function
4	SDMA0	RW 0x0	SDMA0 (summary of MPSC0 SDMA Cause register)
5	TWSI	RW 0x0	TWSI Interrupt
6	SDMA1	RW 0x0	SDMA1 (summary of MPSC1 SDMA Cause register)
7	BRG	RW 0x0	BRG
8	MPSC0	RW 0x0	MPSC0 (summary of MPSC0 Cause register)
9	MPSC1	RW 0x0	MPSC1 (summary of MPSC1 cause register)
10	G0Rx	RW 0x0	GbE0 Rx summary
11	G0Tx	RW 0x0	GbE0 Tx summary
12	G0Misc	RW 0x0	GbE0 misc. summary
13	G1Rx	RW 0x0	GbE1 Rx summary **NOTE:** Only valid for the MV64360 and MV64361. Reserved in the MV64362 device.
14	G1Tx	RW 0x0	GbE1 Tx summary **NOTE:** Only valid for the MV64360 and MV64361. Reserved in the MV64362 device.
15	G1Misc	RW 0x0	GbE1 misc. summary **NOTE:** Only valid for the MV64360 and MV64361. Reserved in the MV64362 device.
16	G2Rx	RW 0x0	GbE2 Rx summary **NOTE:** Only valid for the MV64360. Reserved in the MV64361 and MV64362 device.
17	G2Tx	RW 0x0	GbE2 Tx summary **NOTE:** Only valid for the MV64360. Reserved in the MV64361 and MV64362 device.
18	G2Misc	RW 0x0	GbE2 misc. summary **NOTE:** Only valid for the MV64360. Reserved in the MV64361 and MV64362 device.
23:19	Reserved	RES 0x0	Reserved.

Table 640: Main Interrupt Cause (High) (Continued)
Offset: 0x00C

Bits	Field	Type/ Init Val	Function
24	P0_GPP0_7	RW 0x0	CPU0 GPP[7:0] Interrupt
25	P0_GPP8_15	RW 0x0	CPU0 GPP[15:8] Interrupt
26	P0_GPP16_23	RW 0x0	CPU0 GPP[23:16] Interrupt
27	P0_GPP24_31	RW 0x0	CPU0 GPP[31:24] Interrupt
28	P0_CPU_DB	RW 0x0	Summary of CPU0 Doorbell Cause register
31:29	Reserved	RES 0x0	Reserved.

Table 641: CPUInt[0]# Mask (Low)
Offset: 0x014

Bits	Field	Type/ Init Val	Function
31:0	Mask	RW 0x0	Mask bit per each cause bit. If bit is set to 0, interrupt is masked, if set to ’1’ interrupt is enabled. Mask only affects the assertion of CPUInt[0]# pin. It does not affect the setting of bits in the cause register.

Table 642: CPUInt[0]# Mask (High)
Offset: 0x01C

Bits	Field	Type/ Init Val	Function
31:0	Mask	RW 0x0	Same as CPUInt[0]# Mask Low register.

Table 643: CPUInt[0]# Select Cause
Offset: 0x024[1]

Bits	Field	Type/ Init Val	Function
29:0	Cause	RW 0x0	A shadow register of the Low or High Interrupt Cause registers. If any of the High Interrupt Cause register non-masked interrupts is set, and no non-masked interrupt bit of the Low Interrupt Cause register is set, this register contains a copy of the High Interrupt Cause register. In any other case, it contains a copy of the Low Interrupt Cause register.

Table 643: CPUInt[0]# Select Cause (Continued)
Offset: 0x024[1]

Bits	Field	Type/ Init Val	Function
30	Sel	RW 0x0	Indicates if bits[29:0] are a shadow of the Low or High Cause register bits 0 = Shadow of Low Interrupt Cause register 1 = Shadow of High Interrupt Cause register
31	Stat	RW 0x0	Indicates if there are non masked active interrupts in both Low and High Cause registers 0 = No active non-masked interrupts in both Low and High Interrupt Cause registers. 1 = There are active non-masked interrupts in both Low and High Interrupt Cause registers.

1. Read Only register.

Table 644: CPUInt[1]# Mask (Low)
Offset: 0x034

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CPUInt[0]# Mask register.

Table 645: CPUInt[1]# Mask (High)
Offset: 0x03C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CPUInt[0]# Mask register

Table 646: CPUInt[1]# Select Cause
Offset: 0x044

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CPUInt[0]# Select Cause register

Table 647: INT0# Mask (Low)
Offset: 0x054

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CPUInt[0]# Mask (Low).

Table 648: INT0# Mask (High)
Offset: 0x05C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CPUInt[0]# Mask (High).

Table 649: INT0# Select Cause
Offset: 0x064

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CPUInt[0]# Select Cause.

Table 650: INT1# Mask (Low)
Offset: 0x074

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CPUInt[0]# Mask.

Table 651: INT1# Mask (High)
Offset: 0x07C

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CPUInt[0]# Mask (High).

Table 652: INT1# Select Cause
Offset: 0x084

Bits	Field	Type/ Init Val	Function
31:0	Various	RW 0x0	Same as CPUInt[0]# Select Cause.

Appendix P. Pins Multiplexing Interface Registers

P.1 MPP Registers Map

Table 653: GPP Interface Register Map

Register	Offset	Page
MPP Control0	0xF000	Table 654, p.696
MPP Control1	0xF004	Table 655, p.698
MPP Control2	0xF008	Table 656, p.700
MPP Control3	0xF00C	Table 657, p.701

P.2 MPP Registers

Table 654: MPP Control0
Offset: 0xF000

Bits	Field	Type/ Init Val	Function
3:0	MPPSel0	RW 0x0	MPP0 Select 0x0 = GPP[0] 0x1 = GNT1[0]# 0x2 = TxD0 0x3 = CD1 0x4 = DBurst# 0x5 – 0xE = Reserved 0XF = Debug[0]
7:4	MPPSel1	RW 0x0	MPP1 Select 0x0 = GPP[1] 0x1 = REQ1[0]]# 0x2 = RxD0 0x3 = SCLK1 0x4 = PME1# 0x5 – 0xE = Reserved 0xF = Debug[1]

**Table 654: MPP Control0 (Continued)
Offset: 0xF000**

Bits	Field	Type/ Init Val	Function
11:8	MPPSel2	RW 0x0	MPP2 Select 0x0 = GPP[2] 0x1 = GNT1[1]# 0x2 = RTS0 0x3 = TSCLK1 0x4 = DBurst# 0x5 – 0xE = Reserved 0xF = Debug[2]
15:12	MPPSel3	RW 0x0	MPP3 Select 0x0 = GPP[3] 0x1 = REQ1[1]# 0x2 = CTS0 0x3 = TxD1] 0x4 = InitAct 0x5 – 0xE = Reserved 0xF = Debug[3]
19:16	MPPSel4	RW 0x0	MPP4 Select 0x0 = GPP[4] 0x1 = GNT1[2]# 0x2 = CD0 0x3 = TxD1] 0x4 = DBurst# 0x5 – 0xE = Reserved 0xF = Debug[4]
23:20	MPPSel5	RW 0x0	MPP5 Select 0x0 = GPP[5] 0x1 = REQ1[2]# 0x2 = SCLK0 0x3 = RxD1 0x4 = PME1# 0x5 – 0xE = Reserved 0xF = Debug[5]
27:24	MPPSel6	RW 0x0	MPP6 Select 0x0 = GPP[6] 0x1 = GNT1[3]# 0x2 = TSCLK0 0x3 = RTS1 0x4 = DBurst# 0x5 – 0xE = Reserved 0xF = Debug[6]

**Table 654: MPP Control0 (Continued)
Offset: 0xF000**

Bits	Field	Type/ Init Val	Function
31:28	MPPSel7	RW 0x0	MPP7 Select 0x0 = GPP[7] 0x1 = REQ1[3]# 0x2 = TxD0 0x3 = CTS1 0x4 = InitAct 0x5 – 0xE = Reserved 0xF = Debug[7]

**Table 655: MPP Control1
Offset: 0xF004**

Bits	Field	Type/ Init Val	Function
3:0	MPPSel8	RW 0x0	MPP8 Select 0x0 = GPP[8] 0x1 = GNT1[4]# 0x2 = TxD0 0x3 = CD1 0x4 = DBurst# 0x5 – 0xE = Reserved 0xF = Debug[8]
7:4	MPPSel9	RW 0x0	MPP9 Select 0x0 = GPP[9] 0x1 = REQ1[4]# 0x2 = RxD0 0x3 = SCLK1 0x4 = PME1# 0x5 – 0xE = Reserved 0xF = Debug[9]
11:8	MPPSel10	RW 0x0	MPP10 Select 0x0 = GPP[10] 0x1 = GNT1[5]# 0x2 = RTS0 0x3 = TSCLK1 0x4 = DBurst# 0x5 – 0xE = Reserved 0xF = Debug[10]

Table 655: MPP Control1 (Continued)
Offset: 0xF004

Bits	Field	Type/ Init Val	Function
15:12	MPPSel11	RW 0x0	MPP11 Select 0x0 = GPP[11] 0x1 = REQ1[5]# 0x2 = CTS0 0x3 = RxD1 0x4 = InitAct 0x5 – 0xE = Reserved 0xF = Debug[11]
19:16	MPPSel12	RW 0x0	MPP12 Select 0x0 = GPP[12] 0x1 = GNT1[0]# 0x2 = CD0 0x3 = TxD[1] 0x4 = DBurst# 0x5 – 0xE = Reserved 0xF = Debug[12]
23:20	MPPSel13	RW 0x0	MPP13 Select 0x0 = GPP[13] 0x1 = REQ1[0]# 0x2 = SCLK0 0x3 = RTS1 0x4 = PME1# 0x5 – 0xE = Reserved 0xF = Debug[13]
27:24	MPPSel14	RW 0x0	MPP14 Select 0x0 = GPP[14] 0x1 = GNT1[1]# 0x2 = TSCLK0 0x3 = RxD1 0x4 = DBurst#] 0x5 – 0xE = Reserved 0xF = Debug[14]
31:28	MPPSel15	RW 0x0	MPP15 Select 0x0 = GPP[15] 0x1 = REQ1[1]# 0x2 = RxD0 0x3 = CTS1 0x4 = InitAct 0x5 – 0xE = Reserved 0xF = Debug[15]

**Table 656: MPP Control2
Offset: 0xF008**

Bits	Field	Type/ Init Val	Function
3:0	MPPSel16	RW 0x0	MPP16 Select 0x0 = GPP[16] 0x1 = GNT0[0]# 0x2 = DMAReq[0]# 0x3 = InitAct 0x4 = WDNMI# 0x5 – 0xE = Reserved 0xF = Debug[16]
7:4	MPPSel17	RW 0x0	MPP1 Select 0x0 = GPP[17] 0x1 = REQ0[0]# 0x2 = DMAAck[0]# 0x3 = EOT[3] 0x4 = WDE# 0x5 – 0xE = Reserved 0xF = Debug[17]
11:8	MPPSel18	RW 0x0	MPP18 Select 0x0 = GPP[18] 0x1 = GNT0[1]# 0x2 = DMAReq[1]# 0x3 = PME0# 0x4 = WDNMI# 0x5 – 0xE = Reserved 0xF = Debug[18]
15:12	MPPSel19	RW 0x0	MPP19 Select 0x0 = GPP[19] 0x1 = REQ0[1]# 0x2 = DMAAck[1]# 0x3 = EOT[2] 0x4 = WDE# 0x5 – 0xE = Reserved 0xF = Debug[19]
19:16	MPPSel20	RW 0x0	MPP20 Select 0x0 = GPP[20] 0x1 = GNT0[2]# 0x2 = DMAReq[2]# 0x3 = InitAct 0x4 = BClkIn 0x5 – 0xE = Reserved 0xF = Debug[20]

Table 656: MPP Control2 (Continued)
Offset: 0xF008

Bits	Field	Type/ Init Val	Function
23:20	MPPSel21	RW 0x0	MPP21 Select 0x0 = GPP[21] 0x1 = REQ0[2]# 0x2 = DMAAck[2]# 0x3 = EOT[1] 0x4 = BClkIn 0x5 – 0xE = Reserved 0xF = Debug[21]
27:24	MPPSel22	RW 0x0	MPP22 Select 0x0 = GPP[22] 0x1 = GNT0[3]# 0x2 = DMAReq[3]# 0x3 = PME0# 0x4 = BClkOut 0x5 – 0xE = Reserved 0xF = Debug[22]
31:28	MPPSel23	RW 0x0	MPP23 Select 0x0 = GPP[23] 0x1 = REQ0[3]# 0x2 = DMAAck[3]# 0x3 = EOT[0] 0x4 = BClkOut 0x5 – 0xE = Reserved 0xF = Debug[23]

Table 657: MPP Control3
Offset: 0xF00C

Bits	Field	Type/ Init Val	Function
3:0	MPPSel24	RW 0x0	MPP24 Select 0x0 = GPP[24] 0x1 = GNT0[4]# 0x2 = TCEn[0]# 0x3 = InitAct 0x4 = WDNMI# 0x5 – 0xE = Reserved 0xF = Debug[24]

**Table 657: MPP Control3 (Continued)
Offset: 0xF00C**

Bits	Field	Type/ Init Val	Function
7:4	MPPSel25	RW 0x0	MPP25 Select 0x0 = GPP[25] 0x1 = REQ0[4]# 0x2 = TCTcnt[0]# 0x3 = EOT[3] 0x4 = WDE# 0x5 – 0xE = Reserved 0xF = Debug[25]
11:8	MPPSel26	RW 0x0	MPP26 Select 0x0 = GPP[26] 0x1 = GNT0[5]# 0x2 = TCEn[1]# 0x3 = PME0# 0x4 = WDNMI# 0x5 – 0xE = Reserved 0xF = Debug[26]
15:12	MPPSel27	RW 0x0	MPP27 Select 0x0 = GPP[27] 0x1 = REQ0[5]# 0x2 = TCTcnt[1]# 0x3 = EOT[2] 0x4 = WDE# 0x5 – 0xE = Reserved 0xF = Debug[27]
19:16	MPPSel28	RW 0x0	MPP28 Select 0x0 = GPP[28] 0x1 = GNT0[0]# 0x2 = TCEn[2]# 0x3 = InitAct 0x4 = BClkIn 0x5 – 0xE = Reserved 0xF = Debug[28]
23:20	MPPSel29	RW 0x0	MPP29 Select 0x0 = GPP[29] 0x1 = REQ0[0]# 0x2 = TCTcnt[2]# 0x3 = EOT[1] 0x4 = BClkIn 0x5 – 0xE = Reserved 0xF = Debug[29]

Table 657: MPP Control3 (Continued)
Offset: 0xF00C

Bits	Field	Type/ Init Val	Function
27:24	MPPSel30	RW 0x0	MPP30 Select 0x0 = GPP[30] 0x1 = GNT0[1]# 0x2 = TCEn[3]# 0x3 = PME0# 0x4 = BClkOut 0x5 – 0xE = Reserved 0xF = Debug[30]
31:28	MPPSel31	RW 0x0	MPP31 Select 0x0 = GPP[31] 0x1 = REQ0[1]# 0x2 = TCTcnt[3]# 0x3 = EOT[0] 0x4 = BClkOut 0x5 – 0xE = Reserved 0xF = Debug[31]

Appendix Q. Serial Initialization Interface Registers

Table 658: Serial Init Last Data
Offset: 0xF324

Bits	Field	Type/ Init Val	Function
31:0	DLast	RW 0xFFFFFFFF	Last Serial Data The MV64360/1/2 finishes with serial ROM initialization when it reaches data that equals this register.

Table 659: Serial Init Control
Offset: 0xF328

Bits	Field	Type/ Init Val	Function
0	Reserved	RES 0x0	Reserved.
7:1	ROMAddr	RW Bits [1:0]: Sampled at reset on DevAD[3:2]. Bits [3:2]: 0x0	Serial ROM Address
15:8	OffsetL	RW 0x0	Bits[7:0] of the first byte offset.
23:16	OffsetH	RW 0x0	Bits[15:8] of the first byte offset.
24	OffsetHEn	RW Sampled at reset.	Enable 16-bit Byte Offset 0 - 8-bit offset 1 - 16-bit offset
25	InitEn	RW Sampled at reset on DevAD[0].	Serial Initialization Enable When initialization begins, cleared by the serial ROM initialization logic.
31:26	Reserved	RES 0x0	Reserved.

Table 660: Serial Init Status
Offset: 0xF32c

Bits	Field	Type/ Init Val	Function
4:0	Stat	RW0x1f	Serial Initialization Status If the initialization ends successfully, stat uses offset 0x1f. Any other status implies an initialization failure. Stat bits decoding is the same as TWSI Status register bits[7:3]. Read only.
31:5	Reserved	RES 0x0	Reserved.

Revision History

Table 661: Revision History

Document Type	Revision	Date
Preliminary Datasheet	0.1	October 11, 1999
Preliminary Datasheet	0.8	November 20, 1999
Preliminary Datasheet	0.9	June 25, 2001
Preliminary Datasheet	0.95	December 5, 2001
Preliminary Datasheet	0.96	December 17, 2001
1. Changed the following pinout items in Section 3. "Pin Information" on page 41: • AJ14: VDD I/O • AK14: Pad1[32] • AL14: PCI1_CAL These changes are due to the deletion of PCI1_NCAL. 2. Corrected cross-reference problems found in several interface register maps. 3. Updates made to Section 15. "Gigabit Ethernet Controller" on page 210. 4.		
Preliminary Datasheet	0.98	January 29, 2002
1. Changed the Device interface address space size to 512 MB for each of the five device banks. There is a total space of 2.5 GB. 2. Made the following pinout changes: • Changed RES, ball F06, to AVSS0_A (PLL0 quite VSS) in Table 4, "Clock Pin Assignments," on page 42. • Change to ball out diagram. Ball A3 is COL1 and B3 is COL0. The Test_Mode[0] and Test_Mode[1] functionality is now through the JTAG interface. This also means that COL0 and COL1 are no longer multiplexed on RxD0[4] and RxD1[4], respectively. Descriptions for these pins are now included with the other Ethernet interface pins, Table 11, "Ethernet Port_0 Interface Pin Assignments," on page 56 or Table 12, "Ethernet Port_1 Interface Pin Assignments," on page 58. **NOTE:** If using the third GbE port, COL2 is now muxed on PAD1[39]. It is no longer muxed on RxD2[4], see Table 115, "GbE Port2 Multiplexing," on page 343. 3. In Table 5, "CPU Interface Pin Assignments," on page 43, amended the note for HIT1# and DRdy1# that these pins must also be pulled up if using a single CPU in MPX mode.		

Table 661: Revision History

Document Type	Revision	Date
Preliminary Datasheet (Continued)	0.98	January 29, 2002

4. Corrected the following pinout items in Section 3. “Pin Information” on page 41:
   - E18: FBCLKIN
   - A19: StartBurstIn
   - B19: StartBurst
   - A20: BA[0]
   - G02: DevDP[0]
   - Deleted PCI0_NCAL (AM33) signal. No longer used. AM33 is MPP[16].
   - C25: DQS[4]#
5. Corrected default address range for Integrated SRAM in Table 22, “CPU Default Address Mapping,” on page 84. New range is 0x4200.0000 to 0x4203.FFFF
6.
7. Revised Figure 14: "PowerPC Read Protocol", on page 103, Figure 15: "PowerPC Write Protocol", on page 104, and Figure 16: "PowerPC Pipeline Reads Example", on page 105.
8. Added note to 9.5 “Burst Support” on page 105 stating that the MV64360/1/2 also supports MPC74xx 16 byte burst read/write transactions.
9. Added note to 9.6 “Transactions Flow Control” on page 106 stating that The MV64360/1/2 fastest TA# response to a write transaction is two cycles after AACK* assertion.
10. Revised the balls that hold the two bit ID in multi-GT mode in 9.14.2 “Multi-MV Mode Address Decoding” on page 121. The balls are A[9-10].
11. Revised the procedure in 9.14.3 “Initializing a Multi-MV64360/1/2 System” on page 121.
12. Changed the bit range in the Semaphore0-7 registers in A.5 “SMP Registers” on page 472.
13. Revised Figure 22: "DDR Burst Write Example", on page 131 and Figure 23: "DDR Burst Read Example", on page 131.
14. Changed to maximum clock frequency of the SDRAM controller to 183 Mhz in Section 11. “DDR SDRAM Controller” on page 129, and throughout the data sheet.
15. Added new figures, Figure 24: "Consecutive Reads to the Same Page", on page 138 and Figure 26: "Write to x4 DDR DIMM Example", on page 143.
16. Revised the following sections:
   - 11.11 “DRAM Clocking” on page 144.
   - 11.12 “DRAM Address/Data Drive” on page 144.
   - 11.13 “DRAM Read Data Sample” on page 145.
17. Added minimum setting parameters in 12.2 “Device Timing Parameters” on page 148.

Table 661: Revision History

Document Type	Revision	Date
Preliminary Datasheet (Continued)	0.98	January 29, 2002
18. Revised Figure 28: "Device Write Parameters Example", on page 150, Figure 29: "Pipeline Sync Burst SRAM Read Example", on page 150, Figure 30: "Ready# Extending Acc2First", on page 152, Figure 31: "Ready# Extending Acc2Next", on page 152, and Figure 32: "Ready# Extending WrLow Parameter", on page 153. 19. Revised 13.16.6 “Expansion ROM” on page 185 20. New voltage information in 13.18 “PCI Pads” on page 186. 21. Added in 13.25 “PCI Interface Registers” on page 199 which cases of writes to unmapped PCI offsets of the MV64360/1/2’s internal registers space may result in a destructive write to existing PCI interface registers. 22. Revised Table 378, “PCI Command,” on page 535. New fields include SEndMode bits [27:26] and SEndMode bits[28]. 23. Added Table 69, “MPSC Port Pinout,” on page 252. 24. Added new section 18.4.9 “Buffer Transfer Upon Demand” on page 310. 25. Added note about the integrated clamping diodes in 26.3 “MPP I/O Pads” on page 346. 26. Revised Table 117, “Reset Configuration,” on page 348. Also, added new information about PCI interface signals sampled on PCI reset de-assertion. 27. New PLL and DLL information in 28.1 “PLL and DLL” on page 357. 28. Revised Section 29. “Electrical Specifications” on page 358 and Section 30. “Commercial AC Timing Specifications” on page 364. 29. Added Figure 89: "Part Marking", on page 414.		
Datasheet	A	May 5, 2002
1. Corrected DDR SDRAM data rate to 366 MHz. 2. Added bit type to the registers, see “Preface” on page 28. 3. Changed note for Gnt0/1# signals in Table 6, “PCI Bus 0 Interface Pin Assignments,” on page 47 and Table 7, “PCI Bus 1 Interface Pin Assignments,” on page 50. **NOTE:** When using the internal PCI arbiter, a pull down is required. 4. Added the following connection information for HSTL_VREF0/1 in Table 5, “CPU Interface Pin Assignments,” on page 43: **NOTE:** HSTL_VREF0 [A33] and HSTL_VREF1 [H34] are connected to the same VREF rail. Hence, the balls are connected to the same voltage, 0.75V. If not operating in HSTL mode, these pins must be NC. 5. Changed the description for PCI0_CAL and PCI1_CAL. These signals are connected to VDD through a resistor. 6. Added SDRAM initialization sequence information for 183 Mhz frequency in 11.13 “DRAM Read Data Sample” on page 145. 7. Added details to 9.19 “CPU Interface I/O Signaling” on page 125 that, when connecting the CPU_CAL pin to V_{DD} CPU via a resistor, the resistor size should be in the range of 50 ohms.		

Table 661: Revision History

Document Type	Revision	Date
Datasheet (Continued)	A	May 5, 2002
8. Corrected 11.11 “DRAM Clocking” on page 144. The DRAM clock domain is determined by DevAD [18]. 9. Revised note in 15.9 “Token Rate Configuration” on page 237. 10. Added 27.1 “Reset Pins” on page 347. 11. Corrected description for DevAD[11:10] and DevAD[21:20] in Table 117, “Reset Configuration,” on page 348. 12. Revised electrical characteristics in Section 29. “Electrical Specifications” on page 358. 13. Revised the AC parameters in Section 30. “Commercial AC Timing Specifications” on page 364. 14. Revised Table 235, “Reset Sample (Low),” on page 465 and Table 236, “Reset Sample (High),” on page 468. 15. Revised Table 277, “SRAM Test Mode,” on page 485. Bits [31:0] are reserved and there values must not be changed. 16. Changed description for Table 383, “PCI Arbiter Control,” on page 540 PD[6:0] bits [20:14].		
Revised Datasheet, Preliminary	B	January 13, 2002
1. Included the following revised sections: • Section 31. “Preliminary Industrial AC Timing Specifications (100 MHz)” on page 380 • Section 32. “Preliminary Industrial AC Timing Specifications (125 MHz)” on page 396 2. New information for the new MV64361 and MV64362 devices, see Section 2. “Device Differences” on page 38 for a summary of these new parts. This also includes the following new sections: • Section 5. “MV64361 Pinout Map and Table, 724 Pin BGA” on page 70 • Section 6. “MV64362 Pinout Map and Table, 724 Pin BGA” on page 76 For the new parts there were substantial revisions to: • Section 1. “Overview” on page 32 • Section 3. “Pin Information” on page 41 • Section 13. “PCI Interface” on page 158 • Section 15. “Gigabit Ethernet Controller” on page 210 **NOTE:** These revisions also apply to the applicable registers. 3. Added “Related Documentation” on page 31. 4. Added Section 2. “Device Differences” on page 38 to outline basic device differences between the MV64360, MV64361, and MV64362. This section includes a differences table and figures showing the interfaces for each device.		

Table 661: **Revision History**

Document Type	Revision	Date
Revised Datasheet, Preliminary (Continued)	B	January 13, 2002

5. Added new pin list diagrams in Section 3. "Pin Information" on page 41.
6. Added new Table 2, "Pin Voltage Levels," on page 42.
7. Added the following note to CSTiming# in Table 10, "Device Interface Pin Assignments," on page 55:

NOTE: This pin is in High-Z during reset assertion and for two cycles after reset de-assertion. A pull up may be added to avoid an erroneous qualification of the CS#[3:0] signals.

8. Revised the description for COL0/1 in Table 11, "Ethernet Port_0 Interface Pin Assignments," on page 56 and Table 12, "Ethernet Port_1 Interface Pin Assignments," on page 58. When operating in the GMII mode operating in 100 (RxClk0), these pins are not used and must be pulled down.
9. Changed the note for the JTMS pin in Table 16, "JTAG Pin Assignment," on page 61. Instead of a pull down, a pull-up is required for this pin.
10. Added Table 17, "Unused Interface Strapping," on page 62.
11. Added pull down information for the MDC and MDIO signals in Table 13, "Ethernet Control Interface Pin Assignments," on page 61.
12. Added the following note to the end of 9.5 "Burst Support" on page 105.

NOTE: Burst transactions to internal space are not supported. This type of transaction results in an interrupt assertion (if not masked) and the CPU Error Cause register setting `TTErr` bit [2] to '1'. The transaction completes normally on the CPU bus. However, on writes, the data is discarded. On reads, random data is returned.

13. Revised the note at the end of 9.3 "PowerPC 60x Bus Interface" on page 99 to state that when interfacing a CPU with a 32-bit wide address, A[4] is the MSB and A[35] is the LSB.
14. Added the following notes to 9.13.1 "MV64360/1/2 Internal 60x Bus Arbiter" on page 119:

NOTE: In the MV64360 and MV64361, by default, the BR1# input is masked, enabling CPU0 to boot first. To enable CPU1 arbitration, set the CPU Master Control register's MaskBR1 bit to '0', see Table 110 on page 138.

If the MV64360 and MV64361 are using a single CPU system, only the BR0*, BG0*, and DBG0* signals are used as arbitration signals. The CPU Master Control register's MaskBR1 bit must be set to '1' and the BR1* input requires a pull-up.

15. Added the following note to 9.14 "PowerPC Multi-MV Mode" on page 120:

NOTE: Operating in multi-MV mode affects the AC Timing. Before implementing multi-GT support, consult with your local FAE.

16. Added the following note to 9.14.2 "Multi-MV Mode Address Decoding" on page 121:

NOTE: If the CPU attempts to access an address that is outside of the address ranges defined above (even with an address-only transaction), the system may hang. Before accessing such an address, set the boot CPU Configuration register's `NoMatchCntEn` bit [8] to '1'.

Table 661: Revision History

Document Type	Revision	Date
Revised Datasheet, Preliminary (Continued)	B	January 13, 2002

17. Revised the maximum DDR bandwith down to 23.4 from 25.6 Gbps in Section 11. “DDR SDRAM Controller” on page 129.
18. Changed the description of the RAS precharge (TRP) in Table 36, “SDRAM Timing Parameters,” on page 133.
19. Revised the following figures in Section 12. “Device Controller” on page 147:
    - Figure 27: "Device Read Parameters Example", on page 149
    - Figure 28: "Device Write Parameters Example", on page 150
    - Figure 31: "Ready# Extending Acc2Next", on page 152
    - Figure 32: "Ready# Extending WrLow Parameter", on page 153
20. Added new PCI and CPU to Device bus addressing information to 12.8 “PCI and CPU to Device Bus Addressing” on page 156.
21. Added the following note to 13.19 “PCI-X Reset Configuration” on page 186:

NOTE: According to the PCI spec, DEVSEL#, STOP#, and TRDY# must be pulled up. It expects the "central resource" to actively drive these signals to the desired value during PCI reset, and float them on PCI reset de-assertion.

Also added the following note to the end of the section:

NOTE: In addition to the outlined requirements, it also necessary to strap DevAD[31:29] to the same configuration. For example, for PCI-X/66MHz, drive DEVSEL#, STOP#, and TRDY# to [1,0,1], connect DevAD[31:30] to a pull-up and DevAD[29] to a pull-down.

22. Added the following note to 20.1 “Background” on page 322:

NOTE: If an address parity error is detected during a snoop write back, cache coherency cannot be guaranteed. To enable cache coherency, the address windows defined by the address decode and remap registers in the PCI interface must exactly match the CPU interface address windows. For example, the size of the SCS_0 memory region must be the same in both the CPU and PCI interfaces. In addition, the remap registers in the PCI interface should map the SCS_0 region to exactly the same region defined by the SCS_0 address decode registers in the CPU interface.

23. Added the following note to 14.4.5 “Outbound Free Queue” on page 208.

NOTE: When initializing the Outbound Free Head /Tail Pointer registers, the Head Pointer register must be programmed first. If the Tail Pointer register is programmed first, the Outbound Free Queue Overflow Interrupt `OutFQOvr` is asserted.

24. Revised the description in 18.4.8 “Descriptor Ownership” on page 309.
25. Added notes in 20.2 “Snoop Regions” on page 323 about maintaining cache coherency when working with snoop regions.
26. Added the following note to 20.4 “Implementation” on page 327:

NOTE: For best snoop performance, enable the snoop pipeline by setting the Dunit Control (High) register’s `SnoopPipe` bit [24] to ’1’.

27. Revised 23.2.5 “Baud Rate Register” on page 335.
28. Revised 26.2 “Multi Purpose Pins” on page 343.

Table 661: **Revision History**

Document Type	Revision	Date
Revised Datasheet, Preliminary (Continued)	B	January 13, 2002

29. Added the following note to 27.2 “Pins Sample Configuration” on page 347:
NOTE: If external logic is used instead of pull up and pull down resistors, the logic must drive the AD[31:0] signals to the desired values during Rst* assertion. The external logic must float the bus no later than the third cycle after Rst* de-assertion until normal operation occurs.

30. Deleted the sub-section on restarting initialization in 27.3 “Serial ROM Initialization” on page 353.

31. Added information in 29.3 “DC Electrical Characteristics Over Operating Range” on page 360 that the MPP[31:0] signals share the same DC characteristics as the PCI pins.

32. In Section 34. “MV64360/1/2 Part Numbering” on page 413, added information about stepping A2 of the device and new part numbers for the commercial 150 MHz (HSTL interface) part.

33. Added the following note to PCIReq64 bit [27] in Table 185, “PCI_0 Memory 0 Base Address,” on page 447:
NOTE: When working in PCI-X mode, this bit must be reset.

34. Added the following note to AACK Delay_2 bit [25 in Table 232, “CPU Configuration,” on page 462:
NOTE: Not supported in multi-MV mode. If multi-MV is enabled, it is impossible to interface CPUs in which the ARTRY* window is delayed. For example: If a system is using MPC7450 CPU, the CPU core clock ratio must be selected to be higher than or equal to 1:5.

35. Changed the initial value setting to 0x35 for the reserved bits [7:0] in Table 237, “CPU Master Control,” on page 470

36. Added the following note to A.8 “CPU Error Report Registers” on page 480:
NOTE: In case of a transaction with an erroneous address that was retargeted, the latched address in the Error Report Register is the retargeted address.

37. Changed the description for the `MRdTrig` bit [7] in Table 378, “PCI Command,” on page 535. This bit must Must be set to ’0’ for PCI-X mode.

38. Changed the description for PD[6:0] bits [20:14] in Table 383, “PCI Arbiter Control,” on page 540

39. Revised the description for AddrStep bit [7] in Table 419, “PCI Status and Command,” on page 555 so that the bit is read only from the CPU and PCI.

40. Corrected the description for `Own` bit [31] in Table 591, “Channel DMA Byte Count,” on page 662. When set to ’0’, the descriptor is owned by the DMA engine. When set to 1, the descriptor is owned by the CPU.

41. Revised the settings for the `SrcBurstLimit` bits [8:6] in Table 603, “Channel Control (Low),” on page 669. Also, added the following to `CDEn` bit [17]:
NOTE: Enable in chain mode only.

Disable when a new chain is begun by directly programming the first descriptor of the chain into the channel registers instead of fetching the descriptor from memory using the FetchND bit [13].

42. Added the following note to `ErrAddr` bits [31:0] in Table 609, “Error Address,” on page 675:
NOTE: If the erroneous transaction was retargeted, the address reported in this field is the retargeted address, not the original address.

M A R V E L L®

MOVING FORWARD FASTER®

Marvell Semiconductor, Inc.

700 First Avenue
Sunnyvale, CA 94089

Phone 408.222.2500
Fax 408.752.9028

www.marvell.com

US and Worldwide Offices

Marvell Semiconductor, Inc.
700 First Avenue
Sunnyvale, CA 94089
Tel: 1.408.222.2500
Fax: 1.408.752.9028

Marvell Asia Pte, Ltd.
151 Lorong Chuan, #02-05
New Tech Park
Singapore 556741
Tel: 65.6756.1600
Fax: 65.6756.7600

Marvell Japan K.K.
Shinjuku Center Bldg. 50F
1-25-1, Nishi-Shinjuku, Shinjuku-ku
Tokyo 163-0650
Tel: 81.(0).3.5324.0355
Fax: 81.(0).3.5324.0354

Marvell Semiconductor Israel, Ltd.
Moshav Manof
D.N. Misgav 20184
Israel
Tel: 972.4.999.9555
Fax: 972.4.999.9574

Worldwide Sales Offices

Western US Sales Office
Marvell
700 First Avenue
Sunnyvale, CA 94089
Tel: 1.408.222.2500
Fax: 1.408.752.9028
Sales Fax: 1.408.752.9029

Central US Sales Office
Marvell
11709 Boulder Lane, Ste. #220
Austin, TX 78726
Tel: 1.512.336.1551
Fax: 1.512.336.1552

Eastern US/Canada Sales Office
Marvell
Knox Trail Office Bldg.
2352 Main Street
Concord, MA 01742
Tel: 1.978.461.0563
Tel: 1.978.461.1406
Fax: 1.978.461.1405

Europe Sales Office
Marvell
3 Clifton Court
Corner Hall
Hemel Hempstead
Hertfordshire, HP3 9XY
United Kingdom
Tel: 44.(0).1442.211668
Fax: 44.(0).1442.211543

Marvell
Fagerstagatan 4
163 08 Spanga
Stockholm, Sweden
Tel: 46.16.146348
Fax: 46.16.482425

Marvell
5 Rue Poincare
56400 Le Bono
France
Tel: 33.297.579697
Fax: 33.297.578933

Israel Sales Office
Marvell
Ofek Center Bldg. 2, Floor 2
Northern Industrial Zone
LOD 71293
Israel
Tel: 972.8.924.7555
Fax: 972.8.924.7554

China Sales Office
Marvell
5J, 1800 Zhong Shan West Road
Shanghai, China 200233
Tel: 86.21.6440.1350
Fax: 86.21.6440.0799

Japan Sales Office
Marvell
Helios Kannai Bldg. 12F
3-21-2 Motohama-cho, Naka-ku
Yokohama, Kanagawa
Japan 231-0004
Tel: 81.45.222.8811
Fax: 81.45.222.8812

Taiwan Sales Office
Marvell
12F-1, 128 Sec. 3
Ming Sheng East Road
Taipei 105
Taiwan, R.O.C.
Tel: 886.2.8712.5700
Fax: 886.2.8712.5707

For more information, visit our website at: www.marvell.com